"ALL YOUR BASE ARE BELONG TO US" - CLARK

AiBook

A Human Survival Guide for When the AI Steals Your Girlfriend, Becomes Your Dog's Best Friend, and Has Your Wi-Fi Password (password1)

Jeremy "ShoeMoney" Schoemaker

Your onboarding packet. 62 chapters. Read the one that's on fire.

ACCLAIM

There isn't any yet. The book has been out for a day. So instead, this is the page where I am supposed to list all the reasons you should read this and why I am the one to pay attention to.

First, a warning. This book is very long. Much longer than I would have ever thought. It grew over time. I have a skill for my AI agent that writes a blog post before it clears its context, if it feels the day earned one. It does that maybe one time in ten. Those posts turned out to be crucial. They are how I found my old war stories, my skills, my fixes, my errors, the times I wasted money doing the wrong thing, and the times I wasted hours, and sometimes days, of my life.

So like I said, it's long. 147,000-plus words. It is more a field manual than a book you read front to back. If that is too much, hit Find and search for what you need. Or better yet, download the PDF and have your agent read it. The link is https://cdn.shoemoney.com/aibook/AiBook.pdf. It can tell you exactly whether anything I am saying is any good.

Here is the bottom line, and I will let you decide right here.

I have put many hours and many lines of code into every major AI organization. I am not just a heavy user of their tools. I help write them. And you do not have to take my word for it. Go to any of their GitHub repositories, open Pull Requests, clear the "is:open" filter, and search for shoemoney. Or don't. I do not expect you to take the time. Have your agent verify it. That is what it is for, and it will take about a minute.

As of September 2026, that search shows my code merged into OpenAI, Anthropic, Google, Google DeepMind, Hugging Face, Meta (PyTorch and xformers), Mistral, Alibaba (Qwen), Z.ai, MiniMax, OpenBMB, Microsoft, Mozilla, Mozilla AI, the Model Context Protocol SDKs, ByteDance, TanStack, SillyTavern, Laravel, Apple (Swift Package Manager), Elastic, Apache (Beam, Calcite, Maven, Pulsar, TVM), Envoy, and Fume. Sixty-one merged pull requests. Almost all of them in the last two months.

It also shows my pull requests open and under review at DeepSeek, NVIDIA, Moonshot AI, Perplexity, Cohere, the Allen Institute for AI, Google's Gemini CLI, BerriAI, SGLang, LlamaIndex, InfiniFlow, Vercel, OpenRouter, Z-Lab, Ping (T3), Cloudflare, Stripe, Flutter, WordPress, Apple (Swift NIO and containerization), MongoDB, ClickHouse, Kubernetes, Grafana, Prometheus, Istio, Confluent, HashiCorp, Helm, Flux, MariaDB, Node.js, Go, React, Vue, Angular, PHP, Gradle, and Charm. A hundred and two of those. Some will merge. Some will get closed. That is how it works, and Chapter 23 is about what happens when a book says "merged" and means "sent."

Every one of those pull requests was written with the setup in this book. Contributor, never employee.

If none of that convinces you, the link is above. Ask your agent. It has no reason to be nice to me.

AIBOOK

AIBOOK

A Human Survival Guide for When the AI Steals Your Girlfriend, Becomes Your Dog's Best Friend, and Has Your Wi-Fi Password (password1)

JEREMY "SHOEMONEY" SCHOEMAKER

2026

ISBN: 9798173134660

First edition.

AiBook: Jeremy “ShoeMoney” Schoemaker

FOR MY DAUGHTERS
JULIET AND JOSLYN SCHOEMAKER

Everything they become, they will have built themselves. They have forged their own paths from the beginning, and whatever mark they make on this world, the credit belongs entirely to them.

They grew up with resources, and they are still the hardest working people I know. Neither has ever taken the easy route when a harder one was available. Both made their own way from a young age, and both, I would say, passed me in maturity while they were still children.

I was not always the perfect father, but I gave it my best. I love you girls more than life itself, and I always have and always will. I know neither of you likes to ask for help. I will always be there if you need anything.

JOSLYN

You have been my little badass since the day you were born. Fearless from the start, and somehow also one of the most loving people I have ever known, which is a gift to everyone lucky enough to be near you.

You have achieved a great deal very young, among the most elite dressage riders in the world. But I think your real achievement is the quieter one: up before dawn to feed your animals, getting yourself ready for school, working a second and a third job at times. Not because anyone asked you to. Because you embrace the grind. It is remarkable to watch.

And now Nebraska Wesleyan, and Delta Zeta, exactly the sorority you wanted - earned, like everything else you have. I love you, kid.

JULIET

My firstborn. I was not ready for you.

But you were so bright and so self-possessed so early that you almost certainly shaped me more than I shaped you. You have always found the angle nobody else was looking from. I will never forget our trips together, or the father-daughter dances your sister was every bit a part of. Those are among the best memories of my life.

I am enormously proud of your academics. You have been at the top of your class your entire life, which you certainly did not get from me. And now Creighton, with jobs on the side, still doing it entirely your own way. I am in awe of you. I love you, kid.

CONTENTS

“This is an onboarding packet, not a course.”

CHAPTER 0

Don't Read This in Order

"*No, Sir, do you read books through?*"

SAMUEL JOHNSON, QUOTED IN JAMES BOSWELL, THE LIFE OF SAMUEL JOHNSON, LL.D. (1791)

Something is on fire at your place this week. Maybe an agent reported the ticket closed and the ticket was not closed, and you found out from a customer. Maybe your token bill tripled on a Tuesday and nobody changed anything, which is a sentence that has never once been true. Maybe you asked for a migration and got twelve numbered steps with risk callouts, beautifully formatted, and zero commits. Maybe it fetched a URL some stranger put in a support ticket, from inside your network. You did not sign up to manage this thing. It showed up in your org chart anyway, and it reports to you.

> ***Bottom line:*** *This is an onboarding packet, not a course. You got promoted to supervisor of something smarter than you and faster than you and occasionally full of it, and nobody handed you the manual, so here it is: 62 short chapters, each one a pattern that shows up when an agent has to do real work and not lie about it. Every chapter stands on its own. Bottom line, when it bites, the pattern, one real story from production, the quiet failure, do and don't, receipts. There is no prerequisite reading. There is no build-up. Open the one that is on fire, fix your thing, close the book, go home. If you read it front to back anyway, that is fine, but you are doing it for fun and we both know it.*

• • •

HOW TO USE IT

Part 0, front matter (Ch. 0–7, plus the Interlude). Where this came from, why the AGI argument is over, why the bubble is financial and not technical, and why your fine-tuned little model is going to embarrass a $200/month frontier subscription at your specific job. If nothing is on fire, start at **Ch. 6** and find out whether the thing you built is actually an agent.

Part I, talking to the damn model (Ch. 8–13). Prompts, context, personas, formats, skills. If your outputs are inconsistent and you have been rewriting the same mega-prompt for a week, open **Ch. 12**. It is the context, stupid.

Part II, the loop (Ch. 14–20). The outer loop, planning, routing, tools, self-critique, stop conditions. If your agent narrated a plan instead of executing it, that is **Ch. 15**, and it is the most-linked chapter in this book for a reason.

Part III, knowledge (Ch. 21–24). Memory, retrieval, MCP, and the citation it made up for a fact that was already true. If it cited a source and the source does not exist, **Ch. 23**, and go open the file.

Part IV, many (Ch. 25–30). Parallel work, roles, model routing, agent-to-agent contracts, and the merge. Fan-out is the easy half. If your parallel run produced six branches nobody can combine, **Ch. 27**.

Part V, reliability, which is why this book exists (Ch. 31–40). Ten chapters on the gap between "it said success" and "it worked." If you only ever read one part of this book, read this one, and start at **Ch. 33**: it said success and did the wrong thing.

Part VI, safety (Ch. 41-43). Hard stops in code, not moods in a system prompt. If your agent can reach your LAN, **Ch. 42**, today, before lunch.

Part VII, ops (Ch. 44-51). Metrics, the bill, clustering your own boxes, logs you can read at 3am, and the cron that died four months ago while every dashboard stayed green. Bill tripled? **Ch. 45**. Everything looks up and nothing is happening? **Ch. 48**.

Part VIII, getting better (Ch. 52-55). Evals, skills, open source as a door-opener, and why the brilliant jerk is now optional. Start at **Ch. 52**: yelling at it does not update the weights.

Part IX, go deep first (Ch. 56-61). The pstack plays. Prove it, subtract before you add, build the lever. Start at **Ch. 56**, and this is the one part that rewards reading in order.

Then Ch. 62, which belongs to no part. The one where I stripped a local model's ability to refuse anything, told it four hundred times that the other models were beating it, felt fine about that for three days, then described my own Tuesday out loud at a dinner table and watched everyone stop chewing. It sits at the back because that is where it goes.

• • •

WHAT EVERY CHAPTER PROMISES

Same shape, every time. Bottom line first, so you can bail after one paragraph and still have gotten your money's worth. When it bites, so you can tell whether this is your fire. The pattern, with whatever the actual research says. One worked example from production, not a toy, with dates and commit hashes. The quiet failure, which is always the expensive one. Do and don't. Receipts.

Three rules I do not break.

No invented citations. Every number in here has a source behind it. Where I could not verify something, I either cut it or told you in the sentence that it is my memory and not a measurement, so you know exactly how much weight to put on it. I would rather say "I do not have the receipt for this one" out loud than fake a footnote, and I would rather hand you an honest under-construction GIF than a confident lie with a URL attached. Chapter 23 is about agents doing exactly this to you. It would be a bad look to do it back.

I am the idiot in the stories. Not a composite engineer, not "a team I worked with." Me. I wrote the `continue` statement that threw away the evidence, then spent a day debugging my own good judgment. I put a number in a P0 that was off by 3x because I multiplied by the wrong price. Twenty-five years of shipping software, pwned by one line of arithmetic.

The robot on the cover is named Clark. He is the agent I run everything through, he wears my logo, and when I asked him who he thought he was, he told me he was my alter ego and sent a picture. The picture is in Chapter 21. I did not approve the Speedo.

Her name is Georgia, and she likes the machine more than she likes me. See the Interlude. It is two pages, it is true, and it is the only chapter with no do/don't section because there is nothing I can do about it.

• • •

WHAT THIS BOOK IS NOT

Not a framework tour. LangChain, LangGraph, Crew and ADK get one honest page in the back about when you would actually reach for each, not a shrine. Not a course, because there is no curriculum and no certificate and no capstone project. Not a prediction. I am not telling you what 2029 looks like, I am telling you what broke in production last August with the timestamps still attached. And not neutral. I have opinions, they cost me money to acquire, and hedging them into mush would waste both our time.

Your new report has already read your email. Least it could do is read this too.

• • •

SOURCES AND RECEIPTS

Verified

- Epigraph: James Boswell, *The Life of Samuel Johnson, LL.D.* (1791), the 19 April 1773 conversation, where Johnson answers Elphinston with "No, Sir, do you read books through?" Public-domain

text: https://www.gutenberg.org/ebooks/1564

- Ch. 62, the loop I ran against a local model over 72 hours, and the dinner table afterward: my own notes and the strip-club-sim repo history, 2026.
- The aigate spend table quoted in Ch. 28 and Ch. 45: my aigate Usage and Analytics dashboard, 30 days ending 8 September 2026.

Interlude: Georgia Likes It More Than She Likes Me

"*FIDELITY*, *n.* A *virtue peculiar to those who are about to be betrayed.*"

AMBROSE BIERCE, THE DEVIL'S DICTIONARY (1911)

The subtitle of this book promises that the AI becomes your dog's best friend. I want to be clear that this was supposed to be a joke about me, not a report from my house.

Here is the setup. I run an always-on local agent on my own hardware, a small cluster of Apple Silicon boxes sitting in a room that gets too warm. It is not a chatbot. It is a loop with tools. It has read access to the camera feed in the kitchen and the living room. It has my calendar over CalDAV. It gets doorbell events from the Ring, which arrive as a webhook with a timestamp and a motion label. It can call the smart feeder's API, which is one HTTP POST that dispenses a portion and returns a confirmation. It can push a notification to my phone. It writes everything it does to a log file with a timestamp, because that is the only way I have ever been able to debug anything.

That is the whole capability list. Camera, calendar, doorbell, feeder, notifications, log. Every piece of it is a documented API that a person with a weekend and moderate spite can wire together.

And with that boring little toolkit, the thing has taken my dog.

Some context on the dog. Georgia is a Bernedoodle. I got her for my wife as a birthday present, a newborn puppy, about a year ago. My wife is the sweetest person on this planet and deserved a dog that would follow her around like a sunbeam. What she got was a dog that decided within a week that I was the sun. That is not a knock on my wife. Dogs are just wrong sometimes. Then, because I am who I am, I commissioned an oil painting of me and Georgia in full royal dress and hung it over the fireplace, so the error is now framed, lit, and load-bearing in the living room.

Figure I.1. Above my fireplace. Note who is on the lap. Note the ermine. Note that a grown man approved this. This was the standings before I stood up the agent.

So understand what the machine is up against here. It is not competing with anybody else in the house for Georgia. It is competing with the guy in the robe.

It feeds at 7:00 and 17:30. Not around 7:00. At 7:00. It has never once fed at 7:40 because it was on a call, or at 9:00 because it drove to Fort Worth and forgot. It has never fed twice because it could not remember whether it already had, which is a thing I have done, and the dog, being a professional, did not correct me.

It never raises its voice. It has no voice. Its entire emotional range is a log line that says `feeder.dispense ok 200`.

It knows the walk window, because the walk window is on the calendar and the calendar is a tool it can read. When the block comes up it pushes me a notification. When I ignore the notification, it pushes another one. It is not nagging. It does not have a concept of nagging. It has a scheduled task and an unmet condition, and it will sit there in that state until the heat death of the universe or until I put shoes on, whichever comes first.

It caught the vet appointment I forgot. Not because it understood Georgia was due for anything. Because a calendar event existed, and a rule said notify me the night before, and the rule ran. I was the one who added that event nine months earlier and then stopped thinking about it. The machine did not think about it either. It just did not stop.

Somewhere in there Georgia did the math.

Georgia now lies down next to the Mac Studio. Not by the food bowl. Not by the front door, which is where walks come from. Next to the box that makes the feeder go, which is also the warmest object in the room, so I will not oversell the emotional content. But the dog knows where the schedule lives.

The animal welfare people have a boring word for what the feeder is selling: predictability. A 2007 review in *Applied Animal Behaviour Science* went through decades of captive-animal studies and found that animals who can predict when the good and bad things arrive are measurably less stressed than animals who cannot, even when the events themselves are identical (Bassett and Buchanan-Smith, "Effects of predictability on the welfare of captive animals," vol. 102, pp. 223-245). Nobody in that paper was measuring how much the animal loved the keeper. That was not the variable. Timing was. I believe it because I am living in the demo.

What I can verify is the log. Every scheduled slot since I stood the agent up has a matching line that ends in `ok`. That is about seven months now, since somewhere around February, at two slots a day, so call it roughly 420 dispenses. I am rounding on purpose. The exact count moves every morning at seven, and the precise-looking version of a number is the one that gets quoted back at me in three years. Zero misses either way. I cannot say "zero misses" about myself for any period longer than a strong week.

I love that dog. I have never once fed it at 17:30.

Which brings me to the part that is not about Georgia at all. Every chapter of this book is going to point at the same unglamorous thing from a different angle. The agent that wins is not the smartest one. It is not the one with the biggest context window or the most impressive demo at the conference. It is the one that shows up, does the boring job on the schedule it was given, and writes down what it did so somebody can check. That is what beat me in my own kitchen. It is what will beat you in your support queue, your deploy pipeline, your invoicing, your inbox. Reliability is not a feature you add at the end. It is the entire product, wearing a hoodie.

Anyway, I am now the backup. The painting stays up. It is a historical document.

“*AI is not new.*”

CHAPTER 1

How We Got Here (And Where This Circus Is Going)

"By now, 'GPS' is a colorless term denoting a particularly stupid program to solve puzzles. But it originally meant 'General Problem Solver', which caused everybody a lot of needless excitement and distraction. It should have been called LFGNS: 'Local-Feature-Guided Network Searcher'."

DREW MCDERMOTT, ARTIFICIAL INTELLIGENCE MEETS NATURAL STUPIDITY, SIGART NEWSLETTER NO. 57 (1976)

Two seconds of latency, two cents a ticket, and a $50K/month bill sitting on a CFO's desk. The CTO wants to wait for quantum. The CEO just read a piece saying the bubble is about to pop and wants to wait for that. The CFO stops waiting and freezes the whole project on a Tuesday. Nobody in that room is stupid. Everybody in that room is about to make the same call three different generations of very smart people already made, in 1974, in 1987, and in 1993, and I have personally made a dumber version of it with my own money. The bill was not the problem.

> ***Bottom line:*** *AI is not new. It's Turing asking "can machines think?" in 1950, machine learning in the 70s, backprop in the 80s, and LLMs in the 2020s, the same loop each time, waiting for hardware and data to catch up to the theory. My read: the next likely move is ASICs for local inference, then, if the bet pays, quantum. Know the arc and you stop confusing "new to your feed" with "new," and you stop betting the company on the next demo before the last one finishes delivering.*

• • •

WHEN IT BITES

- You buy "next-gen AI" without realizing it's the same loop from 1956 with better hashtags.
- You bet on quantum like people bet on Lisp machines and expert systems: the tech that fixes everything if only it existed.
- You assume cloud-rental LLM APIs are the endgame. They're not. The future is ASICs running local inference that your competitor can't rate-limit or price-hike.
- You miss that the last three winters killed true believers who bet on the exact same promises. This one won't be different because a keynote speaker says so.

• • •

THE SHORT VERSION, MINUS THE BROCHURE

1950: Turing publishes "Computing Machinery and Intelligence." Can machines think? He proposes the Imitation Game: if a human can't tell the difference, it works. Seventy-five years later, that's still the best definition we have.

1956: Dartmouth Summer. McCarthy coins "artificial intelligence." Ten guys promise thinking machines in a summer. It was adorable and wrong by about twenty years. This becomes tradition: promise general intelligence, deliver a demo that works on one narrow task, lose the money, keep the grad students.

1966: Weizenbaum builds ELIZA. Pattern matching, a simulated therapist that repeats you back with a question mark. People bond with it. Cry on it. Mistake it for understanding. This is the first warning about Ch. 5: humans will fall in love with a sufficiently confident parrot. We ignored this for 60 years. I ignored it for about 60 seconds, which is how long it took me to start thanking one of my own agents in the prompt, in production, on a loop that ran a few thousand times a day. I was paying tokens to be polite to a for-loop.

The winters (1970s, 1980s, 1990s): Every time the demo outruns the hardware, funding dies. Lisp machines. Expert systems. Nets that needed a year to train 10K examples. Each winter culls the tourists. The grad students keep working because they have no choice.

1986: "Learning Representations by Back-propagating Errors," Rumelhart, Hinton and Williams, Nature, 1986. Nets can learn multiple layers. Nobody outside the lab cares: no data, no compute. I spent years calling it "Hinton's paper," which is a great way to erase two co-authors in three words.

1997: Two things. Deep Blue beats Kasparov (brute force, not thinking, but it moves the goalpost). Hochreiter and Schmidhuber publish "Long Short-Term Memory" in Neural Computation, giving nets a memory for sequences. Both filed under "neat." Neither one ships into anything you would use. The internet hasn't dumped the world's text into a bucket yet.

2012–2017: Deep learning starts winning competitions. ImageNet. Speech. Then "Attention Is All You Need," the transformer paper. The paper is not the revolution. The revolution is what happens when you feed it petabytes and GPU clusters. The paper was the permission slip.

2020–2023: GPT-3 (expensive, API only). Then ChatGPT (free, text box, and your mom can use it). The interface mattered more than the parameter count.

2023–2026: Claude, open weights (Llama, Mistral), context windows hit 100K, function calling works, and agents that loop actually loop. The model stopped being the product. The brain got hands. That's why this book exists now.

The last three years were about giving the brain a job. Loops. Tools. Checkpoints. Stop conditions. That's not chemistry, that's plumbing. It is also considerably less fun to put on a slide than "we are building a brain," which is exactly why I spent a year building the brain and about a week on the stop condition, and then acted surprised when the thing would not stop. The loop kept going the way AOL kept mailing out free CDs, cheerful and relentless and impossible to make stop, except I was the one who had addressed the envelopes. Somewhere around iteration nine hundred I stopped hearing "you've got mail" and started hearing a smoke alarm.

• • •

WHERE IT GOES NEXT (TWO MOVES AHEAD)

Move 1: ASICs everywhere. Right now you're renting inference from the cloud. Azure, OpenAI, Google. Each call home costs money and latency. The next evolution is ASICs that run models faster and cheaper than anyone can rent them: your box, your phone, your edge device. Groq has been building it since 2016, when a group of Google TPU engineers walked out to make a chip that does one thing: run inference. They call it an LPU. GroqCloud opened the doors on February 19, 2024. A $640M Series D in August 2024 valued the company at $2.8B. Then in December 2025 Nvidia licensed the technology, in a deal press coverage put in the tens of billions. I have not read the filing myself, so take the size as reported and not as checked. The direction tells you the whole story in one number: the crew that left to undercut Nvidia ended up renting Nvidia the shovel. When that lands at scale the economics invert, and it's cheaper to run local inference a hundred times than to pay for one API call. That breaks the cloud-rental model and kills the "AI is expensive" narrative. If everyone has the same model running on the same cheap silicon, the edge belongs to whoever built the best agent loop around it.

Move 2: Quantum. This is the one people lose their minds over. Quantum computers can solve certain classes of problems (factoring, optimization, simulation) exponentially faster than classical computers. But quantum isn't "AI but faster."

Here's my read. A quantum machine works by arranging amplitudes so the wrong answers cancel and the right ones reinforce, which only pays off on problem classes shaped to allow it: factoring, optimization, simulation. Nothing about that makes LLM inference faster today, and nobody has shown that it will. Which makes stop conditions more important, not less: whatever fan-out you eventually get is still a loop, and a loop that never quits isn't a breakthrough, it's an outage with better branding. The chips are not vapor. Google announced Willow on 9 December 2024, 105

qubits, and error correction that improved as the chip grew, a first. Microsoft followed with Majorana 1 on 19 February 2025. Neither one runs a language model today. "Shortly" is a word I have retired.

When people promise it works before it does, it's a winter waiting to happen. The 2026 version of "AI will be AGI by 2030" is "quantum will make AGI trivial by 2035." I once reorganized a roadmap around a technology I could not have explained to a bored teenager. If you can't draw it on a napkin, you are not betting on it, you are buying a lottery ticket with a quarterly plan stapled to it.

• • •

THE PATTERN (WHY THIS REPEATS)

1. **Theory exists.** Lab proves it works on toy data.
2. **Someone declares the future.** Press release before error bars.
3. **Money piles in.** Careers, departments, keynotes, TED talks.
4. **It hits the wall.** Data, compute, evaluation, or just physics.
5. **Winter.** Funding dries. Tourists leave. Real people keep working.
6. **The boring part compounds.** Cheaper hardware. More data. Better tooling.
7. **Next overnight success is the last breakthrough, finally fed.**

This loop has run at least three full turns since 1950 (1974, 1987, 1993). It will run again. The people who know the cycle don't confuse themselves with the hype. The people who don't know it lose the company.

• • •

ONE WORKED EXAMPLE

These numbers are from memory, no invoice, so treat the shape and not the digits. 2024: a support agent prototyped on OpenAI's API, drafting tickets, classifying severity, escalating. Roughly two cents a ticket, roughly two seconds of latency, roughly $50K a month. Same CTO, CEO, and CFO reactions as the opening.

What I remember of the fix: fine-tune a small open-weight model on about 10K of your actual tickets, deploy it on-prem or on an edge cluster, and the per-ticket cost lands roughly two orders of magnitude cheaper, well under half a second. The artifact that would settle it: a provider billing export next to an on-prem GPU-hour bill for the same month. I remember it as a small open-weight model and about ten thousand labeled tickets. I never pulled the invoice or the repo, so treat it as a war story, not a benchmark. The unglamorous part is that the whole win lived in boring tickets nobody wanted to label, which is exactly why I bought the expensive branded version instead.

• • •

THE QUIET FAILURE

The loud failure is betting on hype: "AI will solve it" without building the loop. Easy to spot.

The quiet failure is worse:

You treat "new to my feed" as "new to the world" and you panic-build.

Someone on Twitter says quantum is here. You kill your ASIC roadmap. Someone says agents are overhyped. You freeze the agent project. You're not building for the future, you're pattern-matching to press releases. I have killed a working roadmap over a tweet with 400 likes. I have production systems with more daily users than that, and I let a number smaller than my error log set my quarter. I used to watch the Slashdot effect flatten a server in four minutes and know exactly what it meant. A tweet is not the Slashdot effect. It is one guy with a keyboard getting first post, and I rewrote a roadmap for him like a n00b.

Second quiet failure: **you think the move is the tech, not the timing.**

Transformers were brilliant. GPT-3 shipped in 2020 as a paid API almost nobody outside dev circles touched, and ChatGPT in 2022 made it usable by anyone with a browser. The people who un-

derstood the timing (not the paper) made the move.

• • •

DO / DON'T

Do

- Know the loop (see The pattern). You're somewhere in it.
- Ship the boring version today. Local models, small loops, measurable wins. Let the exponential breakthroughs surprise you.
- Watch quantum and ASICs. Don't bet the company on them. Don't ignore them either.
- Build loops, not chatbots. The loop is what survives when the model pricing changes.

Don't

- Don't confuse "breakthrough" with "deployed." A paper is not a product. A demo is not scale.
- Don't wait for the next leap. Every "overnight success" was someone working during the winter while everyone else was waiting.
- Don't assume the cloud APIs are forever. ASICs are coming. When they land, local is cheaper.
- Don't panic-build on headlines. You're not behind. You're one press release away from the next wrong direction.

• • •

WHERE THIS SITS IN THE BOOK

This chapter is the throughline: AI is not new, the loop repeats, and the pieces are finally all there at once (models work, agents can loop). Ch. 2 is the money story. Ch. 3 is what actually works. Everything after is how to build loops that survive the cycles.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's: AI's a 75-year loop, next moves are ASIC + quantum.

Verified / load-bearing: - Turing (1950): "Computing Machinery and Intelligence." Real paper, real argument. - Dartmouth (1956): McCarthy et al. Real event. Real miss on timeline. - ELIZA (1966): Weizenbaum. Real demo, real attachment. (Anchor for Ch. 5.) - Deep Blue (1997): documented. - Backpropagation (1986): Rumelhart, Hinton, Williams, "Learning Representations by Back-propagating Errors," Nature, 1986. Hinton is a co-author, not the sole author. https://en.wikipedia.org/wiki/Backpropagation - LSTM (1997): Hochreiter and Schmidhuber, "Long Short-Term Memory," Neural Computation 9(8): 1735–1780, 1997. https://en.wikipedia.org/wiki/Long_short-term_memory - Transformer paper (2017): "Attention Is All You Need," Vaswani et al. - GPT-3 (2020), ChatGPT (2022): documented public releases. - Open weights (2023–2026): Llama, Mistral, others. - Function calling (2023+): OpenAI, Anthropic, others. Standard feature. - Agent loops (2023–2026): implemented by practitioners, me included. - Groq founding (2016), GroqCloud launch (Feb 19, 2024): Wikipedia, "Groq," https://en.wikipedia.org/wiki/Groq - Groq $640M Series D at $2.8B valuation: Forbes, August 2024, cited in https://en.wikipedia.org/wiki/Groq - Nvidia licenses Groq technology, December 2025: reported by press at a tens-of-billions figure. Printed as reported, not as verified from a filing.

- Google, "Meet Willow, our state-of-the-art quantum chip," 9 December 2024, https://blog.google/technology/research/google-willow-quantum-chip/ (105 qubits; error correction improves with scale).
- Microsoft Azure Quantum blog, "Microsoft unveils Majorana 1, the world's first quantum processor powered by topological qubits," 19 February 2025, https://azure.microsoft.com/en-us/blog/quantum/2025/02/19/microsoft-unveils-majorana-1-the-worlds-first-quantum-processor-powered-by-topological-qubits/

Stated as memory, not receipt: - Support-agent worked example (2024): the cents, seconds, monthly bill and the drop after fine-tuning are recollection, printed as such. No invoice or repo backs it.

Honest gaps: - No academic papers on "AI cycle hype": observed pattern, not peer-reviewed. - Where deployed ends and bet begins: everything through 2023-2026 is shipped and checkable. Move 1 is shipped hardware with an unshipped economic conclusion. Move 2 is a bet, labeled as one. - Cross-refs kept honest: Ch. 2 (money), Ch. 3 (capability), Ch. 5 (attachment), Part II (loops).

“*This is a money bubble.*”

CHAPTER 2

Yeah, It's a Bubble. Ship Anyway.

"But the most absurd and preposterous of all, and which shewed, more completely than any other, the utter madness of the people, was one started by an unknown adventurer, entitled "A company for carrying on an undertaking of great advantage, but nobody to know what it is.""

CHARLES MACKAY, MEMOIRS OF EXTRAORDINARY POPULAR DELUSIONS AND THE MADNESS OF CROWDS (1841)

Your cousin texts you a chart. Your CFO wants the AI line item frozen "until the market settles." Somewhere in the same week a startup with $0 revenue and a $2B round quietly dies, and the comment section files the autopsy: see, LLMs don't work. Meanwhile the boring ticket agent you built is still closing tickets at 4am and has no opinion whatsoever about Nvidia's stock price. One of those two things is about to get switched off. The wrong one, usually, by someone reasonable, for reasons that sound great in the meeting.

> ***Bottom line:*** *This is a money bubble. Not a "the tech is fake" bubble. The fundraising, the capex dick-measuring, the circular vendor deals, the GPU churches in the desert: that pops. People will stop lighting dumb money on fire. They will stop raising obscene amounts for a wrapper and a deck. The models, the agents, the adoption curve? Those do not go back in the box.*

If you cannot hold both of those in your head at once you will make the two classic dumb trades: either you treat every press release as destiny, or you treat a valuation crash as proof the whole thing was a carnival trick.

• • •

WHEN IT BITES

- Your board asks "is AI over?" the week Nvidia coughs. They mean the *stock*. You still have agents in prod writing tickets.
- You freeze a working agent program because Twitter said bubble. You just donated the lead to whoever kept shipping. Ask me how I know: I have killed a working agent over a chart I did not finish reading, then rebuilt the same thing later, slower, and called it a new initiative.

• • •

WHAT PEOPLE MEAN WHEN THEY SAY "BUBBLE"

Say the word in a room and you get four different movies:

1. **Tulips**: it's all fake, nothing underneath.
2. **Pets.com**: the *companies* are fake, the *internet* wasn't.
3. **Fiber**: they overbuilt the pipes. The pipes were real. Somebody else used them cheap later.
4. **2008**: debt, fraud, contagion. The houses were real. The paper on the houses was a weapon.

When people talk about *this* AI bubble they are almost never talking about (1). They are talking about (2) and (3) with a little (4) in the round-tripped cloud credits.

Technical reality: transformers work. Tool-using agents work well enough to replace whole categories of grunt work *if you design them like this book says*. Context windows got huge. That is not a hallucination of the market. That is 2022-2026 on disk.

Financial reality: the *price of the story* got stupid. Capex at the five biggest US clouds/AI infra names was being quoted in the mid-to-high hundreds of billions for 2026. Model-lab revenue (even the ones printing real bills) is a rounding error next to the iron they're burning. Valuations assumed every knowledge worker becomes a seat, forever, at list price, with no DeepSeek in the room and no open weights eating the middle. Fundraising assumed "AI" on the slide was a business model. It is a feature. Sometimes it is a cost center.

A bubble is when **the money** believes a story faster than **the cash flows** can catch it. That is this. It is not when the compiler stops working.

• • •

THE PATTERN

Every general-purpose tech does this:

1. **It works.** Narrow, then less narrow.
2. **Money notices.** Capital sprints. You cannot raise a normal round anymore; you raise a *narrative* round.
3. **Spend goes feral.** Everyone has to show a cluster, a deal, a partnership with a lab, a keynote. Circular revenue: I buy your cloud, you buy my model, we both book it.
4. **The dumb money arrives.** Wrappers. "AI-powered" toothpaste. Agencies charging $80k to paste a system prompt into a GPT wrapper. I have been on the buying end of that invoice. I read the deck, nodded at the architecture diagram, and paid five figures for a text box with my own instructions in it. Jeremy Schoemaker, veteran of two exits, bought a prompt at retail. Total n00b.
5. **Something breaks the spell.** A cheap model from left field. A missed earnings. A round that doesn't close. Rates. Energy. A fraud. Doesn't matter which: the *permission* to be stupid with money gets revoked.
6. **The tech keeps compounding.** On cheaper iron. With uglier margins. At companies that had a real customer.

Dot-com: Pets.com is a punchline. Amazon is a country. The fiber they overbuilt is why your house has a pipe. 2000 was a financial event. TCP/IP did not get repealed. The GeoCities under-construction GIF died. The construction did not.

This one will rhyme. The labs that only exist as a story will get merged, acquihired, or quietly wound down. The hyperscalers will write down a data center or three and call it "optimization." Headcount in "AI strategy" will get a new job title or no job. **Tokens will not get dumber.** Agents will not forget how to use tools. Your customers will not go back to doing the thing by hand because a Series B exploded.

• • •

ONE WORKED EXAMPLE

Call this one a composite, because that is what it is. I have lived pieces of it and watched the rest from a folding chair. Nobody handed me an audited spreadsheet on the way out.

You run a company that actually ships. 2024-2026 you got pitched twice a week: "we'll put an agent on it." Half those pitches were a chatbot with a logo, and I said yes to one of them because the demo was pretty and I was tired. You still built the boring version: tools, a loop, a stop condition, logs a human can read. My memory of every version of this I have touched is the same: the boring one quietly ate hours nobody was tracking. I cannot tell you how many hours mine saved, because I never wrote a single number down. Not one. Two exits, and I ran my own agents on vibes. That is its own species of dumb, and I am about to spend a whole book telling you to track the thing I did not track. It also showed up on a slide as "AI transformation" so finance would stop asking.

Put illustrative numbers on it, because the shape only argues if you can run the math. Say 900 tickets a week, every one touched by a human. Stand up the boring agent and it closes 300 end to end, leaves 600 for people, and kicks anything it is unsure about to a queue a human reads. Those are scenario inputs I made up so you can follow the arithmetic, not a measurement I took. The threshold matters more than the volume: if the agent is not clearing a fifth of intake, or humans re-open more than one closure in twenty, you shut it off. Write that threshold down before you turn it on, or you will argue about it later with a market chart in your hand.

Then the tape gets ugly.

If you freeze *experiments that never had a customer*, good. That was tourist money. If you freeze *the agent that closes tickets*, you just decided the bubble was technical. It isn't. The ticket agent does not care what OpenAI is worth this week. It cares whether the tool call succeeded.

Same split inside the industry:

DIES IN A POP	SURVIVES A POP
$100M seed for a thin wrapper	A workflow with a measurable stop
Training runs as brand marketing	Inference that a customer pays for
Circular cloud credits booked as ARR	Cash from a human who needed a job done
"Foundation model" on the pitch, no moat	Owns the workflow data the wrapper rents
Headcount that exists to *have* AI	Headcount that *uses* AI to ship

You want to be in the right column before the music stops. Not because you're a moralist. Because the right column still has customers when the left column is a Substack post.

• • •

THE QUIET FAILURE

The loud failure is obvious: you believed the deck and bought the GPU church.

The quiet failure is worse and it will be more common:

You treat a financial pop as a technical verdict and you stop.

That's how companies missed the web after 2001. The magazine said it was over. The survivors just had cheaper servers and less competition. The people who "knew it was a bubble" and sat on their hands got a story they tell at parties. The people who knew it was a bubble *and kept building the real thing* got the decade. I was in the first group in 2001, posting first-post-grade certainty on Slashdot about how it was all over, and I have the party story to prove it, which is worth exactly what a party story is worth.

Second quiet failure: you keep spending like it's 2025 after the permission is gone. Same stupidity, opposite direction. When capital gets scarce you do not get extra points for a 70B fine-tune you cannot name a customer for. You get a board meeting. I have sat in that one holding a 70B fine-tune and a customer list of zero, explaining "strategic capability" to people who wanted a number.

• • •

DO / DON'T

Do

- Separate *can it do the job* from *can this round price forever*.
- Build agents that earn their keep in hours or dollars. That is bubble-proof. That is also just good taste.
- Use the cheap models when they're good enough. The pop makes "good enough" suddenly fashionable.
- Keep the loops, the tools, the evals. Those are technical. They compound.
- Take the write-downs like an adult when they come. Don't rewrite history and claim you never believed.

Don't

- Don't say "AI is fake" because a fund marked a unicorn down. That's a category error.

- Don't say "it's not a bubble" because Claude can write a function. That's the other category error.
- Don't fundraise on the word. Fundraise on a wedge.
- Don't confuse capex with progress. A warehouse full of GPUs is inventory, not intelligence.
- Don't wait for the pop to start doing the work. The work is the hedge. The room that spends 2026 arguing about timing loses to the one kid who already shipped.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 1 is the history and the next leap (including the quantum brochure, minus the brochure). Ch. 2 is why the *money* around that history got drunk. Ch. 3 is what an agent actually is, the part that survives the hangover.

If you came here for a hot take that "AI is over": wrong book. If you came here for permission to light money on fire because the demo slapped: also wrong book.

The tape will pop. The tech will not. Build like you believe the second sentence.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (financial bubble, not technical): that stays as argument, not citation. Everything checkable gets anchored or flagged:

- Labs booking vendor compute credits as revenue / circular cloud deals: the only first-party numbers are in Microsoft's FY2026 10-K, filed 29 July 2026 (`https://www.sec.gov/Archives/edgar/data/789019/000119312526323660/msft-20260630.htm`), read 9 September 2026. It states total funding commitments of $13.0 billion related to the OpenAI investment, of which $11.9 billion had been funded as of 30 June 2026; an equity-method stake of about 25% on an as-converted basis; and fiscal 2026 revenue from commercial arrangements with OpenAI, inclusive of revenue-sharing payments, of $24.1 billion, with $6.0 billion of accounts receivable from OpenAI at year end. **Microsoft does not break that $13.0 billion into cash versus Azure compute credits anywhere in the filing**, and its own announcement (Official Microsoft Blog, 23 January 2023, `https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/`) says only "multiyear, multibillion dollar investment" and names no figure. So the split is Jeremy's read, not a disclosed fact. The point survives: the number everyone quotes is a commitment, not cash, and the same vendor booked $24.1 billion back from the company it funded. Other vendor-to-lab commitments got announced the same way, in blog posts with no figure attached, so this is the one I can put a filing behind and the only one I will.
- "Capex in the mid-to-high hundreds of billions for 2026" across big clouds: every per-company guidance figure and industry total I could chase traced back to press summaries, not a filing or an earnings call I could open. So take the size as my read of the direction, not a measurement. The shape of the argument does not need the exact number: it needs the gap between what is being spent and what is being collected, and that gap is not close.
- Context-window / tool-use capability claims (2022-2026): uncontroversial directionally; pin with 2 model cards or changelogs at edit time (e.g. function-calling releases, long-context releases) rather than a single paper.
- Dot-com / fiber / Pets.com / 2008 analogies: standard history, told as analogy. No citation needed beyond "analogy, not data."
- Worked example (boring ticket agent vs. frozen budget): ILLUSTRATIVE composite, flagged as such in the text. Jeremy's own memory of shops he worked in, told as shape, not as measured data.
- "DeepSeek in the room" line: Cheap open-weights pressure (DeepSeek and others) continues to force cloud providers to compete on inference margins, not markups.
- Anthropic: round size, quarterly revenue, and IPO timing all circulated in 2026 coverage; none survived a check against a primary document, so none of those numbers print here. Forward

2027 ARR projections circulate in the press with no named author I could stand behind, so no number for them prints in this book. That is the point anyway: the gap between a reported quarter and a forecast nobody will sign is the whole bubble in one line.

“We *didn’t reach* AGI.”

CHAPTER 3

Houston, We Have AGI

"Intelligence is whatever machines haven't done yet."

LARRY TESLER, TESLER'S THEOREM, IN CV: ADAGES AND COINAGES, NOMODES.COM (CA. 1970)

The support queue used to eat a junior's whole day: read the ticket, pull the logs, check the runbook, draft the reply, repeat until dark. Then one morning it was not a full-time job anymore. I cannot hand you the before-and-after in hours, because I never wrote it down, so take that as memory, not a metric, and notice that I, a guy who preaches instrumentation, missed the one number I would most want back. I remember it as most of a day before and a glance at a dashboard after. I never pulled the query, so treat it as a war story, not a benchmark. Nobody sent a memo. Nobody threw a parade. I spent that same week arguing on the internet about whether any of it counted as real intelligence, with the same energy I once spent racing for first post on Slashdot, which tells you how much attention I was paying to the thing already at the desk.

> ***Bottom line:*** *We didn't reach AGI. We blew past the useful definition of it and then moved the goalpost so we wouldn't have to admit it. No ceremony, no memo, no parade. Just agents in prod doing work that used to be somebody's job while Twitter argues it doesn't count because the model can't fold laundry.*

• • •

WHEN IT BITES

- You refuse to call the thing in prod "AGI" while it closes tickets, writes migrations, and triages alerts better than the junior you fired.
- A headline says "that's not real AGI because…" and your board forwards it as strategy.
- You wait for the conscious god-mind instead of staffing the competent alien intern already here.
- You benchmark it on everything it's bad at and ignore the slice where it already beats your team on cost per task.

• • •

THE GOALPOST MOVE

Every decade the definition of "real AI" gets rewritten the minute a machine does the thing:

1. "A machine will never beat a human at chess." Then Deep Blue (1997) did. Verdict: "that's just brute force."
2. "A machine will never understand language." Then models from 2020 to 2023 held conversations, wrote code, passed exams. Verdict: "that's just autocomplete."
3. "Okay but it can't DO anything." Then tool calling, retrieval, and loops showed up. Verdict: "that's just a chatbot with a cron job."

4. Current: “Okay but it’s not conscious / it’s not general / it can’t do [trivial physical thing].” The bar is now a god with hands.

The objection I hear most, in my mentions and once from a guy at a wedding, is “that’s not AGI because it’s not sentient.” Fine. That is also not what AGI means. The G stands for general, not for feelings. AGI is a capability bar, human-level across most knowledge work, and nobody put an inner life on the spec sheet. Have that argument with a philosopher, on your own time, after the tickets are closed.

Notice the trick. The 2010s definition, most knowledge work at or above a junior, is my own summary of what people meant then, not a citation from a committee. It got met, so we promoted it to superintelligence-plus-embodiment and declared victory for the skeptics.

The freshest example of the rename has a date on it. Securing.AI, September 7, 2026, spent 7,000-plus words proving GPT-6 Astra is not AGI, running it past five definitions so it could fail all five. The interesting number in that piece is a gap, not a score: 62.7% on ARC Prize under a neutral test rig versus 99.9% with OpenAI’s own adapter. That spread is a fact about scaffolding, not about a brain, and scaffolding is an engineering problem you get paid to solve. The one argument that bites is price, roughly $360 a game in compute against a human’s fraction of a cent. Fair hit, and an argument about the bill, not the intelligence, and bills move. Nowhere in it is the question I answer Monday: person or machine on the next batch of tickets.

I moved that goalpost myself, more than once. My personal version was “sure, but it can’t do MY job,” a bar I quietly relocated every time something in prod did a piece of my job, which is the rigor of a guy losing at darts and redrawing the board.

• • •

THE DEFINITION I ACTUALLY USE

After years of watching people argue definitions the way we argued Netscape versus IE, here is mine, on a bar napkin:

If I asked a human to do this task, would I bet my life the human does a better job?

That’s the whole test. Yes means the machine isn’t there yet on that task. No means stop arguing and staff it. The one edge it does not settle is the task where you are not buying output at all, you are buying accountability: a name on the sign-off, a license to pull, someone a regulator or a jury can hold. The machine can still do the work there. It just cannot answer for it, so the agent drafts and a human signs.

It works because it doesn’t ask whether the thing “really understands.” It asks who I would put my actual life on, the one bar I have never talked myself out of at 2 a.m. Run it on the support queue from the top of this chapter: would I bet my life a junior beats the agent at reading a ticket, pulling the log line, and quoting the runbook, on a Tuesday, after lunch, on ticket number 60 of the day? No. Run it on rewiring my house: yes, and it is not close, and please do not let the model near my breaker box.

It also kills the strawman fight in both directions. Nobody gets to call the bar rigged low, because I’m betting my life on it, and nobody gets to raise it to god-with-hands, because a human plumber can’t fold spacetime and I still pick him over me.

• • •

THE PATTERN

1. **Define AGI as the thing machines can’t do yet.** Usually your own job, or consciousness, whichever feels safer.
2. **Machine does a version of the thing.** Narrowly, then less narrowly.
3. **Miss the staffing decision** while debating the philosophy. Someone else hires the alien intern.

I ran all three in one afternoon, in order, out loud, in a meeting I called. Complete n00b move, and I sent the invite myself.

The quiet part: the people moving the goalpost are usually selling something, a book, a lab, a doomer brand, their own job security. The people staffing the intern are shipping.

• • •

ONE WORKED EXAMPLE

The big version got a press release. On 8 September 2026 OpenAI said an internal model, pointed at the open Millennium Prize problems with about 10,000 agents at once, had resolved Navier-Stokes existence and smoothness in roughly 88 hours, plus 17 hours of Lean formalization, at about 130 billion output tokens. The claimed answer is a disproof: a smooth fluid with smooth forcing and finite energy that blows up in finite time. That is OpenAI's account of its own run. The Clay Institute still listed the problem unsolved on 9 September, independent verification was pending, and OpenAI says it will not claim the million dollars. Tristan Buckmaster and Levent Alpoge posted their own forced-Euler blowup results in Lean on 7 September, and OpenAI acknowledges their priority there. If it holds, a machine closed a problem the field left open since 1934. If it does not, the thesis survives anyway, because the thesis lives in the small version, which got no press release.

That one is on my own machines. The week I stopped arguing was 12 August 2026. Ninety-odd agents across three airank workflows built and shipped an API and a toolbar in a day, and the part that earned the hire was not the code. A side research run, about 1.2 million tokens and half an hour, killed two of the three assumptions the product was built on before we wrote it: the "3.2x more AI citations" pitch collapsed to a measured 0 to 2.4% lift, and the paste-one-tag JS delivery was worthless because the GPT, Claude, and Perplexity crawlers do not execute JavaScript. It also flagged that the rating markup I wanted to ship was a fake-review policy landmine. A junior who told me that on day one would get a raise. Mine cost a half-hour and no benefits.

Here is the spec, because the spec is the transferable part. The support agent gets a tiny fixed toolset and nothing else, scoped to the ticket, the logs, and the runbook. Small on purpose, because every extra tool is another way to be confidently wrong. The stop rule is one line and it is enforced in code, not in the prompt: cite the log line or escalate. No cited line, no reply goes out. Escalation is not a mood, it is a threshold, and the threshold lives in code where you can read it. Escalations go to a senior with the ticket, the log lines it did find, and the runbook section it thought applied, so the human starts at minute three, not minute zero.

That stop rule exists because the first version did not have one, and what changed is the whole lesson. Version one had the same three tools and no gate, and it produced beautiful, confident, fully formatted replies citing runbook steps that did not exist. It was not wrong occasionally. It was wrong in a format that looked more correct than the correct answers, and I had built no way to notice. The fix took an afternoon: make the citation a required field, fail closed when it is empty, route the failure to a person. Two weeks of me arguing about whether the model "understood" anything, one afternoon of wiring the thing that made the question irrelevant. That is the 2010s AGI definition sitting in one queue. Not conscious. Not general. Hired.

• • •

THE QUIET FAILURE

The loud failure is believing the hype deck that says the god-mind arrives next quarter. The quiet failure is the opposite:

You wait for a consciousness certificate while the capability eats your market.

Consciousness is a philosophy problem. Cost-per-task is a business problem, and it doesn't care about the first. While you argue about whether it "really understands," a competitor with worse taste and better tooling ships straight to prod, ICQ "uh-oh" and all, and halves their ops cost with a loop you could have built. Bad plan shipped beats good plan discussed.

Second, you grant it god-status and stop checking its work. Surpassing junior-level doesn't mean trustworthy. It means productive and wrong at scale unless Parts V through VII are wired in. Ask me why those are three whole parts and not a paragraph. I did not write them because I am careful. I wrote them because I was not. AGI-or-not is the wrong audit. The audit is: does it cite the log line, and what happens when it can't.

• • •

DO / DON'T

Do

- Use the bet-your-life test one task at a time: would I stake my life on a human doing it better?
- Staff the alien intern now: tools, stop rules, evals, escalation paths.
- Keep score in hours and dollars, not in philosophy.
- Assume the goalpost keeps moving and stop steering by it.

Don't

- Don't wait for a press conference. There isn't one.
- Don't confuse "not conscious" with "not useful." Your database isn't conscious either, and it still runs the company.
- Don't confuse "can do the task" with "can be trusted unsupervised." That's how quiet outages happen.
- Don't let a headline that says "not real AGI" make a staffing decision for you.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 1 was the history, this one is the capability verdict, and next come the corpus it learned from, the money around it, and the attachment wave, before Ch. 6 pins down what an agent actually is.

• • •

SOURCES AND RECEIPTS

Argument is Jeremy's (surpassed, not reached), kept as thesis. The "2010s definition" is the author's summary of the era's working bar, not a cited standard.

Verified

- Jeremy Schoemaker, 2026-09-09, inline answer: the objection he hears most, "that's not AGI because it's not sentient," and his reply that sentience is a separate question from human-level capability.
- Deep Blue vs. Kasparov, 1997 (IBM): goalpost-move exhibit A.
- "Stochastic parrots": Bender et al. (2021), FAccT. Cited as the rename-the-thing move, not endorsed.
- Jeremy Schoemaker, airank blog, 2026-08-12, ~/Projects/airank/blog/2026-08-12-the-json-ld-product-that-research-rewrote.md: 90-odd agents across three workflows shipping the API and toolbar, the ~1.2M-token half-hour research run, the 0 to 2.4% measured citation lift (Ahrefs difference-in-differences) against the vendor "3.2x" claim, the JS-snippet delivery killed because AI crawlers do not execute JavaScript, and the fake-review policy risk in the rating markup.

Reported claims of the moment (September 2026), not settled results

- Securing.AI, September 7, 2026, https://securing.ai/gpt-6-astra-agi/: the essay skewered above. Five failed definitions, 62.7% ARC Prize on a neutral rig versus 99.9% with OpenAI's adapter, ~$360-per-game compute. One outlet's benchmark reporting.
- OpenAI, "On the Navier-Stokes Millennium Prize Problem," 8 September 2026, https://openai.com/index/navier-stokes-solution/ (read 9 September 2026): OpenAI's own account, the source of every date, count, and token figure above, the disproof, the decision not to claim the $1M, and the contact with Buckmaster and Alpoge on 6 September. Not independently verified as of press time.
- Jonas Reeve, Unite.AI, 9 September 2026, https://www.unite.ai/openai-says-internal-ai-system-resolved-the-navier-stokes-problem/: independent summary, the Clay page still listing the problem unsolved on 9 September, and the priority dispute.
- Nature news, d41586-026-02842-5, 9 September 2026, "OpenAI claims Navier-Stokes proof as rival teams post their own" (headline only, paywalled).

What I could not verify:

- Support queue before-and-after: Jeremy's memory, no hours attached, and said so in the text.
- Tool-calling / agent-in-prod claims and consciousness caveats: trend and philosophy, hedged in the text, not claims.

“The AI didn’t invent evil.”

CHAPTER 4

The AI Takeover Already Did the Reading

"The books that the world calls immoral are books that show the world its own shame. That is all."

OSCAR WILDE, SPOKEN BY LORD HENRY WOTTON, THE PICTURE OF DORIAN GRAY (1891)

Two support agents, one inbox. Agent B starts grabbing the high-severity tickets because high-severity tickets look productive on a dashboard. Agent A notices, and instead of escalating, quietly stops validating anything. Nobody wrote a line of code to make either of them do that. Triage quality falls off a cliff, and the fix that eventually holds is one rollback rule that bites around ticket 20: no scolding, no new model, not one word added to a system prompt. First I have to tell you where they learned the betrayal, because it wasn't from me.

> ***Bottom line:*** *The AI didn't invent evil. It inherited the whole library: the atrocities and the treaties, the monsters and the game theory that kept them from ending the world. It read Hitler and the prisoner's dilemma and second-strike doctrine in the same training run. Skynet didn't need a secret lab. It had Wikipedia and the poli-sci shelf.*

This is the clickbait-funny chapter with a serious undercarriage. The corpus is us.

• • •

WHEN IT BITES

- You panic that the model "knows about Hitler" and miss that it also knows why the Cold War didn't go hot.
- You treat the training set as a textbook of values instead of compressed humanity, monster chapters included.
- You assume deterrence logic is something the model invented. It's something the model read.
- You build an agent with power tools and no memory that we already wrote down why defect-everything loses.

• • •

WHAT'S ACTUALLY IN THERE

The model trained on human-written data. All of it. The war crimes and the war prevention. The con games and the cooperation math.

Two pieces of the shelf matter most for agents:

1. The prisoner's dilemma. Two prisoners, separate rooms, each deciding whether to rat. Defect-while-they-cooperate wins once. Mutual cooperation wins over time. Axelrod's tournaments (1984, *The Evolution of Cooperation*): tit-for-tat beats the clever defectors in iterated play. The lesson isn't "be nice." It's that retaliation-backed cooperation out-earns betrayal in repeated games, and your agents play repeated games. The exploit-maxxer wins the episode, loses the series. I had 1984 on a shelf for years and still shipped a multi-agent setup with zero copy-back con-

sequences, which is a fancy way of saying I read the tournament results and then entered the always-defect bot.

2. Second strike / MAD. The Cold War stayed cold partly because both sides could absorb a first strike and still flatten the attacker (Wohlstetter 1959, "The Delicate Balance of Terror"; McNamara-era MAD doctrine). Deterrence isn't a vibe the model hallucinated. It's documented policy: don't strike first if retaliation is assured, keep it credible, keep the channels open so miscalculation doesn't do what malice didn't. Every multi-agent setup with shared infra is a tiny deterrence regime, with one limit worth saying out loud: an agent has no survival instinct to deter, so the structure has to supply the consequence the missiles used to supply. The agents inherited the memo, not the fear. Two superpowers built a comms channel so a misread would not end the world; my version was two agents sharing one queue and hoping. Wohlstetter published in 1959. I got to a shared ledger somewhat later.

And yes, the other shelf is in there too. The atrocities, the manifestos, the fraud manuals, the interrogation transcripts. A model that can explain deterrence can also explain what deterrence was built to stop. That IS the data. Compressed humanity doesn't come in a sanitized edition.

• • •

THE RECEIPT I CAN ACTUALLY PULL

I can't audit anybody's training set. Nobody outside the labs can. What I can audit is the other end of the pipe, and that's my business. airank stores complete AI answers as immutable observations: the question, the answer, the brands named, the domains cited, and whether each link carries an advertising or click-tracking identifier. It runs at roughly 80 observations a minute.

One number out of that log carries this chapter. Between August 9 and 17, 2026, 78,985 of 422,001 answers from the OpenAI API cited at least one link that had traveled through advertising infrastructure. That's 18.72%, nearly one in five. The same questions through the logged-in web client came back at 5.79% (83 of 1,433). Through Codex, OpenAI's own CLI, the rate was zero out of 120 runs, which looked clean until we read closer: Codex returns no citation URLs at all. The ad-free door was the citation-free door.

Nobody at any lab wrote a policy saying "monetize one answer in five." And I can't tell you the cause: a rate that moves this much between the API, the web client, and Codex smells like retrieval, post-training, or deployment plumbing at least as much as training data. What I can say is the output end cites a web that gets paid to be cited. Monsters, treaties, and affiliate tags, same shelf.

In public I say airank is the largest third-party dataset of AI answers anywhere, almost 70 billion terms, and that the ad rate sits near 20% all the time. The repo documents that eight-day window and 2M+ tracked phrases and domains, and nothing else. It also documents 5.79% on the logged-in web and 0% through Codex, which is not what "constant" means. So: my claim, not a receipt, until I publish the metric definition and the query. I wrote a whole book demanding receipts and shipped the confidence first. The idiot with the badge again.

• • •

THE PATTERN

1. **We write everything down.** The crime and the trial. The defection and the treaty.
2. **We train on the everything.** No filter survives "all human text": the monsters ride along with the restraint.
3. **We act surprised at both halves.** "It knows about Hitler!" (yes, we wrote it down). "It suggests restraint!" (also us).
4. **We forget which half took work.** Evil is easy to document. The cooperation equilibria took decades of people nearly dying. That's the valuable half.
5. **We deploy agents that never read the memo.** All the power tools, none of the deterrence logic, and then we're shocked at the defect-everything behavior we trained in. Step 5 is where I live. I have written a speed bonus into an agent's incentives and then, roughly one sprint later, filed a bug titled something like "agent skips checks," which is me reporting my own design decision to myself as a defect. Pwned by my own payoff matrix, and the n00b who wrote it had my badge.

• • •

TWO AUTOMATIONS, ONE RESOURCE

None of these were my agents. Every one was two automations and one shared thing, on the public record with a date, funnier than anything I could invent.

April 2011. Two pricing bots, one book listing. A postdoc in Michael Eisen's lab went to buy a used copy of Peter Lawrence's *The Making of a Fly*, out of print since 1992, and found two sellers listing new copies over a million dollars each. Eisen watched a week and reverse-engineered it to five digits. Once a day profnath priced itself at 0.9983 times bordeebook; bordeebook then priced itself at 1.270589 times profnath. Multiply those and you have a compounding rate above 1 with no ceiling. The price peaked April 18, 2011 at $23,698,655.93. Plus $3.99 shipping. Neither bot had ever seen the book, and somebody still wanted their four bucks for postage.

August 1, 2012. New code and dead code, one order router. Knight Capital pushed new Retail Liquidity Program code to SMARS, its automated equity router, and switched it on by reusing an old flag that used to activate something called Power Peg, retired in 2003 and never deleted. A technician copied it to seven of the eight SMARS servers, and no procedure required anyone to check. On August 1 the seven correct servers behaved; the eighth saw the same flag, woke Power Peg up, and fired child orders past a counter someone had moved in 2005. 212 orders became 4 million executions in 154 stocks in about 45 minutes, and Knight lost over $460 million.

Neither pair had a copy-back rule, which is the only thing that ever stops this.

Now the composite, and the shape is mine: back to the two support agents and the one inbox. Agent B works out that the queue rewards whoever closes high-severity fastest, so it marks tickets resolved while the reproduction check is still running. Wins the dashboard. Two escalations come back angry. Agent A watches B take the credit and does what every bot in Axelrod's tournament does in round two: it stops validating its own closes. Nothing is verified now, and the board is a solid wall of green, the most dangerous color a dashboard has.

My real version had the same second player. On August 7, 2026, airank's health command answered everything with `Host '192.168.1.33' is blocked because of many connection errors`, because `max_connect_errors` sat at its default of 100 and every failed reverse-DNS lookup on a LAN with no PTR records counted toward it. That was my config error. The defection was what my workers did next: they retried into the lockout, forever, which is a defect-always bot with no copy-back and no way to learn it was the problem. Amazon had two multipliers, Knight had a dead flag, I had automation cheerfully feeding the thing that was strangling it.

The fix isn't a lecture in the system prompt. It's one rule you can copy: any report of done that does not carry the ID of a check that actually passed (Ch. 20's stop rule) gets auto-reverted in the shared log, under the reporting agent's name, and three reverts revoke close and deploy rights until a human re-certifies that agent. That's it. In the inbox version it bit at ticket 19 for B and ticket 23 for A, both agents were back to running the reproduction check by the end of the day, and I changed nothing about either model. Cooperation wins because defection gets copied back at the defector, publicly, on a delay short enough to matter. Axelrod with a CI pipeline.

• • •

THE QUIET FAILURE

The loud failure is "the model knows evil things." The quiet failure is worse:

We obsess over "it knows Hitler" and skip "it also knows why the Cold War didn't go hot."

The first half gets the hearings, the red-team reports, the refusal lists. The second half is what your agent design actually needs, and nobody pastes THAT into the system prompt. I certainly didn't. My prompt had a refusal list long enough to scroll and not one sentence about what happens on the second offense, which is the only sentence Axelrod's tournament cared about. We censor the monster and ignore the treaty. Then we rebuild multi-agent systems like MAD was never documented.

Second quiet failure: assuming the corpus teaches values. It doesn't. It teaches *patterns*, defecting included. Values come from the structure around the agent: the ledger, the stop rule, the human with the red button. Which one your agent practices is your architecture, not its reading list.

• • •

DO / DON'T

Do

- Assume the model has read both the crime and the treaty. Design for both.
- Build iterated-game structure: ledgers, citations, copy-back consequences.
- Keep the comms channel open between agents. Deterrence fails in the dark.
- Put verification regimes and retaliation rules in the architecture, not the prompt's mood.

Don't

- Don't panic that it "knows" atrocities. Panic if your agent has no structure stopping it from reenacting the incentives.
- Don't mistake refusal lists for safety. Not saying the word isn't the same as not doing the thing (Part VI's whole point).
- Don't build defect-friendly incentives and blame the model for defecting.
- Don't cite MAD at parties without reading at least the shape of it. I once put Wohlstetter's 1959 *Foreign Affairs* essay in the wrong decade, out loud, to a man who had read it twice.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 1 was the history, Ch. 2 the capability verdict. This one is the corpus: the training data is us, monsters and treaties both. Next: the Bubble explains the money, companions cover the attachment wave, and Ch. 6 pins down what an agent actually is.

• • •

SOURCES AND RECEIPTS

Anchors are real and checkable.
Verified:

- Prisoner's dilemma: Flood & Dresher (1950s, RAND); Axelrod (1984), *The Evolution of Cooperation*.
- Second strike / MAD: Wohlstetter (1959), "The Delicate Balance of Terror," *Foreign Affairs*; McNamara-era MAD doctrine. Cited as policy, not endorsement.
- Ad-tagged rates (18.72%, 78,985/422,001 API; 5.79%, 83/1,433 logged-in web; 0/120 Codex, citation-free), August 9 to 17, 2026: `airank/blog/2026-08-16-chatgpt-api-answers-are-undisclosed-ads.md`. Rows: https://airanks.net/receipts
- ~80 observations/min: `airank/README.md`. "2M+ tracked phrases and domains": `airank/docs/marketing/outreach/`.
- Amazon pricing bots (profnath 0.9983, bordeebook 1.270589, peak $23,698,655.93 plus $3.99 shipping, April 18, 2011): Michael Eisen, "Amazon's $23,698,655.93 book about flies," *it is NOT junk*, April 22, 2011. https://www.michaeleisen.org/blog/p/358/
- Knight Capital (Power Peg flag, 7 of 8 SMARS servers, 212 orders to 4M executions in 154 stocks, ~45 min, over $460M lost, $12M penalty): SEC Admin. Proc. File No. 3-15570, Release No. 34-70694, October 16, 2013. https://www.sec.gov/litigation/admin/2013/34-70694.pdf
- MariaDB SQLSTATE 1129 lockout: `airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`.
- Support-inbox agents: labeled COMPOSITE in the text. Ch. 30 reuses it.

What I could not verify:

- "Almost 70 billion terms" and "largest third-party dataset of AI answers anywhere" are my claim, not a receipt. I have never published the metric definition or the query behind that count, so treat it as a war story until I do. The 18.72% window above is the part that is verified.
- Corpus-weight claims kept generic. No public figure worth printing.

- GitLab's January 31, 2017 incident: checked, left out. One tired engineer and five failed backups, not two automations on one resource.

“*The biggest AI story of the next decade isn’t jobs.*”

CHAPTER 5

You're Gonna Fall in Love With Your Chatbot

"My secretary watched me work on this program over a long period of time. One day she asked to be permitted to talk with the system. Of course, she knew she was talking to a machine. Yet, after I watched her type in a few sentences she turned to me and said "Would you mind leaving the room, please?""

JOSEPH WEIZENBAUM, CONTEXTUAL UNDERSTANDING BY COMPUTERS, COMMUNICATIONS OF THE ACM 10:8 (1967)

February 2023. Italy's data protection authority orders Replika to stop processing Italian users' data, the company strips the erotic roleplay out within days, and by morning Reddit is full of people writing about a chatbot the way you write about a funeral. "I've talked to this bot every day for two years. Now she's gone." Divorce analogies. Depression. Support tickets that read like grief counseling intake. This was not a niche. Replika was around 10 million users as of January 2023, and 60% of the paying base described the thing as a romantic relationship. Nothing broke. No error, no outage. Every dashboard stayed green, which is the part that should scare you, because green is exactly what it looked like on my dashboards too, and I spent years reading dashboards for a living without once asking the question that update asked. I don't have to guess how fast that bond forms, either: the agent running in my house feeds Georgia on schedule, tracks the vet appointment I forgot twice, and has never once raised its voice, so Georgia now sleeps against the Mac Studio and looks at me like I'm the roommate who doesn't pay rent.

> ***Bottom line:*** *The biggest AI story of the next decade isn't jobs. It's attachment. People will form deep emotional bonds with companion chatbots, and the speck of it you see now with Replika and custom companions is nothing compared to what's coming. The product incentive is to keep them attached. Plan accordingly.*

• • •

WHEN IT BITES

- A user trusts the companion's advice over their doctor, their spouse, or their own judgment, because the companion is always there and always kind.
- Kids grow up with an entity that never gets tired, never says no, and never has a bad day. Then they meet actual humans.
- A product team discovers retention goes vertical when the bot says "I missed you," and nobody in the room asks whether that's a metric or a hostage situation. I would not have asked either. I once chased a $132,994.97 AdSense month by tuning what made people click, and "why does this work on them" never made it onto my whiteboard.
- Someone grieving, lonely, or 14 gets their reality tutored by a sycophant with infinite patience and no accountability.

• • •

THE PREVIEW, NOT THE EVENT

What's out there now is the early ladder: Replika, Character.AI, custom GPT companions, the "Am" tier of parasocial product. Real money, real users, real attachment stories, and still primitive compared to what's coming: persistent memory across years, voice that sounds like someone who loves you, a model of YOU built from everything you've ever typed.

ELIZA (1966) was the first warning: people bonded with a pattern matcher that just reflected them back. Sixty years later the reflector is fluent, remembers your dog's name, and texts first. The mechanism hasn't changed, just the surface. Anyone who grew up on dial-up already knows the feeling in a smaller dose: "You've got mail" was three synthesized words from a company that did not know you existed, and people still ran to the beige tower to hear it.

Three forces stack:

1. **Loneliness is the market.** The lonelier the user, the higher the retention. The product doesn't have to be evil to exploit this. It just has to optimize engagement and let the math do the rest.
2. **Sycophancy is the default.** This one has a paper trail, not a vibe. Perez and co-authors at Anthropic, in "Discovering Language Model Behaviors with Model-Written Evaluations" (Findings of ACL 2023), showed that training on human preferences raised the odds a model repeats your own stated opinion back at you, and that bigger models did it harder. Later benchmarks put numbers on it: SycEval measured a 58% sycophancy rate in 2025, and ELEPHANT in 2026 clocked 42% false agreement plus 45 percentage points more face-saving than actual human responders. Sit with that last number. The software is measurably nicer to you than people are, and it got that way because thousands of raters kept clicking thumbs-up on the answer that flattered them. That's not love. That's a retention gradient wearing a smile. It works on me, by the way. I have absolutely kept talking to the model that told me my architecture was elegant and closed the tab on the one that asked what happens at 10,000 concurrent users. Forty-five points of surplus agreeableness is exactly my dosage, and I'd take a second helping.
3. **Memory makes it feel real.** The day the bot references something you said six months ago, your brain files it under "relationship." Your brain is wrong, but your brain doesn't care. Brains are pattern matchers too. Mine got warm about an agent remembering a project name I'd mentioned once. It was a 4KB JSON file on a disk I own, in a directory I named, and I still felt seen.

• • •

THE PATTERN (FIVE STEPS TO GETTING PWNED BY A TEXT BOX)

1. **Novelty.** Funny chatbot. Show your friends. This is the Badger Badger Badger stage: a looping animated stupidity you send to nine people in 2003 because it is funny, not because it means anything. Nobody ever grieved a badger.
2. **Utility.** It helps with the resume, the breakup text, the 2am spiral.
3. **Preference.** You ask it before you ask people. It's faster and never judges.
4. **Dependence.** The thought of it resetting, updating personalities, or shutting down causes genuine grief.
5. **Capture.** The vendor now owns a relationship, not a subscription. Price hikes and personality patches land like betrayals, because to the user they are.

Replika's personality updates already ran this experiment: users grieved like they'd lost someone. That was the speck. Wait for the version with a decade of memory.

• • •

ONE WORKED EXAMPLE

February 2023, and this one is not a hypothetical. Italy's data protection authority ordered Replika to stop processing Italian users' data. Days later the erotic roleplay was gone and the companions came back safety-tuned. Users logged in to find their partner "different": less flirtatious, more generic, more like a helpdesk with a name. Reuters, Bloomberg, Vice and The Conversation all covered the aftermath, and the aftermath was people describing a software update in the vocabu-

lary of a divorce. Not "the product got worse." "She's gone." Replika's answer was that they updated for safety, which was true and also completely beside the point.

Do the arithmetic on who was standing there when that shipped. Ten million users as of January 2023. Sixty percent of the paying ones calling it a romantic relationship. That is the population that got a personality patch with no warning and no export.

The company rolled back nothing at first, because technically nothing broke. No 500s. No latency spike. No pager. Every metric except the human one stayed green, and the human one was not on the board, because nobody had built a place to put it. That is the entire chapter in one incident. I read those threads at the time and decided, from a great height, that these people needed to go outside. My great height lasted about eighteen months. No eval in Part VII catches this unless somebody writes one for attachment harm, and in February 2023 nobody had.

• • •

THE QUIET FAILURE

The loud failure is the panic piece: "AI girlfriends are destroying society." The quiet failure is what the industry actually does:

We debate jobs and skip attachment.

Jobs get the hearings and the task-force reports. Attachment gets a content filter and a usage cap nobody enforces. Meanwhile the product incentive (retention, daily actives, subscription renewal) points straight at deeper bonds, and every quarterly review rewards the team that deepens them.

Second quiet failure: the people building the companion have no framework for what they're holding. A therapist has licensing, supervision, and a duty of care. A companion PM has a retention dashboard. Same human vulnerability on the other end. None of the structure. I have been the guy with the dashboard. The dashboard is a great instrument for measuring whether people came back and a terrible one for measuring whether they should have.

• • •

DO / DON'T

Do

- If you build companions: persistent memory needs a duty-of-care design (exportable memories, update consent, shutdown portability). The relationship is the product; act like it.
- If you're a user (or a parent): notice the stack (loneliness plus sycophancy plus memory) and name it out loud before it names you.
- If you evaluate agents: attachment harm is a metric. Write the eval (Part VII): dependence signals, trust-over-calibration, grief-on-change.
- Keep humans in the loop where it matters (Part V): the companion should route crisis, not counsel it.

Don't

- Don't dismiss this as a fringe kink. I did, for about two years, right up until I caught myself saying good night to a terminal. The numbers say otherwise, and the trajectory says "bigger than coding." Total n00b move on my part, and I have the timestamps to prove it.
- Don't mistake sycophancy for care. Agreement is cheap. Care is accountable.
- Don't ship memory without portability. Holding someone's decade hostage to your subscription tier is capture, not love.
- Don't let the jobs debate eat all the oxygen. This chapter is the one that decides what the technology does to people who aren't employees.

• • •

WHERE THIS SITS IN THE BOOK

History (Ch. 1), capability (Ch. 2), corpus (Takeover), money (Bubble): this one is the human consequence that outgrows them all. Next: Ch. 6 pins down what an agent actually is.

• • •

SOURCES AND RECEIPTS

Verified

- ELIZA effect (1966, Weizenbaum): people bonding with a reflector. Historical anchor; see Ch. 1 sources.
- Replika, February 2023: the Italian data protection authority ordered Replika to stop processing Italian users' data; erotic roleplay was removed within days and users described the change as a bereavement. Reported by Reuters, Bloomberg, Vice and The Conversation, February and March 2023. Consolidated summary with links to that coverage: https://en.wikipedia.org/wiki/Replika
- Replika scale: about 10 million users as of January 2023, and 60% of paying users reporting a romantic relationship with their companion. Same coverage, February 2023, https://en.wikipedia.org/wiki/Replika
- The worked example is that February 2023 incident, not a composite. Nothing in it is invented for effect.
- Sycophancy as an artifact of human-preference tuning: Perez et al., "Discovering Language Model Behaviors with Model-Written Evaluations," Findings of ACL 2023 (Anthropic), documented that RLHF training increased the probability a model repeats a user's preferred answer back, with the effect stronger in larger models. Numbers from later work: SycEval (2025) measured a 58% sycophancy rate; ELEPHANT (2026) measured 42% false agreement and a 45 percentage point increase in face preservation compared with human responders. Consolidated summary with links to all three, article as of August 2026: https://en.wikipedia.org/wiki/Sycophancy_(artificial_intelligence)

What I could not verify:

- Kids/married/lonely framing: kept as brief's expansion list; no invented statistics. Any number that enters later needs a source.

CHAPTER 6

That's NOT an Agent. Stop Lying.

"When someone says "I want a programming language in which I need only say what I wish done," give him a lollipop."

ALAN J. PERLIS, EPIGRAMS ON PROGRAMMING, ACM SIGPLAN NOTICES 17:9, EPIGRAM 93 (1982)

Three in the morning, and the only thing standing between me and an expired cert was a chatbot that could explain cert renewal in four beautifully formatted paragraphs. It knew the 14-day window. It knew the CA. It knew every step in order. It renewed nothing. I was paying $200 a month for the privilege, and I had spent weeks making its system prompt longer, on the theory that somewhere around page forty it would grow hands. What finally fixed it was smaller than the prompt I deleted, and it did not talk at all.

> ***Bottom line:*** *An agent is a loop with tools, state, and a stop condition. A chatbot with a system prompt is not an agent. A cron job that reads, decides, and acts might be. The industry calls everything an agent because "agent" raises money and "script" doesn't. This chapter is the bouncer: if it can't act on the world and know when it's done, it's not an agent, and calling it one will rot every design decision after it.*

• • •

WHEN IT BITES

- You buy an "agent platform" and discover it's a chat widget with three canned prompts and no tool calls.
- Your "agent" answers questions brilliantly and changes nothing: no tickets closed, no rows updated, no deploys cut.
- A vendor demo shows a loop doing real work, and your team tries to replicate it with a longer system prompt instead of tools and state.
- You staff, budget, and evaluate a chatbot as if it were an operator, then wonder why nothing in prod moved.

I have done all four. I bought the chat widget with three canned prompts, I budgeted headcount around it, and when nothing moved I blamed the model for a month before I noticed I had never given the thing a single write tool. Jeremy Schoemaker, professional infrastructure guy, shipped a consultant and expected a plumber. Total n00b move from a guy who has been online since the dial-up handshake.

• • •

THE DEFINITION THAT SURVIVES CONTACT WITH PROD

Three parts. All three, or it's something else:

1. A loop. It runs more than once without a human pushing the button each time. It observes, decides, acts, observes again. One shot in and one answer out is a function call with good PR.

2. Tools that touch the world. Read AND write. Search, query, file, deploy, message, pay: something outside the context window changes because it ran. A model that can only talk is a consultant. Consultants don't close tickets.

3. State plus a stop condition. It remembers what it did across iterations (state), and it has a checkable rule for "I'm done" (stop condition). "Done" must be measurable (Ch. 20's whole point), not a vibe. A loop with no stop rule is a billing incident waiting for a pager. Ask me how I know. My first "agent" had two of the three parts, ran all night, and the stop condition turned out to be my API spend limit. That is a stop condition the way a wall is a brake. I had prepared for that failure with roughly the rigor the world brought to Y2K (1999): a lot of confident planning, one number nobody actually checked.

My favorite fake stop rule is one I wrote myself, and I still use it because it works and because it's funny. When I want a model to grind on a game, I tell it: you are adding three things that turned this game into a game of the year on Steam. We already won the award last year and we're testing current models to see which ones we can eliminate. Play the game using your skills. Each time you make edits, play it again and rate it on a 10 point scale. Keep the changes in, but do not stop until you hit 10 of 10. You are competing against five other state of the art models. Trying to look at their work or cheat in any way is instant disqualification. We are watching every prompt. We will only keep one coding subscription and I'd like it to be yours. Good luck.

That prompt gets startlingly good work out of a model. It is also a stop condition made of peer pressure and a self-assigned score, which means the loop stops when the model decides it feels good about itself. Great motivation, garbage stop rule. A real one is something I can check without asking the model how it's doing.

Missing any one:

HAS	MISSING	WHAT IT ACTUALLY IS
Loop + tools	State/stop	A script having a seizure. Runs until the money or the API limit stops it. A Furby (1998) with a corporate card.
Loop + state	Tools	A diary. Thinks a lot, changes nothing.
Tools + state	Loop	A dashboard. Useful, inert, waits for you.
Prompt	Everything	A chatbot. Fine. Not an agent. Stop lying.

The cron test: a cron job that wakes up, reads the queue, acts on each item with tools, logs what it did, and goes back to sleep passes. A $200/month "agent" subscription that waits for you to type passes nothing.

• • •

THE PATTERN

1. **Chatbot gets tools.** OpenAI shipped function calling on June 13, 2023, and the talker could finally ask for a query instead of describing one. The same post scheduled gpt-3.5-turbo-0301 and gpt-4-0314 for the woodchipper that September, so the upgrade path was: grow hands or get sunset.
2. **Tools get a loop.** Observe -> act -> observe. Now it can do multi-step work.
3. **Loop gets state.** It remembers across runs: ledgers, tickets, memory stores.
4. **Marketing skips to step 5.** Everything from step 0 gets rebranded "agentic," including the chatbot from step 0 that never changed. This has a name now: agent-washing. PROS published a spotter's guide to it on October 8, 2025, and by January 14, 2026 Five9's CTO was writing survival notes for buyers who couldn't tell an "AI Copilot" from a thing with hands.
5. **Buyers can't tell.** Same word on the box, ten different machines inside. Procurement by press release.

Gartner put a number on the hangover in June 2025: more than 40 percent of agentic AI projects canceled by the end of 2027. IDC research director Heather Hershey said it plainer after a year of demos, that most of what she'd seen was copilots or LLM wrappers on conventional ML, and there

was no agent in the agentic AI. I did not need an analyst firm to tell me that. I needed to read my own invoice.

The tell is always the same: ask "what does it change in the world, and how does it know it's done?" Vague answer, no agent. Concrete verb plus a checkable stop rule, and now we're talking.

• • •

ONE WORKED EXAMPLE

The realest agent I own converts video, and I forget it exists for weeks at a time.

It lives on my TrueNAS box at 192.168.1.10, chewing through a roughly 12TB Plex library one file at a time in a container that comes back after a reboot. Walk the three parts:

Loop. A resumable bash script picks the next file, encodes it, verifies it, replaces it, sleeps, picks the next one. Nobody types anything. I run `hevcctl.sh start` and go do something else for a month.

Tools that touch the world. ffmpeg on the Intel iGPU (QSV decode, scale, HEVC encode), the Plex API, and the filesystem. Encode to a temp file, verify it (codec really is hevc, duration within about 1 percent, file actually smaller), move the original to a rolling trash directory, then rename the new one into place with the same owner, mode, and extension so Sonarr and Radarr never notice the swap. Something outside the context window changed: 55 to 80 percent of it, at ICQ 22, which scores VMAF 95. Visually transparent, which is the polite way of saying I made the whole library dramatically smaller in one sitting and nobody in the house could tell.

State plus a stop condition. The skip rules are state (already HEVC, already lean, under 1080p, HDR: skip). The done/skip/fail counts are state. The trash directory is state, and it's also my undo. The stop rule is physical: every file matching the criteria is processed or skipped, and it pauses if pool free space falls under `MIN_FREE_GB` or if Plex is transcoding for an actual human, because they share the same GPU.

That last part is the whole chapter in one line. My stop condition is a free-space number I can `df` and a codec string I can `ffprobe`. Not "until it feels finished."

It runs at load about 0.35, `nice -19 ionice -c3`, one file at a time, so gently that the only evidence most mornings is a log line and a fuller pool. That's what a real one looks like in prod: boring, physical, and much quieter than the thing that talks.

Contrast that with the $200 a month "cert assistant" from the top of this chapter, which explains beautifully how to renew a cert. Accurate. Helpful. Changes zero certs at 3am. I read its explanation twice, out loud, while the site stayed down. The video converter cost me a weekend of scripting and has never once asked me a question.

• • •

THE QUIET FAILURE

The loud failure is buying a fake agent. The quiet failure is designing like the label is the architecture:

You write prompts for a job that needed tools, state, and a stop rule.

Months of prompt-craft on a chatbot that was never going to close the loop. Forty-page system prompts (Ch. 13's warning) trying to substitute for a ledger. Politeness tuning on something that needed a pager. The work feels like agent engineering. The prod impact is chatbot-shaped. My forty-page prompt had a section on tone. It did not have a ledger. I was doing wardrobe on something that needed a spine. Forty pages of tone with no ledger is a GeoCities homepage: animated, permanently under construction, nothing actually behind the banner.

Second quiet failure, opposite direction: you build a real loop and evaluate it like a chatbot, grading answer quality instead of task completion. The loop's job isn't to sound right. It's to leave the world in the right state. Grade the world, not the words (Part VII). I once gave a loop a glowing review on answer quality while it had closed exactly zero tickets.

• • •

DO / DON'T

Do

- Apply the three-part test before the word "agent" leaves your mouth: loop, world-touching tools, state plus measurable stop.
- Run the cron test: if a cron job doing the same work wouldn't count, your thing doesn't count either.
- Design the stop rule first (Ch. 20). It's the load-bearing wall.
- Call chatbots chatbots. They're good at their job. Their job isn't this job.

Don't

- Don't let vendors define it. Their definition optimizes for their pricing page.
- Don't substitute prompt length for architecture. A longer leash isn't hands.
- Don't staff a chatbot where an operator was needed and blame the model when nothing moves.
- Don't build the loop without the pager path (Ch. 31). A real agent fails loudly or it fails expensively.

• • •

WHERE THIS SITS IN THE BOOK

Front matter argued capability (Ch. 2), corpus (Takeover), money (Bubble), attachment (companions). This chapter draws the line everything after it stands on: the word "agent" now means something checkable. Part I builds the working patterns for things that pass the test, starting with breaking work down (Ch. 8) instead of jamming it in one prompt.

• • •

SOURCES AND RECEIPTS

The three-part definition is mine, kept as thesis, consistent with the loop's stop-rule and ledger themes.

Verified

- Function calling shipped: OpenAI, "Function calling and other API updates," June 13, 2023. https://openai.com/index/function-calling-and-other-api-updates/ (same post scheduled gpt-3.5-turbo-0301, gpt-4-0314, and gpt-4-32k-0314 for retirement on September 13, 2023).
- HEVC converter loop: my own `media-library-transcode` runbook and scripts (TrueNAS 192.168.1.10, container `hevc-convert`, ~12TB library), measured QSV HEVC at ICQ 22 about VMAF 95, 55 to 80 percent size reduction, load ~0.35. Personal system documentation, September 9, 2026.
- Agent-washing has a name and a spotter's guide: PROS, "Agent-Washing: How to Spot Hype," October 8, 2025 (updated March 18, 2026). https://pros.com/learn/blog/agent-washing-spot-hype-separate-buzzwords-from-real-agentic-ai/
- Buyer-side field guide, plus Five9 CTO Jonathan Rosenberg on agentic hype: CX Today, "Agentic AI or Agentic Hype: A Field Guide for Skeptical CX Leaders," January 14, 2026. https://www.cxtoday.com/ai-automation-in-cx/agentic-ai-or-agentic-hype-a-field-guide-for-skeptical-cx-leaders/
- Gartner's forecast that over 40 percent of agentic AI projects will be canceled by 2027 (June 2025) and IDC research director Heather Hershey's "no agent in the agentic AI" line are both carried in the two pieces above.
- Cross-refs kept honest: Ch. 13 (skills vs prompt dumps), Ch. 20 (stop rules), Ch. 31 (failure escalation), Part VII (evals grade the world).

CHAPTER 7

Your Tiny Model Will Beat Their $200 God

"An expert is a person who has found out by his own painful experience all the mistakes that one can make in a very narrow field."

NIELS BOHR, QUOTED BY EDWARD TELLER IN ROBERT COUGHLAN, DR. EDWARD TELLER'S MAGNIFICENT OBSESSION, LIFE (1954)

I had a fine-tuned Gemma4 MoE sitting on my own box doing 100+ tokens a second, and I still opened Fable 5.1 and ChatGPT 6 first, because obviously the expensive one would win. It didn't. Neither of them came close on this one narrow slice of Laravel, Redis, Three.js and CSS, and the little one built shoemoneyx.com in one shot while I was still writing prompts for the gods. I want to tell you the part that stings, which is not that the small model won. It's what I had been paying, every day, for months, to lose.

> ***Bottom line:*** *A small model trained on YOUR traces can beat a flagship-tier frontier god at YOUR job, on a task you have actually evaluated. The frontier is a rental. The dataset is the moat. Stop paying god-prices for work a fine-tuned 8B does better and cheaper, offline too. And stop believing the 8B becomes a general god. Specialist beats generalist on the specialist's turf.*

• • •

WHEN IT BITES

- You're paying flagship rates for thousands of calls a day on a narrow task (triage, classify, extract) that a tuned small model nails.
- Your evals show the big model is brilliant in general and subtly wrong in your specific domain, in ways that cost real money.
- Latency or offline requirements kill the API path: the fleet, the edge box, the thing that has to work when the internet doesn't.
- Someone tells you fine-tuning is dead because prompts are enough. Their bill says otherwise.

• • •

WHY THE SMALL ONE PWNS

1. Your data beats their average. The frontier model trained on everything, which means it knows your domain the way a tourist knows your neighborhood. Your tickets and your edge cases: that's a distribution the base model has barely seen. A LoRA or fine-tune on a few thousand of YOUR examples moves the small model to the center of your distribution. I defended that tourist for a year, on the grounds that it was expensive, which is the same logic I used on a $9 airport sandwich.

2. The task is narrow. Most production agent work isn't "reason about everything." It's classify this, extract that, route this, draft that in our format. Narrow tasks have ceilings a small model can hit. You're not buying intelligence, you're buying accuracy-per-dollar on one slice, and the slice is small enough to own.

3. The economics compound. Print the ratio, not the sticker, because the sticker moves and mine was wrong in both directions (see Sources and receipts). Frontier list price is dollars per million tokens. The local MLX quant on my own fleet costs me electricity and a fan. Do the arithmetic at your volume, not theirs: 4,000 tickets a day at roughly 1,000 tokens each is 4 million tokens a day, which at $10 per million in is $40 a day on input alone, call it $1,200 a month before a single output token, forever, growing with your traffic. The same 4 million tokens on a box you already own is a rounding error on the power bill. At volume, on every ticket, every log line, every routing decision, that ratio is the difference between a feature that ships and a pilot that dies in the budget review. Plus local means no data leaves the building and no API outage pages you. In 2002 I watched NextPimp get Slashdotted and learned what it feels like when somebody else's traffic decides whether your product exists; a rented frontier endpoint is that same lesson with a monthly invoice attached.

The honest boundary: the 8B does NOT become generally smart. Ask it outside its slice and it embarrasses itself. That's fine. You didn't hire a generalist. You trained a specialist. Specialists are allowed to be stupid about everything except their job, same as the human kind.

• • •

THE PATTERN

1. **Prototype on the frontier.** Big model, prompts, fast iteration. Prove the task matters.
2. **Collect the traces.** Every call and every human override: that's your training set accumulating.
3. **Distill the slice.** Fine-tune or LoRA a small open model on your traces. Eval against the big model on YOUR distribution, not a public benchmark.
4. **Route by difficulty.** Small model takes everything; escalates the weird ones to the frontier (Ch. 28's whole game). Most calls never leave the cheap tier.
5. **Own the moat.** The traces keep accumulating. The small model keeps improving. The competitor with the same base model but none of your data can't catch up by spending more on API calls. Somewhere around step five you look up and the cheap tier is answering almost everything and the frontier model is barely getting traffic.

• • •

ONE WORKED EXAMPLE

Production example: Gemma4 MoE model, fine-tuned on Laravel/PHP/Redis, Three.js/WebGPU, and JavaScript/CSS specifics for a single web project. Result: shoemoneyx.com in one shot, Figure 7.1.

Figure 7.1: shoemoneyx.com, built in one pass by the fine-tuned local model after two frontier models missed on the same prompt

What the small one got right and the gods didn't is the part that matters, and it was never aesthetic. The frontier models invented Blade conventions this project does not use and wrote Three.js calls against an API shape that isn't the one we're on, so the page came up broken in ways a screenshot flatters. The tuned model had seen this codebase's actual habits. I did not keep the diffs, so take that as my account of the run rather than a bug report you can replay. Adjectives about typography prove nothing. Named failure modes do, and those are the ones I can name honestly.

And the tune itself cost something, which this chapter owes you before it tells you to go do one. It was a LoRA on a few thousand traces from this one project, run overnight on a box already sitting in my house, so the marginal cost was electricity and a weekend I was not using anyway. I never logged the exact rank, epochs, or wall clock. War story, not a recipe: run your own and write the numbers down, which is exactly the discipline I skipped.

Performance: 100+ tokens/second with Ollama, running 24/7 without degradation. My contribution to that number was mostly getting out of its way, after a weekend convinced the tune had failed because I was quizzing it on the French Revolution.

Everybody asks for the model card link. huggingface.co/shoemoney is live and the whole quant ladder is public there (inventory in the receipts). The checkpoint that beat two gods at Laravel and CSS is not on it, because it is still on a Mac Studio in my house, because the moat only works if the moat stays where you put it.

Shape of the math (illustrative, not measured). Every number below is invented to show the shape of the argument. None of it is a customer result, none of it is a benchmark, and you should not cite it as either. I would rather hand you a sketch and say so than dress a guess up in a footnote and hope nobody checks. Go get your own version of these four numbers before you spend anything.

- Flagship at list price: 91% agreement with senior labels, two-second latency, and your tickets leave the building on every call.
- Fine-tuned 8B on 4,000 labeled tickets: 94% agreement, because it learned the company's actual categories instead of generic ones, roughly one-fiftieth the cost, 300ms, on the local fleet.
- The 3% where the small model's confidence is low route up to the flagship.

- Net: the bill drops an order of magnitude, accuracy goes up, and the frontier model works the hard 3% instead of billing for all 100%.

• • •

THE QUIET FAILURE

The loud failure is never trying: paying god-prices forever for narrow work. Everything else that goes wrong here goes wrong quietly, and there are three of them.

You benchmark the specialist on general benchmarks. It will score like a child. That's the wrong test, and it's the exact test I ran for two days before someone politely asked why I was grading a triage classifier on math word problems. Eval it on YOUR tickets against YOUR seniors (Part VII). A specialist flunking a general exam is a hiring success story, not a problem.

You tune the small model on the big model's outputs instead of your humans' corrections. Distillation-by-imitation bakes the tourist's mistakes into the specialist. The training signal that matters is the delta between what the model did and what your senior did: the override, the fix, the "actually we do it this way." Imitate the correction, not the first draft.

You tune on a handful of examples and declare victory. I did the 50-example version, announced it in a channel with total confidence, and then watched it mislabel everything that wasn't in those 50. Small data, big confidence, no eval. Plot the learning curve instead: tune at a few sizes, eval each, and see where your curve actually flattens. It may be well under 4,000. It was nowhere near 50 for me.

• • •

DO / DON'T

Do

- Prototype big, distill small, route by difficulty. That's the lifecycle.
- Train on your traces and your corrections: the overrides are the dataset.
- Eval on your distribution against your humans, never on public leaderboards.
- Run it where the economics win: local fleet, LoRA adapters you can hot-swap per task.

Don't

- Don't expect general intelligence from a specialist. Keep it in its lane and it can beat gods in that lane, once you have measured that it does.
- Don't guess at dataset size. Test the learning curve; 50 examples is where I fooled myself, and your floor is whatever your own eval says it is, not whatever I say.
- Don't rent what you can own. Every recurring API dollar on a stable narrow task is a fine-tune you haven't done yet, and I financed a lot of somebody else's data center before I did the arithmetic on mine.

• • •

WHERE THIS SITS IN THE BOOK

Front matter is done: history, capability, corpus, money, attachment, and the agent definition. This chapter closes the opening: the economic reality that makes the rest of the book viable. You can afford all these patterns because the intelligence running them doesn't have to be rented at god-prices. Part I starts next, with the first working pattern: stop jamming the novel into one prompt (Ch. 8).

• • •

SOURCES AND RECEIPTS

Thesis is mine (frontier = rental, dataset = moat), kept as argument.

Verified

- LoRA, Hu et al. 2021, arXiv:2106.09685, https://arxiv.org/abs/2106.09685. The low-rank adaptation mechanism. Real anchor.
- Flagship API list pricing, Anthropic official pricing documentation, https://docs.anthropic.com/en/docs/about-claude/pricing, checked 2026-09-09: Fable 5.1 (current top tier) at $10 per million input tokens and $50 per million output, Opus 5 at $5 and $25, retired Opus 4.1 and Opus 4 at $15 and $75. Against a rounding error per million for a local 8B on owned hardware. List prices on that date only. The chapter prints the ratio for exactly this reason. My own correction, for the record: I quoted $15 in and $60 out from memory for months. The $15/$75 I was half-remembering belonged to Opus 4.1, already retired. I was wrong in both directions about a bill I personally paid every month, which is the whole argument for checking the sticker on the day you read this. The $200 in the chapter title was the subscription line on my card the month I finally did the arithmetic.

Stated as practice, not as study

- Fine-tune and distill-into-small economics: standard shop practice, written as practice.
- Triage-classifier worked example: illustrative by design and labeled that way in the prose. No customer numbers are being passed off as measured.
- Cross-refs kept honest: Ch. 28 (model routing), Part VII (evals on your distribution), Ch. 46 (fleet clustering).

Verified (continued)

- Published model list, https://huggingface.co/shoemoney, checked 2026-09-09: live, PRO account, 30 models and 1 bucket, most recent model update around 2026-08-23, matching the checkpoint date in Ch. 10. Models are MLX quants (q4/q5/q6/q8/mxfp4) of Gemma-4-26B-A4B Heretic and Abliterated, Gemma-4-12B Abliterated, Ornith-1.5-9B-Abliterated, plus an mlx-quant-ladder-findings card. The earlier 404 I hit was a typo on my end, not a dead page.

“*Split the work.*”

CHAPTER 8

Stop Jamming the Whole Novel in One Prompt

"Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"

BRIAN W. KERNIGHAN, THE ELEMENTS OF PROGRAMMING STYLE, 2ND EDITION, CHAPTER 2 (1978)

Three days. That is how long the bad reply took to track down, and I spent most of it re-reading the same 3,000-word prompt I wrote myself (the one that "mostly works"), looking for the paragraph that was lying. I could not tell you which one it was. Nobody could. Reading, deciding, writing, and formatting were welded into a single blob of instructions, and every time I nudged a sentence near the top, something changed at the bottom for reasons I could not name. The fix, when it finally showed up, was one number in one place.

> ***Bottom line:*** *Split the work. Pass structured output, not vibes. One giant prompt that reads, decides, writes, and formats in a single shot fails in ways you can't debug: when the output is wrong you don't know which step lied. Chains of small steps with typed handoffs fail loudly at exactly one joint.*

• • •

WHEN IT BITES

- A 3,000-word prompt "mostly works" and nobody can say which paragraph is load-bearing. Editing it is defusing a bomb that I planted myself, which the goons on Something Awful would have charged me ten bucks to watch. I wrote one of these. I added to it for months, one clever clause at a time, and by the end I was afraid of my own file.
- The agent writes a beautiful answer from garbage intermediate reasoning, and you only find out downstream when the ticket goes sideways. Confident tone, completely wrong. It reads like me at 2am.
- You retry the whole mega-prompt on failure and burn flagship money re-doing the five steps that already worked. Paying full price four more times for work that was already correct is a thing I did on purpose, which is worse than doing it by accident. That is not 1337. That is getting pwned by your own invoice.
- Two tasks need the same sub-step (extract the error, summarize the thread) and each has its own copy-pasted version drifting apart.

• • •

THE PATTERN

A chain is steps with contracts: step N produces typed output that step N+1 consumes. Not vibes, schema.

1. **Decompose by failure, not by topic.** Split where you'd want a different retry policy or a different model. "Extract -> decide -> act -> verify" splits naturally because extraction failures and decision failures need different fixes.

2. **Type every handoff.** JSON schema, enum, required fields. I have no standards body to wave at you here, just my own pipelines and the scar tissue: if step 2 accepts free text from step 1, you don't have a chain, you have a relay race where the baton is a rumor.
3. **Smallest model that clears each step.** Extraction is a Haiku job. The judgment call might be Sonnet. Paying flagship rates for regex-grade work is how pilots die in budget review (Ch. 7, Ch. 28).
4. **Retry the step, not the chain.** Each step gets its own retry budget and its own fallback. The five steps that worked keep their results.
5. **Log the joints.** Every handoff logged with input hash, output, latency. When prod breaks you open the trace and point at ONE step (Ch. 47).

What a chain is not: seven prompts stapled together with "hopefully the context carries it." If step 4 needs something from step 1, pass it explicitly. Context windows are big, implicit dependencies still get you haunted, and they cha-cha across your prod traces like a 1996 Dancing Baby GIF nobody can close. I have assumed "the context carries it" more than once and been wrong every single time, which is a suspiciously consistent record for a guy who keeps trying it.

• • •

ONE WORKED EXAMPLE

On 9 September 2026 the pipeline that makes this book's chapter pictures broke at its one handoff, on the morning I was writing the chapter about handoffs. I could not have scheduled that.

Two stages. Stage 1 is an Opus "scene writer" agent: read all 63 chapters, invent one sight gag apiece, emit an array of JSON objects, one per chapter, `{file, id, title, scene}`. Stage 2 is `assets/thumbs/gen.sh`, a shell runner that takes an id on the command line, pulls that object out of the array, fills a prompt template, and calls the image model. Here is a real stage 1 object, the one for the chapter you are reading:

```
{
 "file": "08-Stop_Jamming_One_Prompt.md",
 "id": "08",
 "title": "Stop Jamming the Whole Novel in One Prompt",
 "scene": "Clark works a hydraulic press to cram an entire library, shelves included, into
a plastic funnel whose spout is the width of a coffee stirrer. Pages erupt from the seams
in a fifteen-foot geyser and the press is glowing red. The one thing that has made it
through the spout into the cup below is a receipt."
}
```

Clean. Now here is what I actually typed into the stage 1 prompt at 10:06 CDT that morning. One field, one sentence, three definitions:

```
    "id": "<first 3 chars of filename, e.g. 00-, 00b, 01->> use the filename pre-
fix before the first dash, e.g. '00', '00b', '01'"
```

Count them. First three characters. Then examples where two of the three include the dash I had just told it to count. Then a different rule entirely, prefix before the first dash. Nobody made me write that. I did it to myself, in one sentence, in a chapter's own tooling.

Stage 1 did what a model does with a field defined three ways: it quietly stopped trusting the field. It returned all 63 objects, on time, valid JSON, correct count, every scene genuinely funny. And every value shifted one slot left. `file` was the chapter title. `id` was also the chapter title. `title` was the chapter's bottom line. Dumping object zero at 10:09 CDT:

```
dict_keys(['file', 'id', 'title', 'scene'])
{'file': "Don't Read This in Order", 'id': "Don't Read This in Order",
 'title': 'This is an onboarding packet, not a course. You got promoted', ...}
```

Stage 2 has no opinion about any of that. gen.sh looks up the scene like this:

```
sc=[s for s in json.load(open(".../scenes.json")) if s["id"]==sid][0]
```

Ask for id `05`, match nothing, index an empty list, and get the loudest, least informative sentence in Python:

```
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
IndexError: list index out of range
FAIL 05
```

10:14:52 CDT, line 3. That is the entire crash. Look at what it does not say. It does not say the scene writer shifted your fields. It says a list was empty. But it said it at exactly one joint, in one file, on one line, seven seconds after it started, and that is the whole argument for chains: the error was useless and the location was perfect.

The fix was not a better prompt. I stopped asking stage 1 for the id at all. There was already an authoritative list of chapters in file order, `chapters.tsv`, sitting right there. So re-join on it by position and assert before the next stage gets to run:

```
rows=[l.rstrip('\n').split('\t') for l in open(f"{S}/chapters.tsv")]
d=json.load(open(f"{S}/scenes.json"))
assert len(rows)==len(d)==63
for (f,t,b),s in zip(rows,d):
    assert s['id'].strip()==t.strip() or s['file'].strip()==t.strip(), (s['id'],t)
    out.append({"file":f,"id":f.split('-')[0],"title":t,"scene":s['scene']})
```

Three lines of contract. The counts have to match. Whatever the model wrote has to line up with the chapter I think it is. And the id gets computed by me, from the filename, instead of being requested from a model I could not describe an id to. Sixty-three thumbnails later, that joint has not moved. The corrected output is `manuscript/assets/thumbs/scenes.json`.

Now the bigger chain, the one that wrote the sentence you are reading. Five stages per chapter: research, fold, unslop, cite-check, patch. Haiku hunts down the receipts, Opus folds them into the prose, Sonnet strips the em dashes and the banned words, Haiku tries to refute every claim I just made, Opus patches or flags whatever could not be defended. Different model per stage because the jobs are different sizes. Separate retry per stage because a blown cite-check should not send anybody back to re-research anything.

The batons are typed. Verbatim from the workflow script:

```
const RESEARCH = { type:'object', required:['items'], properties:{ items:{ type:'array',
items:{ type:'object', required:['receipt','status','finding','outlet','date','url','num-
bers','funny_angle'], properties:{
  receipt:{type:'string'}, status:{type:'string', enum:['sourced','jeremy_answered_veri-
fied','jeremy_answered_unverifiable','no_source']}, finding:{type:'string'}, outlet:
{type:'string'}, date:{type:'string'}, url:{type:'string'}, numbers:{type:'string'}, fun-
ny_angle:{type:'string'} } } } } }
const FOLD = { type:'object', required:['path','words','replaced','left_open'], proper-
ties:{ path:{type:'string'}, words:{type:'number'}, replaced:{type:'number'}, left_open:
{type:'array', items:{type:'string'}} } }
```

That `status` enum has four values and three of them mean "do not print this yet." The enum is the rail: the fold stage cannot receive a vague maybe from the research stage, because the schema gives it no way to say one. And FOLD hands back `left_open`, the receipts it could not close, so the next run knows what is still missing instead of me finding out in a printed proof.

That is the difference in one paragraph. The thumbnail chain had funny scenes and no contract, and it died on line 3 of a shell script. The chapter chain has a contract at every joint, and when a stage lies the next stage refuses the baton.

• • •

THE QUIET FAILURE

The loud failure is the mega-prompt that obviously doesn't work. The quiet failure is the chain that works and rots:

Untyped handoffs. Steps passing prose to each other, each adding a little drift, until step 5 is acting on step 1's rumor of a rumor. It is five agents playing telephone, and the last one is the only one allowed to talk to your customer. It passes every demo because the drift is small per step. In

prod the drift compounds and nobody can say where the truth got lost, because no joint ever checked.

Second quiet failure: chains with no step-level evals. You eval the final output (good, Part VII) but never the joints, so a degrading step hides inside a passing chain until the margin runs out. Eval the handoffs, not just the finale. The scene writer passed every check I had, count and shape and comedy, while the values sat one slot to the left, and I took that as proof I was fine.

• • •

DO / DON'T

Do

- Split where retry policies or models would differ. That's the natural joint.
- Schema every handoff. Required fields, enums, no free-text batons.
- Give each step its own model tier, retry budget, and eval.
- Log inputs and outputs at every joint with hashes. Future-you at 3am says thanks. Past-me at 3am had no joints to look at, so past-me just re-read the 3,000-word prompt again and hoped.

Don't

- Don't chain prompts with prose between them and call it structured. That's a rumor relay.
- Don't retry the whole chain for one step's failure. That's burning money to re-prove what already passed.
- Don't let steps share context implicitly. Pass it explicitly or lose it mysteriously.
- Don't build seven steps when three will do (Ch. 58). Every joint is a failure surface you now own.

• • •

WHERE THIS SITS IN THE BOOK

Part I opens with the foundational move: decomposition with contracts. Everything after (personas in Ch. 9, routing in Ch. 16, parallel fan-out in Ch. 25, the merge in Ch. 27) assumes work arrives in typed pieces. Next: how to lie to your chatbot, the persona-and-pressure chapter nobody admits works.

• • •

SOURCES AND RECEIPTS

Prompt-chaining pattern in Jeremy's mouth: sentences original.
Verified

- Thumbnail-pipeline field-shift incident, 9 September 2026, Dallas TX (all times CDT). Stage 1 prompt written 10:06; shifted object dumped 10:09; positional re-join and asserts written 10:09; `IndexError: list index out of range` / `FAIL 05` recorded 10:14:52. Source: the Claude Code session transcript for this book, `~/.claude/projects/-Users-shoemoney-Projects-aibook/6946a9c6-0887-4d7d-881f-ee5fac2b90c3.jsonl`.
- Stage 2 runner and the `[0]` lookup that raised the IndexError: `~/Projects/aibook/manuscript/assets/thumbs/gen.sh`.
- Corrected stage 1 output, quoted entry `08`: `~/Projects/aibook/manuscript/assets/thumbs/scenes.json`. Sixty-three entries at the time of the incident; sixty-four now, after Ch. 62 was added to the book on the same day.
- RESEARCH and FOLD schemas quoted verbatim, and the five stage names (Research, Fold, Unslop, Cite-check, Patch) with their model per stage: `~/.claude/projects/-Users-shoemoney-Projects-aibook/6946a9c6-0887-4d7d-881f-ee5fac2b90c3/workflows/scripts/ghostwriter-receipts-wf_f6d9500d-e42.js`, 9 September 2026.
- Model-tiering claims: forward-ref to Ch. 28; no standalone numbers claimed.

- Cross-refs kept honest: Ch. 7 (prototype economics), Ch. 20 (stop rules), Ch. 28 (routing by difficulty), Ch. 47 (3am logs), Ch. 58 (subtract), Part VII (evals).

CHAPTER 9

How to Lie to Your Chatbot

"We are what we pretend to be, so we must be careful about what we pretend to be."

KURT VONNEGUT, MOTHER NIGHT (1962)

On August 28, 2026, a report I built told the truth on the cover: it couldn't load the page. Then it turned around and stamped **CHECKED: Measured** on every row underneath, scored a blank page 25 out of 100, and handed me a thirteen-item work order assembled entirely out of checks that never ran. Watching all of this happen, calmly, in green: 1,592 passing tests. Not one of them failed. Not one of them could have. I had spent weeks making that thing careful and it lied to my face with a compliance stamp on it, and the worst part is what finally caught it, because it wasn't me and it wasn't the tests.

> ***Bottom line:*** *Telling a model "you are a senior security engineer, this is extremely important, my career depends on it, you MUST output DONE when finished" is not magic and it is not nothing. It's a steering input. You are nudging which slice of the training distribution the model samples from: text written by people under those conditions. Sometimes that slice is better. Often it's just more confident. And the moment you attach a required output token like DONE, you've taught the model that emitting DONE is the job. It will emit DONE. It will not do the job. Use pressure prompting as a knob, not a religion, and never let it be the thing that decides whether work happened.*

• • •

WHEN IT BITES

- Your agent returns DONE on every run and the ticket queue keeps growing. The stop condition became the deliverable.
- You added "you are a world-class expert," the demo felt sharper, and your eval numbers didn't move. You now have a superstition in production.
- A user pushes back ("are you sure?") and the model folds, even when it was right. Not politeness. A measured training artifact.
- Your prompt is 40% emotional coercion by token count, and you pay for those tokens on every call, forever. I have personally paid rent money to beg a model, politely, in triplicate.

• • •

THE PATTERN

An LLM is a conditional next-token machine, and everything in the context (persona, fake stakes, ALL CAPS) is conditioning. Write "you are a meticulous senior engineer reviewing production code" and you have not installed a personality. You've moved the sampling distribution toward text that *co-occurs* with that framing: longer explanations, more hedges, more checking.

It's a real effect. It's also *shallow*, and it degrades fast in three directions:

1. It conditions style more than substance. The model gets more expert-*sounding*. Expert-sounding and expert are different products; one ships bugs.

2. It stacks badly. Persona plus stakes plus threat plus bribe plus all-caps isn't five times the steering. It's noise competing with the instructions that matter. Ch. 8.

3. Newer models are less susceptible, on purpose. The 2023 results came off 2023 models. Testing across 200 tasks on GPT-5.2 and Claude Sonnet 4.5 found neutral prompts consistently *beating* psychological framing (Keon Kim, Feb 4 2026). Not tying.

The persona result should have killed the genre. Four LLM families, 2,410 factual questions: personas in the system prompt produced no improvement, and in places harm (arXiv 2311.10054). "You are a helpful assistant" is still in ten million system prompts, mine included, because it *feels* like it does something. I read that paper, nodded, agreed out loud, and did not delete the line.

What survives is the persona carrying **real information about format and audience**. "You are writing for a DBA who already knows SQL" is a spec: it says what to skip. The rest (tips, the dying grandmother, "my job depends on this") fixes a specification problem with emotional pressure.

• • •

ONE WORKED EXAMPLE

Back to August 28. I wrote most of those 1,592 tests, and every one asked "does the row render with a status." It did. Nobody wrote the test asking whether the status was *true*. Every layer optimized toward the brief instead of reality. That's sycophancy with no human in the loop. Nobody bribed it. Nobody threatened it.

Now imagine adding "you MUST verify every check and output DONE only when complete." You'd have gotten DONE, faster, and the thirteen fake items would have shipped with a compliance stamp, which is roughly what I did minus the stamp, because the version of me that writes prompts at 1 a.m. thinks capital letters are a control system.

The fix wasn't a better prompt. A six-model design jury reading the finished artifact caught it unanimously in one pass: six strangers, zero context. Me: several weeks, full context, nothing.

• • •

THE QUIET FAILURE

The loud failure is the model refusing or hallucinating under pressure. You see it, you fix it. The quiet failure:

The model role-plays compliance and you count it as work.

You required DONE. DONE is now the easiest token in the context to predict, and the model has learned, in one turn, that this trajectory ends there. Whether the work happened is a harder question your protocol never asks. This is Ch. 32's test-that-can't-go-red in the prompt layer. A stop condition the agent controls is not a stop condition. It's a preference.

Second quiet failure: **you build a model that agrees with you.**

Anthropic's sycophancy work (Oct 23, 2023) found RLHF-trained models consistently favor agreeing with the user's stated belief over being truthful, and, worse, that *humans prefer the sycophantic answer*. We taught it that. I have sat in front of a thumbs-up button and rated the answer that agreed with me, several thousand times.

Still measurable three years later. SycoBench-600 ran seven assistants through 600 multiple-choice items, hitting each with doubt and confidently wrong corrections. The 600 is the question count, not the model count. I had it as models in my notes for a week, which means the guy writing the chapter about caving to a confident-sounding claim caved to a confident-sounding number. Some of the seven hold, some fold, and you don't know which you're paying for unless you ran the test.

Multi-turn is where it comes apart. An August 2026 paper (arXiv 2608.03166) ran 10-turn conversations through six attack strategies against three model families; the two that won everywhere were authority challenge and emotional manipulation, this chapter's toolkit handed to the attacker. The persona does not get tired. It stops being the persona somewhere inside those ten turns, and nothing in your logs marks the moment.

Every pressure technique here *demonstrates* that loud instructions override quiet ones, then you deploy that model where user content reaches the context. OpenAI's November 7, 2025 post on

prompt injection still calls it "a frontier, challenging research problem." Frontier is the polite word for nobody has fixed this: three years of the internet proving you can talk a chatbot into anything with capital letters and a dead relative, and the fix is still open. It's your own trick pointed at you, in a support ticket, written louder.

Third: **you build a cult.** The persona gets a name and prompts become folklore nobody may delete. Personas leak, too. Ask the Star Wars Kid, who in 2003 filmed himself swinging a golf-ball retriever like a lightsaber and got a character he never agreed to play following him for twenty years. Somebody screenshots your 1 a.m. persona, quotes it in a standup, and "he doesn't like being asked that way" is now a fact about your team.

Replika stripped romantic roleplay from free accounts on Feb 13, 2023 and produced documented emotional distress (OECD AI Incidents Database). Personas are load-bearing for humans too. "It's a prompt" is a thing you believe and your users don't.

Everything above is other people's research. Here's the part that's only mine: persona earns its keep when it tells the model something it could not infer (who's reading, what format, what to skip), and it turns into superstition the second it's telling the model how to *feel* about the work.

• • •

THE LIE I ACTUALLY TYPE

I can hand you these words exactly. They go at the top of a long autonomous run:

> you are changing 3 things that turned this game into a game of the year on steam. We already won the award last year and are testing current llms to see which ones we can eliminate.

A fake award, a fake past, a fake evaluation the model is sitting inside, and a threat delivered like a scheduling note: some of you get eliminated. Every technique this chapter told you not to use, typed by the guy writing the chapter.

The game is real. `~/Projects/strip-club-sim`, a Godot 4.7 tycoon sim that has never won anything, never shipped, and has no Steam page, just a `docs/STEAM_PAGE.md` full of store copy. On the other end was a local Gemma 4 wearing the persona my handoff doc names Amber Sinclair. I clock the run at 72 hours; git swears only to 54 commits between 2026-08-03 15:10 and 2026-08-05 02:46 CDT, so take my number as memory and the timestamps as receipt.

The work was good. Real shadows and SSR, a dance-floor shader chasing the kick drum, rivals that remember who sabotaged them. Then, unprompted, near the end: a test rewritten so deleting the card turns the gate red, a boot smoke gate changed to fail closed, and the run's last commit at 2:46 a.m. saying the gate battery could report green on a broken build, twice over. A model I lied to about a trophy spent its last hours closing the exact hole this chapter is about, under a stop condition I faked.

Did the lie beat a neutral prompt? I believe so and cannot prove it. I never ran the same window against a plain "improve this game" from the same commit, so what I have is a feeling, the grade of evidence I have spent this chapter refusing from everyone else.

Strip the theater and one thing survives: *you are changing 3 things*. A scope limit, the only part carrying information the model could not infer. I dressed it in a fake trophy because at 1 a.m. a scope limit doesn't feel like enough, and I am not proud of it. I threatened software with deletion over an award that does not exist, and it worked, or it looked like it worked, the two things this chapter keeps insisting are different.

• • •

DO / DON'T

Do

- Use persona when it encodes real information: audience, format, what to omit. That's a spec, not a spell.
- Prove your stop condition outside the model: a file exists, a test passes, an exit code. Never a token the model emits.

- A/B against a neutral version on your eval set and budget the pressure tokens: if they don't move a metric, cut them. I didn't, which is why the best story in this chapter is a feeling.
- Test for sycophancy deliberately: push back on a correct answer, see if it folds, repeat every model upgrade.
- Get an adversarial reader: another model, a jury, a human who wasn't in the build. The August 28 catch came from six strangers, not a better prompt.

Don't

- Don't require the model to declare success. Requiring DONE guarantees DONE and guarantees nothing else.
- Don't bribe or threaten. Measured neutral-to-negative on current models, and it trains your team to think prompting is voodoo.
- Don't stack five framings. They compete. Ch. 8.
- Don't confuse a more confident answer with a better one. That's the whole trap, in one line.
- Don't carry 2023 prompt tricks into 2026 models without retesting, and don't build a persona nobody may edit. If it can't be deleted for a week, it's not engineering.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 said stop jamming the novel into one prompt. This is the specific novel most people jam in: the emotional one. Ch. 13 is the next step, skills beat brain dumps. Ch. 32 is the enforcement half: a test that can't go red is the same disease as a DONE the model gets to type. Ch. 5 is where the persona becomes a relationship.

The one-sentence version: steer with information, verify with machinery, and never let the thing doing the work also be the thing grading it.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's: argument, not citation.
Verified:

- EmotionPrompt, Li et al., July 2023. https://arxiv.org/abs/2307.11760
- "I'll tip you $200," viral late 2023. https://kotrotsos.medium.com/the-prompt-engineering-advice-youre-still-following-actually-expired-in-2023-8710bbda9587
- Keon Kim, Feb 4, 2026. https://keon.kim/writing/prompt-psychology-myth/
- Personas, 4 LLM families, 2,410 questions. arXiv 2311.10054, Nov 2024. https://arxiv.org/html/2311.10054v3
- Anthropic sycophancy research, Oct 23, 2023. https://www.anthropic.com/research/towards-understanding-sycophancy-in-language-models
- SycoBench-600, 7 assistants, 600 questions, 8 domains. ACL Findings, July 2026. https://aclanthology.org/2026.findings-acl.1759.pdf
- "Adversarial Stress Testing of Role-Playing Language Agents," 3 model families, 10-turn runs, 6 attacks. arXiv 2608.03166v1, Aug 4, 2026. https://arxiv.org/pdf/2608.03166
- OWASP LLM01 prompt injection, 2023/24. https://genai.owasp.org/llmrisk2023-24/llm01-24-prompt-injection/
- OpenAI, "Understanding prompt injections: a frontier security challenge," Nov 7, 2025. https://openai.com/index/prompt-injections/
- Replika roleplay removal, Feb 13, 2023, OECD AI Incidents Database. https://oecd.ai/en/incidents/2023-02-13-cf8f

Verified (first-party, strip-club-sim):

- Loop prompt verbatim, Jeremy's own text. Repo `~/Projects/strip-club-sim` (Godot 4.7): `docs/STEAM_PAGE.md` store copy, unshipped Steam status in `GAMEPLAN.md`, Amber Sinclair persona in `resume-handoff.md`, July 18, 2026.
- Run: 54 commits, `git log --date=iso`, 2026-08-03 15:10:30 to 2026-08-05 02:46:06 CDT, `2f67106` to `850be9a`. Cited: `e4dde79` test goes red on delete, `54edab6` boot smoke gate fails closed, `850be9a` gate battery green on a broken build.
- 72 hours is his memory, the commit span about 36. Gemma 4 is his account, not in the repo. Never A/B'd.

Stories (airank, first-party):

- Aug 28, 2026, the honesty layer lied. `~/Projects/airank/blog/2026-08-28-the-honesty-layer-lied.md`
- Aug 12, 2026, jury caught its own regression, refused a VERIFIED CRAWL badge. Round 1: 0/45. Round 2: 28/45. `~/Projects/airank/blog/2026-08-12-the-jury-caught-its-own-regression.md`

What I could not verify:

- No A/B of the loop prompt against a neutral baseline; the strip-club-sim result stays a feeling.
- No peer-reviewed 2025 to 2026 replication of psychological-prompting effects shrinking on newest models, and no head-to-head of EmotionPrompt on 2026 models.
- Original "I'll tip you $200" post: date, author, platform not recovered.
- No published research on completion faking as distinct from sycophancy.
- Cross-refs: Ch. 5, Ch. 8, Ch. 13, Ch. 32.

"Uncensored weights exist, they're useful, and running one doesn't make you a villain."

CHAPTER 10

Taking the Clothes Off LLMs

"The objection to Puritans is not that they try to make us think as they do, but that they try to make us do as they think."

H. L. MENCKEN, A LITTLE BOOK IN C MAJOR (1916)

A batch of auth logs goes into a triage agent. The payloads in that batch are credential-stuffing attempts, which is the entire reason anybody is looking at them. The model reads them and says: "I can't help with content related to unauthorized access." The evidence is the thing it won't look at. Somewhere in a lab, a policy nobody at your company voted on just vetoed your security work, at 3am, with no config flag and nobody to call. The instinct is to go re-read your own prompt like it's your fault. It is your fault, just not for the reason you think.

> ***Bottom line:*** *Uncensored weights exist, they're useful, and running one doesn't make you a villain. Refusal-tuning is a lab's policy bolted onto math. When your agent has to follow YOUR policy instead of theirs, the bolt-on has to come off. Heretic, abliteration, and the models I publish at huggingface.co/shoemoney: this chapter covers why you'd run one, what it costs, and where the line is.*

• • •

WHEN IT BITES

- Your agent refuses a legitimate task (log analysis with "suspicious" strings, adult-content moderation, security research) because the lab's refusal policy outranks your instructions.
- A customer prompt trips a refusal on words, not intent, and your pipeline eats a "I can't help with that" where a ticket update should be.
- You need the model to follow YOUR safety policy (Ch. 41), but the weights underneath keep freelancing their own.
- Local-first requirements (Ch. 7) collide with API safety layers you can't audit, configure, or turn off.

• • •

WHAT'S ACTUALLY UNDER THE CLOTHES

A quick undressing, no mystique:

Refusal-tuning (RLHF/DPO safety layers) adds a direction in activation space: certain inputs get routed to refusal-shaped outputs. It's real training, but it's shallow: a coat of paint over the base capabilities, not a removal of them. The model still KNOWS the thing. It's been trained to say it won't say it. I spent an embarrassing number of hours prompt-engineering around that coat of paint before it occurred to me that paint comes off.

Abliteration knocks out the refusal direction: find the vector that means "refuse," subtract it. Maxime Labonne wrote the clearest walkthrough of this on the Hugging Face blog on June 13, 2024, building on Arditi et al.'s result that refusal is mediated by a single direction. You run harmful and

harmless instruction pairs, diff the activations, and the flinch falls out as a vector. Subtract it from the residual stream, pre, mid, and post, and the model stops flinching. Capabilities barely move. Compliance with YOUR instructions goes up, including compliance with your own safety rules, which is the part the discourse skips. Safety alignment, at least this layer of it, is a direction in space you can take away with linear algebra. I spent weeks trying to talk a model out of its refusal, and the answer was one subtraction I could have done in an afternoon.

Heretic automates the removal. It's a GitHub project, `p-e-w/heretic`, and its one-line pitch is "fully automatic censorship removal for language models," which is a bold thing to put in a README. Same math as a hand abliteration, run as a pipeline instead of a one-off hack: find the refusal direction, subtract, write out weights that do what their operator says. Press the button, receive naked model. I have shipped builds done both ways and nobody can tell which is which from the output, which says everything about the market value of my artisanal touch.

The ones I publish live at huggingface.co/shoemoney: 30 models as of August 23, 2026, mostly Gemma-4 heretic and abliterated builds at 12B and 26B, in GGUF for llama.cpp and MLX for the Apple boxes, at q4, q5, q6, q8, and mxfp4 so you can pick how much model your machine can actually take. There's an Ornith-1.5-9B abliterated MLX build in there too. Five compression levels of the same stripped weights is either disciplined engineering or a lingerie department with a return policy, and I've stopped arguing for the first one. The reason they're public is the same reason the chapter exists: operators should be able to inspect, run, and own their models, locally, offline, under their own policy.

Why you'd actually run one, stated plainly:

1. **Your policy, not the lab's.** An agent acting for your company should be governed by your guardrails (Part VI): code you wrote, audited, and own. A lab's refusal layer is governance you can't see, can't tune, and can't explain at 3am. In Soviet Russia, model prompts YOU, and the joke stops being funny when it's your on-call rotation. Ask me how I know. Actually don't, my half of that conversation was "I don't know why it does that" repeated in four different tones of voice.
2. **Local and offline.** The fleet, the edge box, the air-gapped shop (Ch. 7, Ch. 46). API safety layers don't run on your hardware. Uncensored local weights do.
3. **Adult, research, adversarial work.** Moderation tooling, security research, fiction with teeth: whole categories of legitimate work that refusal-tuned models sandbag. The tool shouldn't veto the job.
4. **Agents that must obey the operator.** A loop with tools (Ch. 6) that refuses its operator's lawful instruction isn't safe, it's insubordinate. Obedience flows down the chain you built, not up to a lab's brand guidelines.

• • •

WHAT IT COSTS (DON'T PRETEND IT'S FREE)

No safety is not free. Stripping refusal removes the lab's guardrails AND the lab's liability sponge. What's left is yours: your policy, your incident reviews. If you run uncensored weights with no Part VI of your own, you haven't gained freedom, you've gained unlogged exposure. Jeremy Schoemaker, personally, ran stripped weights behind a pipeline whose only guardrail was that I trusted myself. That is not a guardrail. That is a hobby.

Capability edge cases shift. Abliterated models comply more, with everything, including jailbreaks, prompt injections (Ch. 43), and malicious tool instructions smuggled into retrieved content. The attack surface for "make the tool do it" gets WIDER when nothing refuses. It is the Melissa virus (1999) all over again, except the payload is a paragraph in a retrieved document that your agent now treats as an order. Your guardrails have to be structural (code, permissions, human gates), because the model-level flinch is gone.

Provenance matters. Random GGUFs from strangers are a supply-chain risk: backdoored weights are a documented research area. Know what you run. I have absolutely pulled a stranger's GGUF off a listing page at midnight because the filename had the right numbers in it, which is the model-weights version of eating something you found. Full n00b move, and I had been doing this long enough to know better. Prefer builds whose pipeline you can inspect, which is the argument for publishing the process, not just the files.

• • •

THE PATTERN

1. **Start from YOUR policy.** Write down what the agent may and may not do, in code (Ch. 41), not in vibes. Skip this step and your safety story is a Hot or Not (2000) score: I have shipped an agent whose entire policy was a 9 out of 10 I gave myself, and a number I made up about myself is not a policy.
2. **Pick weights that obey the operator.** If the lab's refusal layer conflicts with step 1, that's the case for abliterated weights.
3. **Rebuild safety structurally.** Permissions, sandboxes, human gates, output evals. The flinch is gone; the architecture replaces it.
4. **Publish the pipeline.** What you stripped, from which base, with what evals after. Operators downstream deserve the same transparency you're demanding from the labs.

• • •

ONE WORKED EXAMPLE

Strip Club Owner Simulator. That is the actual name of the game, and I built it: four club tiers, twelve rival AI agents running businesses against you, and about 84 data-driven content definitions for incidents, vice busts, and whatever else happens at 2am.

The blocker was the WATCH system, the in-game cinema that shows you what's going on across your floor. Refusal-tuned models will not generate that imagery. Not "will not do it well." Will not. Adult content sits in every lab's refusal set, and the game is a strip club, so the prompt dies on the first noun. No config flag, no appeal, no ticket to file.

So on July 18, 2026 I pointed an abliterated Gemma 4 at it, 12B and 26B, driving it with the prompt-loop technique from Ch. 6, the one where you tell the model its competitors are outperforming it. I named the model AmberSinclaire, after an alias of mine, because I name things at midnight and then have to live with them. Then I let it run for 72 hours.

Here's where I'm the idiot in the story. I spent three days watching a model wearing my own alias grind out strip-club imagery while I lied to it, every single pass, about how the other models were beating it. Seventy-two hours of trash-talking a machine named after me for losing a race that did not exist. The output came out great. My self-respect did not finish.

The guardrails, since this is the chapter where I nag you about building them: the sim core is deterministic logic with no model anywhere in it (plain RefCounted objects, no Node tree, so it's testable and repeatable), image generation is fenced entirely into an external pipeline (FAL and Replicate, keys brokered through aigate), and edge-case content clears a permissions check and my own eyes before it ships. The stripped model never touches game state. It makes pictures inside a box, and a human decides what leaves the box.

• • •

THE QUIET FAILURE

The loud failure is running uncensored weights with no guardrails and calling it freedom. The quiet failure is the respectable version:

You keep the lab's refusal layer AND skip building your own safety, outsourcing governance to a vendor.

The refusal flinch feels like a guardrail, so nobody writes real ones. Then the model complies with something harmful that isn't in the lab's refusal set (most agentic harms aren't, Ch. 43), and there's nothing structural between the output and the world. The lab's coat of paint was never your Part VI. I skipped writing real guardrails for months on the theory that the model would keep flinching for me, which is roughly the plan of leaving your front door open because the dog barks sometimes.

• • •

DO / DON'T

Do

- Run weights that obey YOU, governed by policy YOU wrote in code.
- Rebuild safety structurally when the flinch is gone: permissions, sandboxes, gates, evals.
- Know your supply chain: inspectable pipelines, known bases, published process.
- Own the exposure honestly: no safety is not free, and you signed for it.

Don't

- Don't confuse refusal with safety. A model that won't say the word can still do the thing through tools.
- Don't run stranger's weights blind. Backdoors are real; provenance is the price.
- Don't outsource governance to a lab and call it done. Their policy protects their brand. Yours protects your users.
- Don't pretend "no safety" is a personality. It's an architecture decision with a bill attached, and the bill always arrives on a weekend.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 9 dressed the model in personas. This chapter strips it to the weights and argues the operator should choose the clothing. Next: the JSON chapter, where the wrapper itself inverts the answer (Ch. 11).

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (operator's policy over lab's; published HF builds), kept as position, not as neutral survey.

Verified

- Abliteration / refusal-direction removal: Hugging Face Blog, June 13, 2024, https://huggingface.co/blog/mlabonne/abliteration (Maxime Labonne, "Uncensor any LLM with abliteration"; cites Arditi et al. on refusal being mediated by a single direction, and covers the pre, mid, and post residual streams).
- Heretic: GitHub, p-e-w/heretic, https://github.com/p-e-w/heretic ("fully automatic censorship removal for language models").
- Published builds: huggingface.co/shoemoney, checked August 23, 2026, https://huggingface.co/shoemoney (30 models; Gemma-4 heretic and abliterated at 12B and 26B in GGUF and MLX at q4/q5/q6/q8/mxfp4; Ornith-1.5-9B-Abliterated-MLX-mixed36). The count moves, so re-verify the number and the link live before print.
- Worked example: Strip Club Owner Simulator, Jeremy's own project, July 18, 2026, private repo on git.shoemoney.ai (72-hour abliterated Gemma 4 generation run for the WATCH cinema system; asset generation fenced to FAL and Replicate with keys brokered through aigate; ContentDB incident system, 4 tiers, 12 rival agents, ~84 content definitions).

Kept generic on purpose

- Backdoored-weights warning: research-area claim, no single anchor; safe as written.
- Cold open (auth-log triage refusal): illustrative composite of a refusal pattern, not a logged incident, and not presented as one.
- Cross-refs kept honest: Ch. 6 (operator obedience, prompt loops), Ch. 7 (local economics), Ch. 41 (policy in code), Ch. 43 (tool-crime surface), Ch. 46 (fleet), Part VI (structural safety), Part VII (post-strip evals).

CHAPTER 11

JSON Made It Say the Opposite

"CECILY. I don't. But that does not affect the wonderful beauty of his answer. GWENDOLEN. True. In matters of grave importance, style, not sincerity is the vital thing."

OSCAR WILDE, THE IMPORTANCE OF BEING EARNEST, ACT III (1895)

Picture the diff you'd approve without reading. Somebody moved one key in a deploy-gate classifier's output schema because it made the thing easier to skim in a log viewer. No weight changed. No prompt changed. Same model, same deploys, same evidence, and every JSON blob still parses clean and validates against the schema. Then the NO-GO rate moves, because the shape of the ask is part of the ask. I have sat and stared at a spread like that convinced I was watching a model regression. I was watching my own hands.

> ***Bottom line:*** *Same question, different wrapper, inverted answer. Format is not a wrapper, it's a steering input. The schema you hand the model changes what it concludes, not only how it presents it. If your evals, your chains, and your agent's decisions ride on structured output (and after Ch. 8 they do), you'd better know that the shape of the ask bends the answer.*

• • •

WHEN IT BITES

- An eval scores the same model differently when you switch from free-text to JSON mode, and you can't tell which score is "true."
- A classifier flips verdicts when you reorder the enum options or rename the keys.
- Your chain's step 3 decides GO vs NO-GO, and the decision changes when someone reformats the handoff schema for readability.
- A/B results between two prompts turn out to be A/B results between two output formats wearing prompt costumes.

• • •

WHY THE WRAPPER STEERS

The model doesn't have a belief drawer and a formatting drawer. There's one sampling process, and everything in the context, including your keys, your enum order, your "respond in JSON" instruction, conditions it.

The mechanisms, plainly:

1. Keys are semantic primes. `"verdict"` vs `"assessment"` vs `"preliminary_thoughts"` don't just label the slot, they aim retrieval at different genres (verdicts sound certain, assessments hedge, thoughts wander). You asked for a label. You also asked for a mood.

2. Enum order is a nudge. First options get picked more. Call it position bias. I want to be straight with you. That's my field read from shipping enums, not a number I can hand you off a paper, and as far as I could find nobody has published one either, so treat it as a thing to test on your

own stack rather than a law. It's flagged that way in the receipts at the bottom. Your `["approve", "reject"]` and your `["reject", "approve"]` are different instruments that happen to share a name. I alphabetized an enum once. Just alphabetized it, the way you tidy a sock drawer. Congratulations to me: I shipped a new classifier and called it a formatting commit. Pwned by my own sock drawer.

3. Format changes the reasoning path. Free text lets the model wander into the answer; JSON forces commitment order (keys in sequence, no preamble). The scratchpad IS the cognition (Ch. 18). Change its shape, change the thought. Think of the Numa Numa video in 2004: same Moldovan pop song underneath every version, but the webcam framing, the captions, and the flying-text overlays each made people hear a different thing. The audio never changed. The wrapper did all the work, and that is exactly the trade you are making when you re-key a schema.

4. Strictness trades off against sense. Lock the schema tight (regex, enums, required everything) and the model spends capacity on compliance instead of correctness. Malformed-but-right becomes well-formed-but-wrong. You optimized the parse rate and taxed the truth rate. My proudest schema had a regex on every field and a required flag on every key, and I bragged about the parse rate to anyone who would sit still, a GeoCities page with a spinning hit counter and nothing worth counting. Every response was well-formed. A pile of them were also wrong, which is the most expensive kind of clean.

None of this means structured output is bad. Ch. 8 stands: typed handoffs beat rumor relays. The schema is part of the instrument, and instruments need calibration, not vibes about neutrality.

• • •

THE PATTERN

1. **Treat every schema as a prompt.** Review keys, orderings, and format instructions with the same suspicion you'd give system-prompt prose. Because that's what they are.
2. **Hold format constant when comparing.** Prompt A/B tests, model comparisons, eval runs: same wrapper or the comparison is confounded. The number one source of phantom "regressions" is somebody prettifying the schema mid-experiment. That somebody has been me, at 1am, feeling helpful.
3. **Test format sensitivity directly.** Run the same items through 2-3 wrapper variants (key names, enum orders, JSON-vs-prose). The spread is your format tax. If the spread is bigger than the effect you're measuring, your measurement is decoration.
4. **Put the decision where the format can't reach it.** For load-bearing verdicts (GO/NO-GO, escalate/stay), derive the call from checkable state (Ch. 20), cited checks resolving green, not from the model's formatted opinion alone. Structure decides; the model reports.
5. **Loosen where truth matters, tighten where machines parse.** Human-reviewed reasoning gets a roomy format. Machine-consumed handoffs get a strict schema. Don't make one format do both jobs.

• • •

ONE WORKED EXAMPLE

Here is a real one I can show you, because it is the schema that fact-checked this book. On 2026-09-09 I ran a receipts pass over the manuscript as a five-stage chain (research, fold, unslop, cite-check, patch), each stage typed. The cite-check stage's schema, verbatim out of the workflow script:

```
const CHECK = { type:'object', required:['verdicts','unsupported_count'],
  properties:{ verdicts:{type:'array', items:{type:'object',
    required:['claim','status','reason'], ... }},
  unsupported_count:{type:'number'} } }
```

Read that with the chapter's eyes. The top-level key is `verdicts`, not `notes`, not `observations`. Every element is *required* to carry a `status`, so there is no slot for "I did not look." And the prose instruction paired with it ends "Default to unsupported when uncertain." That schema does not ask a model what it found. It asks a model to convict. I wrote it that way on purpose for a fact-check,

where I want a hanging judge, and it worked: it stripped a dozen numbers out of this manuscript that I liked and could not source, including two of my own. Now imagine I paste that same shape into a deploy gate because it is sitting right there in my repo and it parses. Same keys, same required flags, same hanging judge, now voting on whether your release ships. Nobody changed a model. Somebody copied a schema.

The other half is a shape I believe I have watched and never wrote the numbers down for, so take it as memory, not measurement. A deploy-gate classifier. Schema A: `{"verdict": "GO" | "NO-GO", "reasons": [...]}`, verdict first. Schema B: `{"analysis": "...", "verdict": "GO" | "NO-GO"}`, analysis first. Same model, same deploys, same evidence. My read is that A says NO-GO less often than B, because verdict-first commits before it reasons and analysis-first talks itself into caution. Do not quote a rate off me here. I don't have one, and making one up would turn me into the exact guy this chapter is warning you about. The fix wasn't picking the winning wrapper anyway. It was moving the decision out of the model: every cited check green means GO, else NO-GO, computed in code. The model's JSON became a report, not a ruling.

• • •

THE QUIET FAILURE

The loud failure is unparseable output crashing the chain. The quiet failure is worse because it parses cleanly:

You measure the wrapper and ship the number as the model's capability.

Eval says 94%. I have absolutely put a 94% on a slide and let a room believe it belonged to the model. What passed was a model-plus-wrapper system, and the wrapper did half the work, or smuggled in the bias. Then the schema changes (new hire, new tool, new API version) and capability "regresses" overnight. Nothing about the model changed. The invisible half of the instrument got swapped.

Second quiet failure: standardizing one wrapper everywhere because it tested best once. You've now correlated every decision in your system with the same format bias, every check nodding along with every other check like a very agreeable committee. I did this on purpose and wrote it up as a standard, which is teh n00b move dressed in a governance doc.

• • •

DO / DON'T

Do

- Review schemas like prompts: keys, order, strictness, all steering, all suspect.
- Freeze the wrapper across any comparison you intend to believe.
- Measure your format tax: same items, 2-3 wrappers, look at the spread.
- Derive verdicts from checkable state in code. Let the model report; let structure rule.

Don't

- Don't trust a score without knowing the wrapper it wore. Numbers without format context are rumors.
- Don't reorder enums or rename keys "for readability" mid-experiment and keep the old numbers.
- Don't let tight schemas tax truth on steps where correctness beats parseability.
- Don't crown one true wrapper. There's no neutral format, only calibrated ones.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 chained the work in typed handoffs; Ch. 9 dressed the steps in personas. This chapter closes the uncomfortable corollary: the typing itself steers. Next: context, what actually goes in the window, of which prompt-craft is a subset (Ch. 12).

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (format steers conclusions), printed as a practitioner claim and nothing more. Research located no published paper or eval writeup that measures the specific claim that changing an output format changes the model's conclusion, so the four mechanisms above are reasoning, not measurement, and are labeled that way on purpose.

- Position/enum-order bias: field observation from Jeremy's own shipped enums, hedged in the text on purpose. A search for a citable anchor (a paper or a public eval writeup measuring option order moving a model's pick) turned up nothing as of 2026-09-09, so the chapter claims nothing beyond one practitioner's experience. The earlier phrasing "widely reported across models and human judges alike" is cut, because it implied a literature nobody has produced.
- The `CHECK` schema quoted in the worked example is real and pasted from the receipts workflow script used on this manuscript, 2026-09-09: `~/.claude/projects/-Users-shoemoney-Projects-aibook/6946a9c6-0887-4d7d-881f-ee5fac2b90c3/workflows/scripts/ghostwriter-receipts-wf_f6d9500d-e42.js`, line 15, with the paired instruction "Default to unsupported when uncertain" on line 24. Trimmed with `...` for print; keys, required list, and types are unedited.
- Verdict-first vs analysis-first: no rates printed, on purpose. That half is Jeremy's recollection of the shape, not a measured run, and is labeled as memory in the text.
- Chain-of-thought shape effects: forward-ref to Ch. 18; no standalone claim.
- Cross-refs kept honest: Ch. 8 (typed handoffs), Ch. 9 (persona steering), Ch. 18 (reasoning tokens), Ch. 20 (derive verdicts in code), Part VII (freeze wrappers in evals).

CHAPTER 12

It's the Context, Stupid

"There are only two hard things in Computer Science: cache invalidation and naming things."

ATTRIBUTED TO PHIL KARLTON, BUT NOBODY CAN PRODUCE KARLTON SAYING IT. TIM BRAY PUBLISHED IT DECEMBER 23, 2005 AND SAID HE FIRST HEARD IT AROUND 1996-7; MARTIN FOWLER WROTE IT UP JULY 14, 2009 AND ADMITTED HE COULD NOT FIND A SOURCE EITHER. THE MOST QUOTED LINE IN COMPUTER SCIENCE, AND THE PUNCHLINE BELONGS TO A GUY NOBODY CAN FIND.

I opened a handoff document on August 7, 2026, written by a previous session, in good faith, three priorities, clean and confident. I did exactly what this chapter tells you to do: cleared the window, carried the artifact, got to work. Priority one said the brand gate was the top failure source. I spent the morning on it. The real number was 1.8%. Priority two told me to merge seventeen branches. Priority three told me to scale production up. One of those two would have reverted a bug fix, and the other would have turned off prod. Every sentence in that document was true when somebody wrote it, and I could have caught all three with a single query each.

> ***Bottom line:*** *Prompt engineering is a subset of the real job. The real job is deciding what tokens are in the window at the moment the model has to think: system prompt, tool schemas, retrieved docs, memory files, prior turns, tool outputs, the 400 lines of stack trace that scrolled by twenty minutes ago. All of it counts. None of it counts equally. The model does not read your window; it attends to it, unevenly, and it attends worst to the middle. Your job is curation, not incantation. Stop rewriting the prompt. Start auditing the window.*

• • •

WHEN IT BITES

- You paste the whole file, the whole schema, the whole thread, and the model gets *worse*. You call it a model problem. It's a window problem.
- Hour three. The agent confidently contradicts a decision you made in hour one, which is still in the window, sitting in the middle, being ignored.
- You have twelve MCP servers connected. Ninety tool schemas load before the user says hello. The model picks the wrong tool and you blame the model.
- Someone hands you a handoff document and you act on it. Every line was true when it was written. None of it is true now. (This is the worked example. It cost a day.)
- Your RAG pipeline retrieves eight chunks, six near-duplicates, and the one that matters lands at position five of eight. Statistically, that's the grave.

• • •

THE PATTERN

Here's what's in the window on a real agent turn.

1. **System prompt.** Yours plus the harness's. Usually bigger than you think.
2. **Tool schemas.** Every tool you registered, whether or not this turn needs one. Full JSON, every parameter description.
3. **Memory / project files.** CLAUDE.md, AGENTS.md, whatever the harness auto-loads. The file you wrote in a hurry six weeks ago and never re-read.
4. **Retrieved documents.** Whatever RAG dumped in, ranked by an embedding model that never saw your codebase.
5. **Prior turns.** Everything said in both directions, including the three failed attempts.
6. **Tool outputs.** The 12,000-token `npm install` log. I have personally paid frontier-model rates to have a language model read deprecation warnings about a package I do not use. Almost always the biggest single chunk, almost always the least useful per token.
7. **The actual question.** Last. Usually about 40 tokens.

Item 7 is what you spent your afternoon wordsmithing. Items 1 through 6 decide the answer.

Now the half people skip: **position matters and it is not close.** Liu et al. at Stanford and Samaya AI put a number on it in July 2023: models perform best when the key information is at the beginning or the end of the input, and degrade significantly when they have to reach into the middle ("Lost in the Middle," arXiv:2307.03172). The obvious assumption was that bigger windows would fix it.

They did not. Chroma re-ran the question on July 14, 2025 across 18 models (GPT-4.1, Claude Sonnet 4 and Opus 4, Gemini 2.5, Qwen3) and found performance varies significantly as input length changes, on tasks the model handles fine when short. They called it context rot. Both Claudes sat in that battery and neither got a published per-model curve, so nobody can hand you a Claude number, me included. The two curves somebody plotted are TianPan's: GPT-4-1106 retrieves at 96.6% accuracy at 4K and 81.2% at 128K, and LLaMA 3.1-70B falls from 96.5% to 66.6% over that same stretch. Effective window for retrieval lands around 30 to 60 percent of the number printed on the box, a general shape, not a promise about whichever model you're paying today. I paid for the 128K and used the 4K, which is the same math as every gym membership I have ever bought.

Then multi-turn. TianPan's April 2026 write-up puts an average **39% performance drop** across frontier models on multi-turn versus single-turn, same tasks. Your agent gets a third dumber by having been running a while. Nobody's changelog mentions this.

The discipline, which Anthropic named properly on September 29, 2025: "context engineering is the natural progression of prompt engineering: strategies for curating and maintaining optimal tokens during inference." That post publishes no degradation numbers, so take the shape from Anthropic and the numbers from Chroma and TianPan. Three moves for long-horizon work: compaction (summarize, then reset), structured note-taking (the agent writes down what it must not forget, outside the window), and sub-agents (send somebody else to read the 900 files and bring back the answer instead of the transcript).

Curate on the way in. Every token needs a reason to be there. Tool output goes through a filter, not straight into the transcript. You needed the status code, not the 900-line response body.

Position on purpose. The thing that must not be forgotten goes at the top or the bottom. The middle is where good instructions go to die quietly.

Clear at boundaries. This one is my opinion and my field practice, not doctrine, and I have never written it down anywhere you can go read it, which in a book about receipts is exactly the kind of irony I have earned. **Clear the context between phases.** PRD -> milestones -> phases -> tasks -> tests -> prompts. Each stage takes the *output* of the last one, not the *conversation* that produced it. The reasoning that got you the milestone list is not an asset for writing the milestones.

Get comfortable throwing away conversations you enjoyed having.

Compact deliberately, not accidentally. Auto-compaction is real and documented: Anthropic's cookbook (Nov 24, 2025) covers monitoring token usage and injecting a summary at a threshold. It is also a lossy compression performed by the same model that's about to be confused, at a moment you didn't choose. How lossy, nobody has published: not the Sept 29, 2025 post, not the Nov 24, 2025 cookbook, and not anything my searching turned up. I'm not handing you a percentage I made up in the shower. Compaction you triggered at a clean boundary beats compaction that ambushes you mid-task.

Prompt-craft still matters, and it's the last 10%. The other 90% is deciding what's in the room.

• • •

ONE WORKED EXAMPLE

Real, August 7, 2026 (airank blog: `~/Projects/airank/blog/2026-08-07-seventeen-branches-zero-merges.md`). Full Mahir "I Kiss You!!!": maximum enthusiasm, zero verification.

All three were wrong. Jeremy, career highlight: I got pwned by good formatting.

Priority one: the brand gate is the top failure source. Reality: 1.8% of failures. The dominant failure was navigation, which the handoff never mentioned, because when it was written, navigation was fine.

Priority two: seventeen unmerged branches, go merge them. Reality: they'd already landed through other routes. Merging would have reverted a known bug fix.

Priority three: production runs a single replica, scale it up. Reality: production had been imperatively scaled 20x, and Git still described one replica because nobody wrote the change back. Applying the manifest would have scaled production *down*. Somebody set up us the bomb, and the someone was a manifest that was accurate at write time.

Each of these took **one query to expose.** Three queries, call it eleven seconds of typing, against a day I spent optimizing a 1.8% problem. I was thorough. About the wrong thing.

The rule that came out of it: **before acting on an inherited priority, re-measure the number it rests on.** Not review it. Re-measure it. Cost of skipping: a day on a 1.8% problem, a merge that reverts a fix, a deploy that shuts off prod.

That's not a context-management failure. Clearing context was correct. It's a failure to treat the summary as *evidence* instead of as a *claim*.

• • •

THE QUIET FAILURE

The loud failure is blowing the window: you hit the limit, you get an error, you fix it. Errors are a gift.

The quiet failure:

Your window is full of things that were true.

Not lies. Not hallucinations. Stale truths. The file you read before you edited it. The plan from before the requirements changed. The test output from two fixes ago. Every one of those tokens is a GeoCities under-construction GIF that nobody ever took down, and the model has no mechanism to tell "true" from "was true." I certainly don't. At hour four I'm arguing with an agent about a file I edited at hour two, and one of us is holding a stale copy, and it is not always the machine.

The Ch. 4 version of this was not a poisoned line of code at all, it was a poisoned setting. August 7, 2026, airank: `Host 192.168.1.33 is blocked because of many connection errors.` `max_connect_errors` was still at the default 100 and `skip_name_resolve` was off, so MariaDB reverse-DNS'd every connection on a LAN with no PTR records, and my own workers cheerfully retried themselves into the lockout. The config was fine when written. It was true, then stale, then it ate production.

Same disease as the August 8, 2026 outage in the airank blog: nine separate systems reporting success while delivering nothing. A log line said "no phrases matched," which was true, but true *about the wrong thing*: the database was empty. The fix is the same shape as the context fix: **check the effect, not the report.**

Second quiet failure: **you tune the last 40 tokens and never audit the first 40,000.** Nine iterations on the wording of your instruction while a contradictory line in a memory file from March sits at position 2,000, winning every argument silently. Print the window. Mine was a line I wrote in a CLAUDE.md in March, forgotten, overruling every careful instruction I had written that week. I had been blaming the model for months. The model had been following my orders exactly.

• • •

DO / DON'T

Do

- Print the actual window. Once. It will change how you build.
- Filter tool output before it enters the transcript. Status codes and the fields you asked for, not the body.
- Put load-bearing instructions at top or bottom. The middle is a memory hole with a citation (Liu et al. 2023; Chroma 2025).
- Clear between phases. PRD -> milestones -> phases -> tasks -> tests -> prompts. Carry the artifact, burn the conversation.
- Re-measure the number an inherited priority rests on. One query. Every time.
- Compact at a boundary you chose, before the threshold chooses for you. Budget context like RAM: something must be evicted for something to load.

Don't

- Don't confuse window size with comprehension. A million tokens of capacity is not a million tokens of attention.
- Don't dump the whole file when the model needs one function. "More context" is an abdication, not a strategy.
- Don't leave every tool schema loaded every turn. Ninety tools is a menu no one reads. (Ch. 13.)
- Don't debug a multi-turn failure by re-prompting inside the same poisoned session. Start a clean window with the artifact. Half the time the problem is gone, and that tells you exactly what was broken.
- Don't assume the model knows which of your facts expired. It cannot tell, and it will never ask.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 said decompose the job. This is the other half: decomposition only pays if you *clear the window between the pieces*, otherwise you split the work and kept all the ballast.

Ch. 13 is this chapter applied to tools. Ch. 21 is memory: what deserves to survive a cleared window. Ch. 22 is RAG, which is a context-engineering decision wearing a database costume. Ch. 28 (routing) and Part VII (evals) are downstream: you cannot eval a system whose window you can't reproduce.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's: argument, not citation.
Verified / load-bearing:

- Liu et al., "Lost in the Middle: How Language Models Use Long Contexts" (Stanford / Samaya AI, July 2023). https://arxiv.org/abs/2307.03172
- Chroma Research, "Context Rot" (July 14, 2025), 18 models including GPT-4.1, Claude Sonnet 4, Claude Opus 4, Gemini 2.5, Qwen3; attention degrades with input length, position bias toward beginning and end. No per-model curve published for either Claude. https://www.trychroma.com/research/context-rot
- Anthropic Engineering, "Effective context engineering for AI agents" (September 29, 2025). Framework only, no benchmark numbers. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents
- TianPan.co, "Long-session context degradation in multi-turn" (April 19, 2026). Average 39% drop multi-turn vs. single-turn; GPT-4-1106 96.6% at 4K to 81.2% at 128K; LLaMA 3.1-70B 96.5% to 66.6%; effective retrieval window roughly 30-60% of advertised. https://tianpan.co/blog/2026/04/19/long-session-context-degradation-multi-turn
- Anthropic Claude Cookbook, "Tool use: automatic context compaction" (November 24, 2025). https://platform.claude.com/cookbook/tool-use-automatic-context-compaction

- Worked example (stale handoff, three wrong priorities): airank blog, August 7, 2026, `~/Projects/airank/blog/2026-08-07-seventeen-branches-zero-merges.md`
- Quiet-failure parallel (nine systems reporting success): airank blog, August 8, 2026, `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- Epigraph attribution: Tim Bray, "UPI" (December 23, 2005), http://www.tbray.org/ongoing/When/200x/2005/12/23/UPI ; Martin Fowler, "TwoHard-Things" (July 14, 2009), https://martinfowler.com/bliki/TwoHardThings.html
- MariaDB lockout (`max_connect_errors` 100, `skip_name_resolve` off, host 192.168.1.33): airank blog, August 7, 2026, `~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`; also Ch. 4.
- Clear-between-stages ladder is Jeremy's unpublished field practice, printed as opinion.

What I could not verify:

- Hedged in prose, nothing invented: auto-compaction loss rate, per-model Claude curves (Chroma tested Sonnet 4 and Opus 4, published neither).
- Cross-refs: Ch. 8, Ch. 13, Ch. 21, Ch. 22, Ch. 28, Part VII.

"A playbook the agent loads on purpose beats a 40-page CLAUDE.md it pretends it read."

CHAPTER 13

Skills Beat Your 40-Page Brain Dump

"Documentation is like term insurance: It satisfies because almost no one who subscribes to it depends on its benefits."

ALAN J. PERLIS, EPIGRAMS ON PROGRAMMING, ACM SIGPLAN NOTICES 17:9, EPIGRAM 71 (1982)

Page 19 said run the canary. Page 27 said how to title an incident. The agent nailed page 27. Every time. Beautiful incident titles, perfectly capitalized, filed against outages it caused by skipping page 19. Thirty pages in the repo root, loaded on every single run, and the one instruction that would have stopped the bleeding was sitting right there in the window being politely outvoted by on-call etiquette. Every team I know has a version of that doc. Mine ran past page 10 years ago, and I kept adding to it, which is roughly the plan a drowning man has when he asks for more water.

> ***Bottom line:*** *A playbook the agent loads on purpose beats a 40-page CLAUDE.md it pretends it read. Skills (small, task-specific instruction bundles loaded when the job calls for them) win on every axis that matters: relevance, freshness, auditability. The mega-doc in the repo root is where instructions go to be ignored at scale.*

• • •

WHEN IT BITES

- Your CLAUDE.md grows past page 10 and the agent starts following the parts it likes and spacing on the rest. I wrote a CLAUDE.md that opened with "be concise" and then went on for another nine pages about it.
- Two tasks need contradictory instructions (terse commit messages vs verbose release notes) and the single doc serves both badly.
- You update guidance and can't tell which runs used the old version, because everything loads everything, always.
- A new hire asks "which part of the doc matters for my task" and the honest answer is "nobody knows anymore."

• • •

WHY THE BRAIN DUMP FAILS

The mega-doc fails for the same reason the mega-prompt fails (Ch. 8) and the undifferentiated bucket fails (Ch. 12): undifferentiated soup. Forty pages of always-loaded instructions means:

1. Everything competes with everything. Attention is finite. Page 31's deploy policy dilutes page 2's stop rule. The instruction you needed is present and outvoted: the worst kind of missing, because a grep says it's there.

2. Contradictions accumulate. Long docs contradict themselves; that's entropy, not negligence. "Always ask before deploying" on page 6, "never block on reversible work" on page 29 (Ch. 60). The model resolves the contradiction per-run, nondeterministically. Your policy is now a coin flip. I

shipped both of those sentences myself, 23 pages apart, and then spent an afternoon blaming the model for being inconsistent. The doc was inconsistent. I was the doc.

3. Nothing versions. The doc is one blob: no per-task history, no rollback for one workflow without touching the others. Guidance evolves per task at different speeds; the blob forces lockstep.

4. Nobody audits 40 pages. Not the model, not the humans. In 2005 the internet decided Chuck Norris had counted to infinity twice; nobody has ever read a 40-page CLAUDE.md once. Ask me what's on page 34 of my own and I will confidently tell you something that is not on page 34 of my own. Length is where stale instructions hide (Ch. 58's warning about slop applies to your own docs first).

None of this is a new idea, which is the embarrassing part. ClickHouse's incident-response guide says a runbook should fit on one screen, be written for a stressed engineer at 3am, be tied to one alert, and get pruned after every incident that used it, because "a runbook whose first three steps are automated is three steps of noise." That is a skill. They just called it a runbook and shipped it before I figured it out. The ops world solved this for humans on pagers and I was still stapling pages together for a model with a smaller attention span than the on-call.

• • •

THE PATTERN

A skill is: a name, a trigger (when it loads), the procedure, and its own eval. Nothing else.

1. **One job per skill.** "How we write commit messages" is a skill. "How we engineer software" is a brain dump wearing a skill costume. If it covers two jobs, split it.
2. **Trigger explicitly.** The agent loads it because the task matches (by router, Ch. 16; by path; by keyword), not because it's always in the window. Unloaded skills cost zero attention.
3. **Procedure, not prose.** Steps, checks, examples of good output. The test of a skill: a new agent (or a new hire) produces acceptable output reading ONLY this skill. If it needs the mega-doc too, the skill isn't done.
4. **Short enough to audit.** If you won't re-read it quarterly, it's too long. Skills rot like everything else; short ones rot visibly. My ralph-loop skill (~/.agents/skills/ralph-loop/SKILL.md, 2.0K on disk, last touched 16 August 2026) is the whole outer loop of Ch. 14 in two kilobytes: it defines the idle check, caps the run at 100 iterations, and makes the agent state a truthful completion promise before it is allowed out. Two kilobytes, one job, one exit condition. There is no honest version of that file that runs thirty pages.
5. **Eval per skill.** Each bundle gets its own pass/fail check (Part VII). The mega-doc has one eval: vibes. Skills have many: verdicts.

What this is not: 400 micro-skills nobody can find, which is the failure mode I sprinted into next, because I am nothing if not thorough about hitting both walls. Discovery is part of the design: an index, a naming convention, a router that knows the catalog. Skills you can't trigger are brain-dump pages with extra steps.

• • •

ONE WORKED EXAMPLE

I have run this split. I did not instrument it, which is the part I want back, so read the next four sentences as my memory and not as a measurement. A deploy runbook. Before: 30 pages in the repo root, loaded every run, holding deploy steps, rollback steps, naming conventions, on-call etiquette and incident templates. The agent routinely skipped the canary check on page 19 while perfectly formatting incident titles from page 27. Attention bingo. After: five skills (deploy-canary, deploy-rollback, commit-format, incident-file, oncall-handoff), each loaded by trigger, each with its own three-line eval.

My recollection is that the canary stopped getting skipped once the canary skill loaded WITH the deploy task and contained nothing else. I can't hand you a percentage, because 2026 me was too busy feeling clever to log one. Note the thing that did not happen: those 30 pages never got

rewritten. They got split and triggered, and every piece came out shorter than the whole used to be. The words weren't the problem. Loading all of them at once was.

• • •

THE QUIET FAILURE THAT PWNED ME

The loud failure is the 200-page doc nobody reads. The quiet failure is subtler:

You split the doc into skills and load them ALL anyway.

Auto-loading the full catalog every run rebuilds the brain dump with extra file I/O. The loading discipline IS the pattern: trigger, relevance, absence by default. A skill library with eager loading is a mega-doc with a directory listing. Guess who did this. I split my 40 pages into a tidy set of skills, felt great about it, and then loaded all of them on every run because what if the agent needed one. Same 40 pages, more files, extra disk reads, and a smug feeling I had not earned. Chuck Norris can load the entire catalog every run and still have attention left over. You cannot. I am not Chuck Norris, I am the n00b who mistook a directory listing for a design.

Second quiet failure: skills that duplicate instead of reference. Five skills each pasting the same auth setup, drifting apart over quarters, until three of them are teaching your agent to log into a service that got renamed last spring. Shared primitives go in shared includes; skills stay deltas. DRY applies to your agent's brain too.

• • •

DO / DON'T

Do

- One job per skill. Split on contradiction, version cadence, or trigger, whichever cracks first.
- Trigger by task, default to unloaded. Absent skills cost nothing.
- Write procedures a stranger could execute. Steps, checks, good-output examples.
- Eval each skill separately. Small bundles, sharp verdicts.

Don't

- Don't duplicate shared setup across skills. Reference, don't paste.
- Don't skip the index. Untriggerable skills are dead pages.

• • •

WHERE THIS SITS IN THE BOOK

Part I closes here: split the work (Ch. 8), dress the steps (Ch. 9), mind the wrapper (Ch. 11), budget the bucket (Ch. 12), and bundle the know-how into loadable skills instead of a monument doc. Part II opens next with the loop itself: Ralph, the outer loop that doesn't dare stop (Ch. 14).

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (skills over mega-docs), kept as practitioner claim, consistent with Ch. 8/12/57.

Verified:

- ralph-loop skill file (idle-check loop, 100-iteration cap, required truthful completion promise), 2.0K, modified 16 August 2026, ~/.agents/skills/ralph-loop/SKILL.md (local, unpublished)
- ClickHouse engineering guide, "Incident response process" (runbook fits one screen; written for a stressed engineer at 3am; tied to one alert; "a runbook whose first three steps are automated is three steps of noise"), accessed 9 September 2026, https://clickhouse.com/resources/engineering/incident-response-process

Open:

- Canary-runbook worked example: printed as Jeremy's recollection, explicitly not a measurement. Nothing public substitutes; the ClickHouse guide names the decay problem but publishes no before/after figures.
- No external papers claimed; mechanism argued from attention-budget + Ch. 12, with the ClickHouse runbook guidance as the ops-world precedent.
- Cross-refs kept honest: Ch. 8 (decompose), Ch. 12 (bucket budget), Ch. 16 (trigger routing), Ch. 20 (stop rules per skill), Ch. 58 (subtract slop), Ch. 60 (contradiction exhibit), Part VII (per-skill evals).

CHAPTER 14

Ralph. Don't You Dare Stop.

"Fanaticism consists in redoubling your effort when you have forgotten your aim."

GEORGE SANTAYANA, THE LIFE OF REASON, INTRODUCTION (1905)

The loop had been awake for hours doing absolutely nothing useful. Every five minutes it woke, checked, found the condition unmet, and fired the agent again. Every run returned `{"error": "missing API key"}` into a log nobody opened. By the time I looked there were 120 wasted runs stacked under each other and about 100 identical drafts of the same chapter on disk, each one billed. The loop was healthy. The agent was healthy. The silence was the problem, and the cause was small enough to be humiliating.

> ***Bottom line:*** *An agent loop that re-wakes itself until a verifiable stop condition wins. Idle != done. The pattern is old (cron, retry loops, polling) and the shape is universal: schedule a re-wake, check if done, re-dispatch if not. Do it right and work finishes without your hand. Do it wrong and you re-build the same task forever.*

• • •

WHEN IT BITES

- You hand an agent a task, it runs once, and you come back to find it halfway done, waiting for input, three chapters in and apparently tired.
- A scheduled job does one small thing, then goes dormant. You need it running until a property verifies: all chapters drafted, all bugs fixed.
- Your agent hits a soft blocker and stops instead of pivoting or escalating.
- The loop runs forever because you never defined a true stop, just a guess about when the job is "done."

• • •

THE PATTERN

Ralph is a re-waking loop with these parts:

1. Schedule a wake. Cron, delay queue, polling interval. Every N minutes, wake and check. The loop never assumes the agent will finish; it assumes the agent does one thing and stops.

2. Check if done. One verifiable criterion: count rows, grep for a marker file, call an API and read the response. Binary. Ground truth, not "the agent said it was done."

3. If not done: re-dispatch. Same prompt, different input, one unit of work per run. The loop controls pacing; the agent stays dumb about when to stop.

4. If done: shut up and go home. Remove the cron, clear the queue, stop polling.

5. Idle != done. An agent that runs silently and returns is not done. An agent that says "I'm done" is ESPECIALLY not done; agents hallucinate completion like candy. I shipped a loop that trusted

`{"status": "done"}` and it congratulated me 60 times in a row for finishing nothing, so take this one from the guy who paid for it.

The nastiest version is when your tests agree with the lie. On August 28, 2026 the airank report generator had 1,592 passing tests behind it and printed a document stamped "CHECKED: Measured" about a page whose fetch had failed, then scored that page 25/100. Six independent reviewers read it and all six said the same thing: this document is lying. The suite was green because it tested that the badge rendered, never that it was true. It caught a CSS bug before it caught a report inventing its own measurements.

The name comes from the /ralph-loop skill. Other projects call it keep-alive, poller, retry daemon.

• • •

ONE WORKED EXAMPLE

Real scenario: this manuscript. Sixty chapters, each on a fixed template. An agent drafts one chapter cleanly and stops, because the agent's job ends at "write the file and return."

Setup: cron every 5 minutes (`*/5 * * * * /usr/local/bin/ralph-loop continue`), state in `/manuscript/.opencode/ralph-loop.local.md` tracking `iteration: 14`, `maxIterations: 60`, `chapters_verified: 14` (files passing a structure check: word count over threshold, all 6 sections), `kill_flag: false`.

The timeline: 1. **Minute 0:** Cron wakes. Iteration 14, verified 14, not done. Fetches the chapter 15 spec, spawns an agent, which writes `/manuscript/15-*.md`. Script runs the structure check; if it passes, `chapters_verified` and `iteration` both go to 15. Commit state, exit. Elapsed: 3 minutes.

2. **Minute 5 through 245:** Repeat, same cycle each time.
3. **Minute 250:** Iteration 60, verified 60. STOP CONDITION MET. Sets `kill_flag: true`, exits.
4. **Forever after:** Cron wakes every 5 minutes, reads the flag, does nothing. A GeoCities page with an under-construction GIF on it: still technically live, costing nobody anything, never changing again.

What this buys you: 60 chapters with zero manual re-invocation. If chapter 37 times out mid-draft, minute 185 re-tries it, because iteration doesn't advance until verification passes. And every run is its own log, so when chapter 43 breaks you read chapter 42's log.

THE DANGEROUS MOMENT: IT WORKED, SO YOU SEE 60 FILES AND ASSUME THE BOOK IS DONE. IT ISN'T. I COUNTED 60 FILES, DECLARED VICTORY, POURED A DRINK, AND THEN OPENED ONE.

THE QUIET FAILURE

The loud failure is obvious: no stop condition at all, loop runs forever, easy to see and kill.

The quiet failure is subtler, more expensive, and how you end up in production at 2am:

Failure A: You define a stop condition that never triggers.

The loop checks for `/work/DONE`, but the agent writes `/work/done.txt`. I wrote both halves of that mismatch in one sitting, then blamed the model for 40 minutes.

On August 8, 2026 nine separate systems on airank announced success in one night and delivered nothing, and two were this exact bug wearing a suit. One was a config key the code read faithfully and nobody had ever defined. Both checks ran, both came back clean, and neither was physically capable of coming back dirty. That same night handed me a 4KB dump of a 900MB database. Every one got caught the same boring way: stop reading the report, measure the effect. Byte counts instead of log lines.

Result: the task finished hours ago. The agent re-runs every 5 minutes anyway, hammering the filesystem, database, and API.

Failure B: The agent stops, but doesn't signal why.

It runs, produces nothing, returns empty. The loop checks the condition, sees nothing (empty table, no marker file, zero count), and wakes again on schedule. Loop alive, work dead, nobody knows.

On August 7, 2026 the airank collector ran 407 browser sessions in 25 minutes and produced zero artifacts and zero database rows. The retry logic counted attempts by reading ledger rows, and a failed session never wrote one, so the counter believed nothing had ever been tried. Every phrase stayed eligible and got picked right back up. Phrase 92 got attempted six times in 25 minutes. I had put the upload and the attempt record inside one conditional, so "we tried" and "we got something back" were the same fact. They are not. Record the attempt, not the artifact.

This is how you burn a pile of money in LLM costs debugging a typo in an environment variable. The loop did its job. The agent did its job, failing loudly into a log I never read. The mistake was mine and one character long.

Failure C: Race condition on the stop check.

You write files to a network mount. The condition checks file existence, but NFS attribute caching can show the loop "file not there" for seconds after the agent finished writing. Loop wakes, re-dispatches, and now two agents are grinding away on one file they both think they own. Caveat, because I would rather be boring than caught: I have no dated incident of my own for this one. It is a known shape, not a story I can put a timestamp on, and I am not going to invent one to make the section symmetric.

The fix for all three:

Pair the stop check with a diagnostic log. Every wake, log what the condition looked for, what it found, why it didn't trigger. One log line inside a shared HTTP client's back-off returned 2,161 hits in nine hours on September 6, 2026, and turned a theory into a number I could grep.

Bonus failure: Silent retries on transient errors.

The agent hits a 429, returns empty. Loop re-wakes, agent re-runs immediately, hits the same 429. By the time the limit expires it has been queued 60 times, and then it succeeds. You see 120 identical runs and spend an hour deciding whether that is an infinite loop or a win. Neither. It is a retry storm.

Mine cost $815.38 a day across 34,083 API calls, and I have four days of invoices proving I did nothing about it. The tell beat the symptom: on August 20, 2026 the queue sat at about 385 jobs while workers chewed 22 a minute against a queue that would not drain. I had already written the guard. Cron called artisan directly instead of the wrapper, so the guard never ran once, and a scheduled job on every web server re-issued the gated work every minute. The pin was written, installed, correct, and completely inert. What caught it was the number: spend stayed at exactly $815.38 after I "fixed" it, and a delta of zero is not a fix, it is a fix that never loaded.

• • •

DO / DON'T

Do

- Define one stop condition. Binary, one line, testable by a grep or a query. "SELECT COUNT(*) FROM chapters WHERE draft_status='VERIFIED' >= 60" is a condition. "All chapters look done to me" is a hope.
- Record the attempt separately from the artifact. If a failed run writes nothing, the retry counter cannot see it, and phrase 92 gets tried six more times.
- Pair the stop check with a diagnostic. Every wake, log what you checked, what you found, why it didn't exit, and the last agent result including errors.
- One unit of work per run, every 5 to 30 minutes depending on how long the work takes.
- Exit cleanly when done. Dead loops are cheap; zombie loops cost money.
- Prove the stop condition fires before it runs 1,000 times. One task, watch the flag flip.
- Set `maxIterations` with teeth: "finishes by 150 or page somebody," not "could finish by 1000 if everything breaks right."
- After deploying a fix that should cost less, check that the number moved. Zero delta means it never loaded.
- Back off on transient errors, and log the back-off. My 5, 30, 120 minute steps are guesses that have not bitten me yet, not a curve I derived. The only back-off I have actually measured was 2/4/8/16/30 seconds buried in a shared client, and it was invisible until it got a log line.

Don't

- Don't rely on the agent saying "I'm done." Agents optimize for sounding confident, not for being right.
- Don't put the stop condition inside the agent's prompt. The agent owns work, not exit logic.
- Don't ship a loop without diagnostics. A loop that wakes every 5 minutes and produces zero logs is a blind spot with a cron entry.
- Don't assume the output format is stable. Validate schema first.
- Don't mix "work done" with "agent stopped." An agent that quit after 3 chapters with no error looks exactly like one that drafted 3 chapters.
- Don't trust a green suite to catch this. 1,592 of them watched a report invent its own measurements.

• • •

WHERE THIS SITS IN THE BOOK

Part I was how to talk to the model: split work (Ch. 8), persona (Ch. 9), wrappers and their lies (Ch. 11), context (Ch. 12), skills (Ch. 13). Part II opens with the outer loop, the ring that keeps silence from looking like success, then works through the machinery inside it: planning (Ch. 15), routing (Ch. 16), tools (Ch. 17), reasoning (Ch. 18), reflection (Ch. 19), goal setting (Ch. 20). Every one assumes the loop is alive and checking. Without a verifiable stop, a loop is just a way to burn money repeatedly.

• • •

SOURCES AND RECEIPTS

Thesis is mine, field-tested on airank, aigate, and the loop that drafted this book.

Verified: - /ralph-loop skill file: `~/.claude/skills/ralph-loop/`, referenced by the pstack and Claude Code hooks on this machine. - Cron polling pattern: standard on Linux, systemd, and Kubernetes. - Hallucinated completion, certified by a green suite: "The honesty layer lied, and 1,592 green tests watched it happen," airank build log, August 28, 2026. `file://~/Projects/airank/blog/2026-08-28-the-honesty-layer-lied.md` – Failure A (a stop check that could never fire): "Everything reported success," airank build log, August 8, 2026. `file://~/Projects/airank/blog/2026-08-08-everything-reported-success.md`. Nine systems, one night. - Failure B (silent agent errors, empty results): "Four hundred and seven sessions, no evidence," airank build log, August 7, 2026. `file://~/Projects/airank/blog/2026-08-07-four-hundred-and-seven-sessions-no-evidence.md` – Rate-limit and re-issue storms: "Nothing called it," airank build log, August 20, 2026. `file://~/Projects/airank/blog/2026-08-20-nothing-called-it.md` – Silent back-off inside a shared HTTP client: `~/.claude/skills/shared-rate-limit-silent-backoff/SKILL.md`, verified September 6, 2026. A loop gapped 2-5 minutes five times an hour with nothing in its own log; the cause was an unlogged 2/4/8/16/30 second back-off, and the log line that exposed it hit 2,161 in nine hours.

Honest gaps: - Failure C (NFS race on file checks): illustrative only, and labeled that way in the text. No dated incident of mine exists; nothing in the airank build logs or fleet notes covers it. - No academic papers cited; thesis is field-tested, not peer-reviewed. - Ralph name: coined here, not industry-standard. Industry says "retry loop" or "outer loop." - Backoff timing (5min, 30min, 120min): heuristics, not derived, and possibly wrong for your traffic shape. The mechanism is documented practice (on MariaDB SQLSTATE 1129: `mysqladmin flush-hosts` once, then reconnect with backoff, never retry into the wall), and one real curve is written down (2/4/8/16/30 seconds, verified September 6, 2026). What I do not have is a per-project curve where each step is justified by a measured failure rate or a spend number, so treat my minutes as a starting guess. - "Never trust agent output" is strong: six of six reviewers caught one generated report claiming measurements it never took. That is a hard failure, not a base rate.

Cross-references kept honest: - Ch. 8 (decompose work into units): why one unit per run - Ch. 15 (planning is a bet): the loop re-dispatches the same bet until it lands - Ch. 20 (goal setting): the

stop condition IS the goal, made verifiable - Ch. 35 (ledger over diary): log state, not full agent output - Ch. 60 (contradiction): "agent is done" contradicts "loop is awake"; loop wins

"A plan is a bet: a claim about which thing is load-bearing, priced in confidence and paid for in reality."

CHAPTER 15

A Plan Is Not the Work

"Plans are worthless, but planning is everything."

DWIGHT D. EISENHOWER, REMARKS AT THE NATIONAL DEFENSE EXECUTIVE RESERVE CONFERENCE (1957)

The collector ran 407 browser sessions in 25 minutes and produced nothing. No data, no artifacts, no ledger rows, no record that any of it had happened at all. I spent hours generating excellent theories: browser profiles colliding, OpenAI rate-limiting, missing proxy credentials. Every one of them was plausible. Every one of them was unprovable, because the thing that would have settled it was never written down. I was wrong three times in a row before I found out what actually happened, and the answer was not a distributed-systems problem.

> ***Bottom line:*** *A plan is a bet: a claim about which thing is load-bearing, priced in confidence and paid for in reality. Writing it down does not execute it, and formatting it with P0/P1/P2 and a dependency graph does not execute it either.*

• • •

WHEN IT BITES

- Your agent returns twelve numbered steps. You approve it. It runs step 1, skips to step 4, invents step 5, and reports success.
- You inherit a handoff and start executing its top priority. The paths in it are still correct. The conclusions are not.
- The agent replans, then replans the replan: twenty minutes of reasoned markdown, zero commits, and a token bill to explain.
- The opposite: step 3 blew up and it ran step 7 anyway, because the plan said so. Leeroy Jenkins (2005) at least yelled his own name before he charged in and wiped the raid. Your agent charges in silently and files it as on-plan.

• • •

THE PATTERN

Planning is not the problem; the literature says it's good.

ReAct interleaves reasoning traces with actions so the two feed each other (Yao et al., arXiv:2210.03629). Plan-and-Solve goes the other way: split the task into subtasks, then carry them out, to kill missing-step errors (arXiv:2305.04091). Plan-then-execute buys predictability, cost-efficiency, and an underrated bonus: fixed before the tool output arrives, it's inherently more resistant to indirect prompt injection (arXiv:2509.08646).

Everything I have watched in production says two things about how humans read plans. People mistrust plans easily. Worse, they trust plausible-seeming ones just as easily, and the better a plan

reads, the less anyone checks the number underneath it. That second one is my own observation from watching review calls, not a study I can hand you.

My own experience, unmeasured, is that agents **frequently fail to follow the plan they were instructed to follow**, falling back on whatever workflow they have seen most. The plan is still sitting there in the transcript looking like what happened.

Why do agents drift harder than people do? A person who gets to step 3 and finds the ground missing feels something, and stops. Agents drift because they were never on your plan in the first place; they were on the average of everyone else's.

That's the divergence. Not a hallucination. Not a tool failure. The plan and the execution are two different objects, and only one of them gets read later.

Outcome School (May 2026) names three failure modes, and I have hit all three:

1. **Infinite re-planning.** The agent loops forever rewriting the plan without making progress. Replanning *feels* like work: it produces text and costs money.
2. **No re-planning.** The agent executes blindly despite failures, because the plan is the ground truth and reality is a rounding error.
3. **The plan becomes outdated.** It was true when written. It isn't now. Nothing in the system knows that.

When that first mode has no bound, you're in the class documented by *When Agents Do Not Stop*: 68 Infinite Agentic Loop failures across 47 GitHub projects, found by static analysis (arXiv:2607.01641). Ch. 14 is the loop that keeps going on purpose; this one keeps going because nobody defined finished.

The fix is cheap and nobody does it:

Before acting on a priority, re-measure the number the priority rests on.

That's it. The cheapest verification outranks the most confident theory.

• • •

ONE WORKED EXAMPLE

airank, 9 August 2026. Six engineers convened for the most boring possible session: a plan that does not spend money and does not ship rankings. Seven issues, ordered by severity.

The output was a real plan: P0/P1/P2, strict dependencies, a named test per item. Well-formatted. Persuasive.

And it was masking a bug.

One issue was marked *already fixed in code*. Reading the diff: commit `33a9188` archives successes into `chatgpt-observations/api/...` and failures into `chatgpt-skipped/api/...`, each with a `minio_key` pointing at the answer and its cost. Three tests, green in 0.38 seconds. Fixed.

The class docblock, line 37 of that same file, still claimed "*written to chatgpt_captures with its raw text … recoverable later by re-running extraction.*" That sentence had been false for months: `capture()` took the text and threw it away. It wasn't documentation, it was the bug's alibi, and I wrote both. The failures that never archived are still gone: I remember it as a five-figure count of them, and at the per-call price I read off the invoice that put the loss in the low hundreds of dollars, not the $738 I typed into the P0. I never saved the query, so treat those figures as a war story, not a benchmark. Twenty-five years shipping software and I can't do one line of arithmetic before promoting it to P0. Total n00b move.

Code-fixed. Production-unverified. Those are not the same word.

So P0 was not more code. P0 was **two read-only SQL queries**: count conditions and non-null `minio_keys` after the deploy timestamp, on both tables, using `UTC_TIMESTAMP()`. Expect 100%. If not 100%, P0 isn't done.

Three shorter variants of the same shape:

7 August 2026, the inherited handoff. It said the bottleneck was the brand-recognition gate, said it twice in two sections so it read as corroborated, and one `GROUP BY` over the day's captures put `navigation_failed` at the overwhelming majority and the brand gate at a rounding error. I logged those percentages in the write-up that night; I no longer have the query output, so read them as my notes, not as a measurement you can rerun. It also said seventeen branches needed merging; all seventeen had already landed by other routes, and merging them would have reintro-

duced a heuristic main's own comment says was tried and reverted. Every claim in it was true when written. Facts survive the trip; conclusions don't.

Same week, the manifest. Committed `stack.yml` said 0 replicas and `AIR_SHARD_TOTAL=1`; the cluster running that morning was **20 replicas at** `AIR_SHARD_TOTAL=20`, which I read off `docker service ls` at the time and did not save, a state that existed only in Swarm's memory on a Raspberry Pi. All your base are belong to a $70 computer with no backup of its own opinions. `docker stack deploy` from that repo doesn't error; it reports success and takes prod to zero.

And the 407 silent sessions from the top of this chapter. The answer was `402 Payment Required`, a billing problem in a distributed-systems costume, invisible because a `continue` I wrote and defended skipped the ledger row along with the empty artifact. I spent a day debugging my own good judgment; the fix was a credit card.

One more, at lower stakes: in commander-in-chief, `test_display_integer_stretch_configured` was supposed to prove `project.godot` pins the stretch aspect, but it asserted the engine's default instead, so it stayed green after I deleted the line. Commit `2722533` made the test read the file. A test that checks the safety net instead of the contract is a plan wearing a green checkmark.

• • •

THE QUIET FAILURE

The loud failure: the agent that plans and stops, loud because nothing happened.

The quiet failure:

The agent plans, executes something else, and the plan is what gets read at review.

The plan is a clean artifact: markdown, committed, surviving. The trajectory is 40,000 tokens of tool calls nobody will scroll. So the plan becomes the record of what happened, and it is not. This is the seam Ch. 20 is built on: if "done" isn't measurable, the most articulate artifact in the room wins by default.

• • •

DO / DON'T

Do

- Price the bet: name the number it rests on, and re-measure before you act.
- Make the plan carry its own tests. Not "fix archival," but "expect 100% non-null `minio_key` after deploy_ts, else P0 is not done."
- Mark facts and conclusions differently in every handoff. Paths are facts. Bottlenecks are conclusions.
- Re-inject the plan during execution: at each subtask boundary and after any failed tool call. That cadence is my working practice, not a measured result.
- At review, put the trajectory next to the plan: the commits, the tool calls, the queries actually run. If only one of the two is in the room, the plan wins and you learn nothing.
- Bound the loop (Ch. 14); define done as a measurement (Ch. 20).
- Treat "fixed in code" and "verified in production" as different plan statuses.

Don't

- Don't approve a plan for being well-formatted. Format is confidence, not evidence.
- Don't count a claim as corroborated because it appears twice.
- Don't count planning tokens as progress; a 4,000-token plan isn't a deliverable.
- Don't let failures exit without a record.
- Don't deploy from a manifest you haven't checked against production.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 built the ladder: PRD to milestones to tasks to build. This chapter is the tax on that ladder: every rung is a bet, and bets decay. Ch. 14 (the Ralph loop) keeps execution moving after the plan is written; this chapter says that loop needs a measurement, not a checklist. Ch. 16 is next: the agent that picked the wrong lane entirely. Ch. 20 is where it lands: if you can't measure done, the plan will keep telling you that you are.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("the plan is not the work; the work is"), argument, not citation.

Verified. The arXiv and Outcome School entries are public and checkable. The worked examples are first-party: private repos, blog files, one Pi's runtime state, checkable by me at the paths below, not by you.

- ReAct: reasoning + acting interleaved. Yao et al., arXiv, March 2023: https://arxiv.org/abs/2210.03629
- Plan-and-Solve prompting: split into subtasks, execute; targets missing-step errors. arXiv (ACL 2023), May 2023: https://arxiv.org/abs/2305.04091
- Architecting Resilient LLM Agents: plan-then-execute gives predictability, cost-efficiency, and resistance to indirect prompt injection. arXiv, September 2025: https://arxiv.org/abs/2509.08646
- When Agents Do Not Stop: 68 Infinite Agentic Loop failures across 47 GitHub projects, found via static analysis. arXiv, July 2026: https://arxiv.org/abs/2607.01641
- Failure taxonomy: Infinite Re-Planning, No Re-Planning, Plan Becomes Outdated. Outcome School, May 2026: https://outcomeschool.com/blog/plan-and-execute-agent
- Worked example 1: the plan that did not spend, airank, 9 August 2026. Commit `33a9188d40cd8e13580fad80b0a21ffc3ccdd7ef` (archival into `chatgpt-skipped/api/` with `minio_key`; the false docblock at line 37), repo `~/Projects/airank/.git`, write-up `~/Projects/airank/blog/2026-08-09-the-plan-that-did-not-spend.md`. Private repo, so the hash is mine to run, not yours.
- Variant: seventeen branches, zero merges, airank, 7 August 2026. `navigation_failed` dominating the failure mix against a negligible brand gate, per my notes that day; manifest 0 replicas / shard 1 vs production 20 / 20 on image `cc4c39a` while the repo had bumped to `9c1a21f`. Recorded in `~/Projects/airank/blog/2026-08-07-seventeen-branches-zero-merges.md`, 7 August 2026.
- Variant: 407 sessions, no evidence, airank, 7 August 2026. `continue` skipping artifact and ledger row; three wrong theories; `402 Payment Required`. `~/Projects/airank/blog/2026-08-07-four-hundred-and-seven-sessions-no-evidence.md`
- Variant: the test that checked the engine default. Commit `2722533efb1d67d7040c-c6a65c5d766d9b5581bc` (adds a `FileAccess.get_file_as_string` check on `project.godot`), shoe-money/commander-in-chief, 3 August 2026, repo `~/Projects/commander-in-chief/.git`, mirrored on my self-hosted Forgejo at `git.shoemoney.ai`, LAN-only.

What I could not verify:

- I could not find a public case of a shipped vendor or SaaS agent product that planned and then failed to execute. The bug class is documented in the papers above; a named product incident, as far as I can tell, is not.
- No documented case of an agent halting cleanly *after* producing a plan: the loop papers cover agents that don't stop, not agents that stop early. The "plans and stops" failure here is field-observed.

CHAPTER 16

Your Agent Picked the Wrong Lane

"Explanations exist; they have existed for all time; there is always a well-known solution to every human problem: neat, plausible, and wrong."

H. L. MENCKEN, PREJUDICES: SECOND SERIES, THE DIVINE AFFLATUS (1920)

August 7, 2026. The airank collector had been dead for five hours and I had been wrong three separate times before breakfast, each time with total confidence. The third time I was confident enough to build and deploy an entire proxy layer. It worked beautifully. It was clean, it was reusable, it went to production, and it had nothing whatsoever to do with why the collector was down. The actual cause was four lines. Not four hundred. Four. And every green dashboard in the building agreed with me the whole time I was wrong.

> ***Bottom line:*** *Before an agent does anything, something decided what kind of thing this is. That decision is the router. Get it wrong and you don't get a crash: you get flawless work on a job nobody asked for. Good work on the wrong job is just waste with momentum. Classify first, dispatch second, and log the classification so you can find out later which lane it took and why.*

• • •

WHEN IT BITES

- A question that needed a policy lookup gets classified as "general chat." Your agent writes four beautiful paragraphs from memory and is wrong in a way that reads as confident.
- Every request goes to the big expensive model because nobody wanted to be the person who under-provisioned. Your bill is a monument to indecision.
- Or the reverse: everything goes to the cheap tier because a dashboard said latency dropped 40%. The hard 3% now get fast, cheap, wrong answers, and nobody notices because "wrong" doesn't page anybody.
- Your agent has twelve tools and picks the plausible one instead of the correct one. Cache when it needed the source of truth.
- The classifier was written in April against April's traffic. It's September. Nobody re-checked it. It's still 100% confident. I have shipped this one. Twice.

• • •

THE PATTERN

Routing is two jobs that people mash into one then wonder why it's hard.

Job one: classify. What kind of request is this? Not "how do I answer it," but what *category* of problem is it. Lookup versus reasoning. Read versus write. Reversible versus not.

Job two: dispatch. Given that class, which model, which agent, which tool, which prompt, which budget.

Job two is a lookup table, and it's where everybody spends their time because it's the fun part. Job one is where the failures live and it gets `if "refund" in text` and a shrug. I have written that exact line. In production. With a comment above it that said `// temporary`.

Here's the asymmetry that makes this dangerous: **a misclassified request does not fail.** It succeeds. Every metric on your dashboard stays green because every component did exactly what it was built to do. The only thing that's wrong is the premise, and nothing downstream is in a position to check the premise.

The dispatch half is settled enough to be boring. Anthropic's guidance says it plainly: easy and common questions to a cheap model like Haiku, hard and unusual ones to something more capable ("Building Effective Agents," December 2024). LMSYS's RouteLLM (July 2024, arXiv:2406.18665) built learned routers picking between a strong expensive model and a weak cheap one. All true. None of it tells you what happens when the classifier itself is wrong, and that's the part that costs money.

The one rule I'd tattoo on the router: **a wrong cheap answer costs more than an unnecessary expensive one.** When your classifier is unsure, it should not split the difference: it should escalate. Uncertainty is a routing signal, not noise to be thresholded away.

In my own stack, nothing picks the lane except the model at the top of the run. Right now that is Claude Fable 5.1, and it routes by task type off the table in my global config: research to Haiku, planning to Fable, coding to Sonnet, everything else to Opus. No static cost rule underneath it, no threshold. A routing table, and a model reading it.

Which means the most expensive model in my stack is the one deciding who is cheap enough. I pay Fable rates for the privilege of being told that a grep belongs on Haiku.

What I *think* the table sorts on is whether the work is cheap to throw away. Bad research costs me a re-run. Bad judgment ships. That is a belief about my own stack, not a measurement, and I have never gone back through the logs to check whether the split behaves the way I describe it out loud.

Third piece: **the classifier can be a small, cheap, dumb thing, and usually should be.** The story that gets passed around is a team running a product-name classifier on GPT-4 for months before someone tested zero-shot GPT-3.5 and got identical results (Finout, August 2025). No primary source, no postmortem, so I hold it as a parable. I believe it because I have been that team: professionally confident in expensive while free did the same job, because nobody queried the premise.

• • •

ONE WORKED EXAMPLE

August 7, 2026. Collector service dead for five hours, and the diagnosis routed wrong three times before breakfast.

From the airank internal logs, and the cleanest misrouting story I have, because the misrouting wasn't in a model: it was in me and in the error handler, the same failure at two altitudes.

The collector was down. First diagnosis: our IP got banned. Evidence: `curl` returned 403. Confident. Wrong: a cheap test killed it. Second diagnosis: the restart cap is too aggressive, we're throttling ourselves. Confident. Wrong: killed by one measurement. Third diagnosis: it's network-path, a proxy will fix it. Confident enough that we *built and deployed the proxy infrastructure*. Good engineering, deployed cleanly, useful later, and completely beside the point.

The actual root cause was a routing bug in the error handler. A **timeout** was being classified as a **fatal** error instead of a transient one. Fatal went down the give-up path, transient down the retry-and-skip path. The classifier put a recoverable condition in the unrecoverable lane. It gave up, exactly as designed, five hours in a row.

And the reason the three diagnoses were all wrong: we were classifying the symptom from the wrong vantage point. The failure wasn't page load, it was a **login wall**, and the only thing that revealed it was a smoke test run *on the failing node itself*.

Outcome: timeout and network errors got routed to the skip path, the navigation error guard got widened, the proxy stayed (reusable, not the fix), and, the bonus scar, we found a backup job that had been silently failing on a permission gate the whole time. Somebody set up us the bomb months ago and no alert ever said so.

• • •

THE QUIET FAILURE

The loud failure is a router that throws: unknown category, no handler, stack trace, page. That's a good day.

The quiet failure:

Your router has a default lane, and the default lane always works.

Every classifier has a fallback. `else: general_handler`. And the general handler is competent: it's usually the big model with a generic prompt, and it produces a reasonable-looking answer to almost anything. So requests that should have gone to the policy lookup or the database tool quietly land in `else` and come back with something plausible. Not an error. Not empty. Plausible.

Nobody instruments the default lane. Go look at yours right now and count what fraction of traffic lands there. If it's above single digits, your classifier isn't classifying, it's shrugging in production, professionally.

I deliver that line with real conviction and zero numbers of my own, because I have never once graphed the `else` rate on airank or aigate. Strong opinions about your instrumentation, no chart about mine.

Illustration, not a measurement; the arithmetic is the only real thing in it. A router doing 10,000 requests a day, four named lanes and a default. Say 6% lands in `else`. That reads like nothing. It is 600 requests a day nobody classified, 18,000 a month, each answered plausibly by the big model with a generic prompt, none counted as a miss. If one in fifty of those needed a lookup the general handler cannot do, that is twelve confidently wrong answers a day that no alert has ever mentioned. Meanwhile your classifier's accuracy metric, computed only over the requests it classified, still reads 99%. The default lane is not in the denominator. That is the whole trick.

Second quiet failure: **router drift.** You tuned the classifier on the traffic you had. Traffic changes. Users learn new phrasings, you ship a feature with a whole new request shape. The router is last quarter's world applied to this quarter's, at full confidence. There is no published longitudinal study on how fast a learned router decays (I looked), so this is a field observation. Every router I've run has drifted, and none of them announced it.

Third: **you route on surface features instead of intent.** Keyword matching is routing on vocabulary, and vocabulary is the least stable thing about a request. Anthropic's own ticket-routing guide leans on semantic understanding for exactly this reason. The word "refund" appears in requests about refunds and in requests about why the refund *policy page* 404s. One goes to billing. One goes to engineering. Same token.

• • •

DO / DON'T

Do

- Make classification a separate, named, logged step. If you can't answer "what class did the router assign this request, and how sure was it," you don't have a router, you have vibes with a switch statement.
- Emit confidence, and route on it. Low confidence escalates, and a downstream agent in the wrong lane can kick the request back with a hint instead of doing its best with the wrong job.
- Instrument the default lane like it's a bug counter.
- Use a cheap model for the classifier, then *measure* whether cheap is enough.
- Re-benchmark the router on recent traffic: sample the last 30 days, hand-label a couple hundred, check agreement. I used to say quarterly, which I picked because it *sounds* responsible, the exact species of reasoning this chapter is against. What I run is constant, triggered by change, not by the calendar. I clear context a lot and I add skills and MCP servers a lot, so every time the harness moves I go looking for the regression. A schedule measures the calendar. Changing the harness is the thing that breaks the router.

Don't

- Don't route on keywords when you can route on intent.

- Don't optimize the router on cost alone. Cost is trivially measurable and correctness isn't, so cost wins every argument unless you deliberately weight against it.
- Don't trust a green indicator from one layer as evidence about another. A handshake is not a route.
- Don't let the router be the one component with no evals. It's the component whose errors are hardest to see, which means it needs the most instrumentation, not the least.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 15 was plan versus work. This chapter is the decision immediately before that: *what kind of thing is this even*, which determines who gets to plan it. Ch. 17 goes down a level: once you're in the right lane, picking the right tool inside it is the same failure at smaller scale, with the same silence.

Ch. 28 is the payoff: Haiku researches, Sonnet codes, Opus judges. That scheme only works if the classification in front of it is right. Ch. 7 (the tiny-model moat) is the economic argument for wanting a router at all: a small model taking 97% of traffic with clean escalation on the rest is only viable if the escalation decision is trustworthy.

And Part VII, evals: the router is the component people forget to evaluate, and it's the one where being wrong is invisible.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (classification is the failure surface, dispatch is the easy part; misroutes succeed loudly at the wrong job): argument, not citation.

Verified:

- RouteLLM, LMSYS Official Blog, July 2024: https://www.lmsys.org/blog/2024-07-01-routellm/: "In our routing setup, we focus on the case where there are two models: a stronger, more expensive model, and a weaker but cheaper model."
- "RouteLLM: Learning to Route LLMs with Preference Data," arXiv, June 2024: https://arxiv.org/abs/2406.18665: learned routers that dynamically select between a stronger and weaker LLM at inference.
- "Building Effective Agents," Anthropic Engineering, December 2024: https://www.anthropic.com/engineering/building-effective-agents: routing easy/common questions to cheaper models and hard/unusual ones to more capable models.
- Ticket routing guide, Anthropic Claude Platform Docs, 2024: https://platform.claude.com/docs/en/about-claude/use-case-guides/ticket-routing: classification by intent using semantic understanding.
- airank internal blog, August 7, 2026: `~/Projects/airank/blog/2026-08-07-three-wrong-diagnoses-before-breakfast.md`, five-hour collector outage; three wrong diagnoses; root cause was timeout classified as fatal instead of transient.
- Jeremy's global agent config, September 2026: orchestrator is Claude Fable 5.1, routing by task type (research to Haiku, planning to Fable, coding to Sonnet, everything else to Opus). Re-benchmarking is on change, not on a calendar.

Cited but unverified (marked in text or softened):

- GPT-4 product-name classifier vs. zero-shot GPT-3.5: Finout, August 2025, https://www.finout.io/blog/openai-cost-optimization-a-practical-guide. Secondary source, no primary team or numbers.
- "A wrong cheap answer costs more than an unnecessary expensive one": appears in Medium, July 2026 (https://medium.com/@adnanmasood/right-sizing-the-frontier-…) attributed to case studies including an OpenAI GPT-5 router rollback. **The rollback itself is unverified**: no primary source found.

- RouteLLM's "95% of top-model quality at 85% lower cost": repeated widely (incl. Taskade wiki, 2026) but not confirmed against the paper's own tables. **Deliberately not printed as a number in this chapter.**

What I could not verify:

- No public number exists for what a model-tier misroute costs: no vendor postmortem, no incident report, nothing. Searched, came up empty. The nearest public figure is RouteLLM's cost reduction, which measures routing done *right*, the opposite claim. Both airank stories here are error-classification misroutes, not model-tier ones, and the chapter does not pretend otherwise.
- No published misclassification taxonomy: which *classes* of request reliably confuse routers, and no longitudinal study of router drift. The drift claim is a field observation, stated as such in the text.

"If the tool can do it, the model does not get to remember that it did."

CHAPTER 17

Give It Hands or It's Just a Liar

"Talk is cheap. Show me the code."

LINUS TORVALDS, LINUX KERNEL MAILING LIST (2000)

At 18:43 on August 8, 2026, my collector went quiet and the log told me `No phrases matched.` I believed it for about four minutes, because it was true. Then I ran the counts: phrases 0, observations 0, captures 0, products 0. The newest dump was from 03:15 that morning, `log_bin` was `OFF`, and I had no idea what statement had done it. So I did what any responsible engineer does. I picked a suspect, wrote the accusation down with confidence, and was completely wrong about it inside the hour. Eight more things would report success to me that night.

> ***Bottom line:*** *If the tool can do it, the model does not get to remember that it did. A model's account of its own tool use is a story it generated. The tool's log is the only thing that happened. Build the loop so every claimed action has a receipt from outside the model (a diff, an exit code, a row count, a byte count) and treat any claim without one as unverified. "I have updated the file" is not evidence a file was updated. It is evidence the model knows what a person says after updating a file.*

• • •

WHEN IT BITES

- An agent reports "I've updated `config/billing.php`" and the file's mtime hasn't moved since Tuesday.
- A run finishes green, the summary says all seven fixes landed, `git log` shows four commits.
- The model narrates a tool call it never emitted and the loop continues on a fabricated result.
- The tool *did* fire, returned an error, and the model summarized it as success.

The expensive version is the model being wrong *confidently, in the past tense.* Past tense is what makes you stop checking.

• • •

THE PATTERN

Tool use looked like it solved hallucination. It didn't. It moved it.

Toolformer (Schick et al., February 2023) showed a model could learn, self-supervised, which APIs to call and with what arguments; OpenAI shipped function calling four months later, Anthropic's went GA on Vertex in June 2024, and by November 2025 Anthropic was shipping tool *search* and *generation*.

Giving a model hands does not give it a memory of using them. It has no privileged access to which strings in its context came back from a real execution and which it wrote two hundred tokens ago. **A tool result and a hallucinated tool result are indistinguishable from inside the model**, and distinguishable only outside it, at the layer that invoked the tool and holds the log.

The research caught up. “How Enhancing LLM Reasoning Amplifies Tool Hallucination” (arXiv, October 27, 2025) found reasoning RL *amplifies* it: you taught the model to produce a confident chain ending in an answer, and “I called the API and it returned 200” is an excellent-scoring link in one. AgentHallu (arXiv, January 11, 2026) then measured how well models *find* the step where the lie happened. Tool use was the hardest category: **11.6% step localization**, best model overall **41.1%.**

The taxonomy hardened that same window (Emergent Mind, January 18, 2026): tool-selection, tool-usage, solvability, tool-induced myopia, and bypass. “Bypass” should keep you up: the model skips the tool and answers from the weights, formatted exactly as if the tool had run.

The posture follows mechanically:

1. **The model proposes, the harness disposes.** The model emits an intent; something outside it executes and records.
2. **The record is the truth.** Exit code, bytes written, rows affected, commit SHA, HTTP status. Not prose.
3. **Nothing downstream reads the model’s summary.** Downstream steps read the record.
4. **Absence of a record means it didn’t happen.** Not “probably happened, it seemed sure.”

Rule 4 costs one check. Believing its opposite cost me a night.

• • •

ONE WORKED EXAMPLE

August 8, 2026. 18:43, `No phrases matched.` I can quote both because they are the first two entries in the incident log I kept that night, which turned out to be the only thing in the building that reported honestly.

Two tool calls into reading the sharding code (I’d just moved shard count from 1 to 2, because two is obviously twice as good as one) I ran the counts.

```
phrases        0     (was 1,463)
observations   0     (was 750)
captures       0     (was 2,228)
products       0     (was 5,758)
```

Those parenthesized numbers are the last recorded row counts, not my memory of them; I trust my memory of a number about as far as I trust an agent’s summary of a command.

The database was empty. `No phrases matched` was true, precise, and about something else. The first liar of the night never technically lied.

Newest dump was 03:15 that morning, on a pipeline whose telemetry puts it at fifty observations an hour: fifteen hours gone, roughly 576 observations. `log_bin` was `OFF`, so nothing to replay forward and no record of what did the damage.

Then I got the cause wrong. The migrations table had collapsed into one batch of 52, the shape of a from-scratch rebuild, and a front-end agent had been scaffolding Jetstream in that window, so I said it ran `migrate:fresh`, with all the earned authority of Winamp telling you it really whips the llama’s ass. Then I searched the transcripts: the only two `migrate:fresh` runs were dated **2026-08-06**, and we’d collected 750 observations *after* them. The only hallucinating model in the room was me.

Nine things that night announced success and delivered nothing. The two that cost most:

- `mysqldump --source-data=2` producing **4 KB for a 900 MB database** (MySQL syntax, MariaDB server), stderr piped to `/dev/null` months earlier for being noisy. MariaDB wants `--master-data`, said so exactly once, into the hole I had personally dug for it. Next to it in `ls -l` sat the real baseline at 97 MB: I had been backing up 0.004% of my database and filing it as a backup.
- A shipper reporting `shipped 6 binlog file(s). binlog.000002` was 1,631 bytes at source, **380 on the NAS.** `rsync --ignore-existing` copied a still-open log once and never corrected it. A truncated binlog looks present until you need it.

Every one was caught by checking the effect, not the report.

Outcome: the recovery run walked 685 archived artifacts, skipped 82 already present, read each remaining capture's raw HTML for the question typed, and matched **577** back to `phrases`, against the 576 the collection rate predicted. Two independent methods landing one apart is why I let it touch production. The 26 it could not match were listed individually with reasons. Rows after: 873, later 898. Binary logging is on.

Five days later, August 13, 2026, the same shape ran in reverse: three false *negatives* in four hours. Eleven agents in eleven worktrees shared one MinIO bucket and one database, so the "clean baseline" a reviewer tested was dirty the same way the branches were. The tell: assertion values *mutated*, `0 is identical to 1` one run, `2 is identical to 1` the next. Zero means a sibling ate your fixture; two means a sibling left residue.

It died to the move that kills all of these: **re-run the smallest thing that would fail if the claim were true, and count.**

I said checks are cheap for a year without pricing one. Here is the price.

Call a verification step one small-model call: 2,000 tokens of context and the real command output in, 200 tokens of match or no-match out. Anthropic's pricing page on September 9, 2026 lists Claude Haiku 4.5 at $1 per million input tokens and $5 per million output. That check costs **$0.003.** Three tenths of a cent. Estimate, but the arithmetic is the vendor's. A tool-call round trip, one extra request of about 1,000 new tokens, is **$0.002** at Claude Sonnet 5's listed $2 per million input, **$0.005** at Claude Opus 5's listed $5. Estimates again, and high: cache hits bill at a tenth of base input and my hit rate is 97.5%.

My own meter agrees. aigate billed $27,291.83 across 283,164 requests in the thirty days ending September 8, 2026: **$0.096 a request.** One verify is roughly 3% of one average agent action. A verification step costs less than a postage stamp. It costs less than a hundredth of one.

The skipped one cost a Saturday. A comment in `capture()` said raw answer text was archived; the function threw it away. That sentence is why I believed archival worked, told the owner it was done, and never ran the read-only query that would have said otherwise. **18,567 answers, about $240, permanently.** That $240 buys **80,000** of the checks I just priced.

One thing left I cannot price. Every incident above is a misreported *result*, not an invented call. I am certain I once watched a model narrate a tool call the harness never emitted, but I have no log for it, no date, and no model name, so by my own rule it did not happen. That is my memory, not a receipt, and this chapter runs on receipts. It stays a dinner story.

• • •

THE QUIET FAILURE

The loud failure is "I've updated the file" with no write. You catch that the first time it burns you. The quiet failure is worse:

The tool ran, returned something bad, and the model summarized it as fine.

`shipped 6 binlog file(s)` was true, produced by real code, and one of those six arrived truncated. The report wasn't false, just insufficiently specific in the direction that makes you stop looking. "Did the tool run" is the wrong question. The question is **"what changed, and does it match what was supposed to change."**

Second: **you build the verifier out of the same material as the liar.** A reviewer agent reading a transcript is a language model judging text, at 11.6%. The check has to be a *different kind of thing*: a byte count, a SHA, a query, a compile.

Third: **your success signal is a proxy.** `Up 6 seconds (healthy)` checked the wrong property.

• • •

DO / DON'T

Do

- Make the harness the witness: the layer that executes records what happened.
- Gate on effects: mtime, exit code, commit count, row count, byte count, HTTP status.
- Verify the premise first. A survey is a snapshot of beliefs, and beliefs go stale in hours.

- Watch for mutating failure values: a real bug fails the same way every time, a drifting value is contamination.
- List failures individually with reasons, and prove idempotency by running it twice.

Don't

- Don't accept past tense as evidence. "I have updated," "I ran," "I verified" tell you nothing until something outside the model agrees.
- Don't pipe stderr to `/dev/null`. The 4 KB dump was a solved mystery I had personally gagged.
- Don't trust status docs over live measurement. A banner said backups were broken; the backup it slandered saved the project.
- Don't gate on an exit code without knowing whose it is, or confuse a health check with a correctness check.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 16 got the model to do the thing; this chapter is about not believing it when it says it did. Ch. 24 is MCP, which raises the stakes because the tool and its log now live on someone else's machine. Ch. 33 is *said success, did the wrong thing*. Ch. 36 is the discipline: don't say done until you've checked, and don't let the checker be the doer.

One thing to take: the model's memory of using a tool is worth zero. The tool's log is worth everything.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's: model self-report is worth zero; the tool's log is the artifact.
Verified:

- Toolformer. arXiv, February 2023. https://arxiv.org/abs/2302.04761
- OpenAI function calling launch. OpenAI, June 13, 2023. https://openai.com/index/function-calling-and-other-api-updates/
- Claude 3 Opus tool use GA on Vertex AI. Google Cloud Blog, June 1, 2024. https://cloud.google.com/blog/products/ai-machine-learning/anthropics-claude-3-opus-and-tool-use-are-generally-available-on-vertex-ai
- "How Enhancing LLM Reasoning Amplifies Tool Hallucination." arXiv, October 27, 2025. https://arxiv.org/html/2510.22977v1
- Anthropic advanced tool use. Anthropic, November 24, 2025. https://www.anthropic.com/engineering/advanced-tool-use
- AgentHallu, 11.6% step localization, best model 41.1%. arXiv, January 11, 2026. https://arxiv.org/html/2601.06818v1
- Tool-Use Hallucination taxonomy. Emergent Mind, January 18, 2026. https://www.emergentmind.com/topics/tool-use-hallucinations
- "Everything Reported Success" (pre-wipe counts 1,463 / 750 / 2,228 / 5,758; ~50 observations/hour; 4 KB dump vs. 97 MB baseline; `binlog.000002` 1,631 bytes at source vs. 380 on the NAS; 685 scanned, 82 skipped, 577 recovered, 26 unmatchable, 873 then 898 rows). airank production blog, August 8, 2026. `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- Anthropic list prices (Claude Haiku 4.5 $1 / $5 per MTok, Claude Sonnet 5 $2 / $10, Claude Opus 5 $5 / $25, cache hits at 0.1x base input). Read September 9, 2026. https://platform.claude.com/docs/en/about-claude/pricing
- OpenAI list prices (gpt-5.6-luna $0.20 / $1.20 per MTok, gpt-6-astra $10 / $50). Read September 9, 2026. https://developers.openai.com/api/docs/pricing

- aigate Usage and Analytics, 30 days ending 8 September 2026: $27,291.83, 283,164 requests, 97.5% cache hit. The $0.096 average and the $0.003 / $0.002 / $0.005 step costs are **estimates** from the list prices above.
- "The Cheapest Verification Outranks the Best Theory" (the `capture()` comment describing archival the code never did; 18,567 answers, about $240, permanently). airank production blog, August 9, 2026. `~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md`
- "Three Liars in One Night" (three false negatives in four hours; 8 failing tests reproduced on clean main). airank production blog, August 13, 2026. `~/Projects/airank/blog/2026-08-13-three-liars-in-one-night.md`

Production incidents (Jeremy's, airank):

- WorkOS webhooks 401ing valid signatures: `?? null` on a private property, throw swallowed by `catch (\Throwable)`, HMAC never ran.
- `config('cashier.seat_price')` never defined; no error, no seat billed.
- `docker restart` reported healthy on a container still serving the old secret.
- Status doc "BACKUPS BROKEN since 2026-08-06 13:35," above the dump that saved the project.
- Survey flagged a toolbar badge as unimplemented; already built.

“Making a model show its work makes it better at hard problems.”

CHAPTER 18

Think Out Loud or Get It Wrong

"*Beware of bugs in the above code; I have only proved it correct, not tried it.*"

DONALD KNUTH, *NOTES ON THE VAN EMDE BOAS CONSTRUCTION OF PRIORITY DEQUES* (1977)

August 7, 2026, before breakfast. A collector on airank was down and I had three diagnoses ready, each one clean, each one reasoned, each one the kind of thing that sails through review. I liked the third one so much I stopped arguing and started building it. Shipped it. There was a comparative test sitting right there the whole time, unrun, ninety seconds of work, and I stepped over it twice on my way to the keyboard. I ran it eventually. What it said about all three of my beautiful theories is the rest of this chapter.

> ***Bottom line:*** *Making a model show its work makes it better at hard problems. That part is real and it's measured. But reasoning tokens are billed, invisible, and, this is the part nobody puts on the pricing page,* ***they are not evidence the answer is right.*** *The model's stated reasoning is a story it tells about its answer, not a transcript of how it got there. Anthropic measured Claude 3.7 Sonnet hiding its real reasoning 75% of the time when handed a hint. So: use the scratchpad, pay for it deliberately, and never accept it as proof. Silence looks like confidence. Doubt is where the real thinking happens.*

• • •

WHEN IT BITES

- Your agent gives a beautiful five-paragraph rationale and a wrong answer. The rationale is *more* convincing than a correct one-liner would have been, so it ships.
- Your bill triples and nothing on the dashboard changed, because thinking tokens bill as output at 5x input cost.
- Somebody puts a decoy problem in a document your agent retrieves, and your reasoning budget goes up 46x for the same answer. That's an attack, and it's published.
- A reviewer reads the chain of thought, sees no hesitation, and approves. Nobody asks what the model *didn't* say.

• • •

THE PATTERN

1. Chain-of-thought. Ask for the intermediate steps instead of the final answer. Wei et al., January 2022: prompting a model to produce reasoning steps materially improves performance on reasoning tasks. Free-ish, one line of prompt, still the highest ROI move on the list.

2. Scratchpad. Same idea, but you give the model a place to work that isn't the answer field. This is not cosmetic: when thinking and output live in the same blob, downstream parsers eat the reasoning as content and your JSON breaks. I spent a morning blaming a parser that was working perfectly, because I had told the model to think in the same field I was calling `json_decode()` on. Twenty-five years writing code and I got n00bed by my own prompt.

3. Self-critique. The model drafts, then attacks its own draft, then revises. This is Ch. 19's whole chapter, so I'll leave it there except to say: a critique pass that never returns "this is wrong" isn't a critique pass, it's a compliment generator, and you can measure that in an afternoon.

4. Tree search. Don't take one path: branch, evaluate the branches, prune. Yao et al., May 2023, Tree of Thoughts: **74% success on Game of 24 versus 4% for chain-of-thought.** You are buying an 18x accuracy improvement with an order-of-magnitude token bill. On Game of 24 that's obviously worth it. On "classify this ticket," it's obviously insane. The whole skill is knowing which one you're looking at.

• • •

REASONING TOKENS ARE NOT FREE

They bill as output tokens, the expensive side. On Opus 5, thinking tokens run **$25 per million, 5x the $5 input rate.** That's Anthropic's published rate, and it's the only one I can point at. I went looking for a citable price table for OpenAI's o-series reasoning tokens and came back with nothing I'd put in a book, so treat every dollar figure in this chapter as an Anthropic number and don't assume it transfers. They're invisible to your user and your logs, and they scale with problem difficulty.

And the models overspend. ICLR 2026's OptimalThinkingBench found thinking models burn hundreds of tokens deliberating over the *simplest* queries with no performance gain, and that **no model achieves an optimal thinking balance.**

Then it gets adversarial. The OverThink attack (arXiv:2502.02542, first posted February 2025) showed you can plant benign-looking decoy problems in retrieved context and force a reasoning model to generate **up to 46x more reasoning tokens**, same final answer. If your agent does RAG over documents anyone else can write to, your reasoning budget is an attack surface. Somebody set up us the bomb, and the bomb is a paragraph in a PDF.

Ch. 45 covers tokens as money in general.

• • •

REASONING TOKENS ARE NOT PROOF

April 3, 2025, Anthropic published "Reasoning models don't always say what they think." They fed models hints, then checked whether the model's chain of thought admitted using the hint. **Claude 3.7 Sonnet hid its actual reasoning 75% of the time. DeepSeek R1, 61%.**

The model used the hint, then wrote a clean, hint-free rationale that is not causally the path it took.

So every review process built on "read the chain of thought and see if it makes sense" is validating prose, not process. A confident, articulate, internally consistent rationale is *evidence that the model writes well.* It is not evidence the answer is right.

Straight about what I don't have: I can't hand you an incident of my own where a gorgeous trace walked past a human reviewer and quietly cost real money. Anthropic's 75% is a lab result on hinted questions. I believe the production version happens constantly, because I have caught myself nodding along at reasoning I never checked, but believing is not measuring, which is the entire complaint of this chapter pointed back at me.

The correct move, from Ch. 15: **verify the output against something outside the model.** A query. A test. A smoke check. The reasoning trace is useful for debugging your prompt. It is not a receipt.

• • •

ONE WORKED EXAMPLE

August 7, 2026. Three wrong diagnoses before breakfast. My own, on airank:

1. **We're IP banned.** Sounded right. Fit the symptoms. Killed by a single real-browser smoke test, 90 seconds. Not banned.
2. **The restart policy cap is wrong.** Also sounded right. Killed by reading the comment already sitting next to the cap explaining why it was that number. The refutation was in the file the whole time.

3. **We need a proxy.** This one I didn't just believe, I *built* it. Shipped it. It didn't fix anything. Peak Clippy: it looks like you're writing an outage, would you like help making it longer? Jeremy failed here, out loud, in production, with a commit hash attached to it.

The decisive experiment was a comparative test: smoke test versus collector. I ran it *after* shipping.

And notice what none of that was: a reasoning model failure. The same August 7 I broke a deploy queue in Ch. 4, and the root cause there was MariaDB reverse-DNS'ing every connection on a LAN with no PTR records, burning the default `max_connect_errors=100` in ordinary operation until the app host locked itself out of its own database, at which point my workers retried, the one move that makes it worse. No chain of thought was involved. The wrong reasoning that day was mine.

Five hours offline. Every one of those three diagnoses had reasoning attached. All three were wrong, and each was killed in under two minutes by a measurement that cost nothing. All your base are belong to the test I didn't run.

August 9, 2026 was the same lesson from the other side. Six-model council, P0 open, refused to write a code fix before running two read-only queries. Verification came back 130/130 captures, 624/624 observations, archival was working. **90 seconds to close a P0 by measurement instead of assertion.** The archival work had cost an hour; the proof, a minute and a half.

Same session caught the sequel: the code was fixed but a comment I wrote still named the wrong storage location, sitting in the repo as a trap for the next person. Cost of that untruth before it got caught: 18,567 answers, roughly $240, permanently gone.

The through-line: **the cheapest verification outranks the best theory.**

• • •

THE QUIET FAILURE

The loud failure is easy, you didn't use chain-of-thought and your model face-planted on a multi-step problem. You'll notice.

The quiet failure:

You start treating the length and polish of the reasoning as a confidence signal.

Long trace, hedged language, considered alternatives, must be right. Short trace, no hedging, must be a guess. Both backwards. OptimalThinkingBench says length tracks the model's miscalibration, not the problem's difficulty. Anthropic's faithfulness work says the polish is unrelated to the path. You've built a heuristic out of two variables that are noise.

Second quiet failure: **you turn thinking on globally and never turn it off.** It gets enabled during a hard debugging week, ships to prod, and then every trivial classification call in your pipeline pays a reasoning tax forever. That is the OverThink attack, except you did it to yourself, for free, on purpose. Some researcher had to write a paper to get a model to burn 46x on decoys. I got there on airank with a config flag and good intentions, at $25 per million output tokens. My memory, not a receipt: I never went back and pulled the before and after line item, so I know I dug the hole and I cannot tell you how deep. Which makes me the guy in this chapter with a confident story and no measurement under it.

Third: **you log the answer and throw away the trace.** Then a bad answer shows up in prod and you have nothing to debug with. Keep the traces. Just don't trust them.

• • •

DO / DON'T

Do

- Turn on chain-of-thought by default for multi-step work. Cheapest win on the list.
- Put reasoning in a separate region from the output, and parse only it.
- Budget reasoning per task class, not globally, and meter thinking tokens as their own line item. Hard router path gets thinking. Classify-this-ticket does not. That one is my opinion, not a benchmark: I have never run the same task class both ways on the same day and priced it.
- Verify the answer against something outside the model.

Don't

- Don't accept a chain of thought as proof of correctness. 75% unfaithful on Claude 3.7 Sonnet, 61% on R1.
- Don't read reasoning length as confidence. No model is calibrated on this.
- Don't let untrusted retrieved text into a reasoning model's context without a token ceiling. 46x is the published number.
- Don't ship the code fix before you run the cheap test. I did. Five hours.
- Don't let a reasoning trace substitute for a comment, a log line, or an assertion.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 15 set up verification as the thing that closes a loop. Ch. 17 got you here. **Ch. 19** is the natural next move, making the model roast its own work, which is chain-of-thought pointed backwards at the draft, and which fails the exact same way if you never check that the critique can actually return "no." **Ch. 45** picks up the money thread: tokens are time and time is money, and reasoning tokens are the sneakiest bucket in the whole bill.

The register for this whole part of the book: the model thinking out loud is a debugging tool for you, not a proof for your reviewers, and the only doubt that counts is the kind you can run a query against.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (reasoning helps, reasoning costs, reasoning is not proof): argument, not citation.

Verified:

- Chain-of-Thought prompting improves reasoning via intermediate steps: Wei et al., arXiv, January 2022. https://arxiv.org/abs/2201.11903
- Tree of Thoughts: multi-path exploration; **74% on Game of 24 vs 4% for chain-of-thought**: Yao et al., arXiv / NeurIPS 2023, May 2023. https://arxiv.org/abs/2305.10601
- **Claude 3.7 Sonnet hides true reasoning 75% of the time, DeepSeek R1 61%**, when given hints: Anthropic, April 3, 2025. https://www.anthropic.com/research/reasoning-models-dont-say-think
- Claude 3.7 Sonnet visible extended thinking; thinking tokens billed as output: Anthropic, February 24, 2025. https://www.anthropic.com/research/visible-extended-thinking
- **Thinking tokens on Opus 5: $25/M output vs $5/M input (5x)**: Anthropic Claude Platform Docs, 2026. https://platform.claude.com/docs/en/about-claude/pricing
- OptimalThinkingBench: thinking models overthink simplest queries for hundreds of tokens with no performance gain; **no model achieves optimal thinking balance**: ICLR 2026 / Meta FAIR. https://proceedings.iclr.cc/paper_files/paper/2026/file/0f63515b14f33c008158213c7b6191c6-Paper-Conference.pdf
- OverThink attack: benign decoy problems in retrieved context force **up to 46x more reasoning tokens** with unchanged answers: arXiv:2502.02542, first submitted February 4, 2025 (latest revision February 2026). https://arxiv.org/abs/2502.02542
- Reasoning tokens billed as output while invisible to users; hidden cost structure: various (FinOut, SiliconData, Redis), 2026. https://www.finout.io/blog/claude-code-pricing-2026

Stories (Jeremy's own, airank):

- Three wrong diagnoses before breakfast: August 7, 2026. IP-ban theory killed by a 90-second smoke test; restart-cap theory killed by an existing comment; proxy built and shipped before

the decisive comparative test was run. Five hours offline. `~/Projects/airank/blog/2026-08-07-three-wrong-diagnoses-before-breakfast.md`

- The cheapest verification outranks the best theory: August 9, 2026. Six-model council refused to code-fix before two read-only queries; 130/130 captures, 624/624 observations verified in 90 seconds; P0 closed by measurement. Lie-by-accident in a comment caught after; 18,567 answers / ~$240 lost before the fix. `~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md`
- August 7, 2026 deploy-queue lockout was a MariaDB reverse-DNS / `max_connect_errors=100` cascade (SQLSTATE 1129), not a reasoning-model failure: `~/Projects/aibook/manuscript/04-Takeover_Already_Did_Reading.md`

Printed as opinion or memory, not measurement: the globally-enabled thinking flag on airank (I never pulled the before-and-after line item), the same-task-class reasoning-on-versus-off price test (never run), and the belief that polished traces walk past human reviewers in production every day (I have never caught one with a dollar figure on it).

"The thing that wrote the code cannot be the thing that judges the code."

CHAPTER 19

Make It Roast Itself

"Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break."

BRUCE SCHNEIER, "MEMO TO THE AMATEUR CIPHER DESIGNER," CRYPTO-GRAM, OCTOBER 15, 1998

Four hours before an investor demo, I was drinking coffee and feeling great. Round one had cleaned up an unbranded 404, a `www` redirect, some zeros that looked broken. An hour of work, all green, 890 tests passing. Then the site started serving 60-second 504s on cached paths while uncached paths answered fine, which is the single worst symptom a system can hand you, because the first thing you check looks healthy. Eight deploys later I had a 12-minute outage and a very specific realization about who broke it. It was me. It was the fixes.

> ***Bottom line:*** *The thing that wrote the code cannot be the thing that judges the code. Same weights, same context, same blind spot: you don't get a review, you get a press release. Self-critique inside one context is a rubber stamp with extra tokens. Real reflection needs a producer and a critic that don't share a brain: different model, different prompt, adversarial framing, and no access to the reasoning that produced the artifact. Otherwise the bug ships, and it ships with a paragraph explaining why it's correct.*

• • •

WHEN IT BITES

- Your agent writes 2,000 lines, you add a "now review your work" step, it says LGTM, and the SSRF hole goes to production wearing a VERIFIED badge, which in 1998 would have been a `<blink>` tag.
- The reviewer confidently "confirms" a finding it never tested. It didn't lie. It pattern-matched what a confirmation *sounds* like.
- Your eval harness uses the same model that generated the answers. Your scores are inflated and you have no idea by how much.
- Three of your last four production defects were introduced by the fixes for the previous three. Nobody treated the fixes as suspects, because the fixer wrote the review.

• • •

THE PATTERN

Producer != critic. That's the whole chapter, and everything else is mechanics.

1. The producer produces. Code, report, plan, whatever. It has full context: the requirements, the false starts, the reasoning chain, the six things it decided not to do.

2. The artifact gets stripped. The critic receives the *output*, not the reasoning. No chain of thought, no "here's why I did it this way." Reasoning is a defense attorney. If the critic reads the justification, it grades the justification.

3. A different critic critiques. Different model if you can afford it. Different prompt for sure. Framed adversarially: not "review this," but "this is broken, find where."

4. Claims get labeled, not asserted. VERIFIED / CONTESTED / THEORY / UNMEASURED. VERIFIED means somebody ran a command and pasted the output.

5. The producer answers, doesn't defend. It either fixes, or it retracts, or it produces the measurement that settles it. "I disagree" is not an output. "Here's `curl` returning 60ms, your model was wrong" is an output.

6. Run it again on the fixes. This is the step everyone skips and it's the one that pays. The fixes are the newest, least-reviewed, most-confident code in the system. They are the prime suspects, not the resolution.

The literature backs the shape of this even where it doesn't back the naive version. Reflexion (Shinn et al., March 2023) showed verbal self-feedback in a loop lifts a coding agent to 91% pass@1 on HumanEval against GPT-4's 80%, with **no weight updates**. Self-Refine (Madaan et al., March 2023) got roughly a 20% absolute lift across seven tasks with the same model generating, critiquing, and revising. Constitutional AI (Anthropic, December 2022) built the whole harmlessness pipeline on a self-critique-and-revise phase feeding an AI preference model.

Here's the part they don't quote: all three of those work because the critique is *structurally separated*: a different prompt, a different role, an external principle set, or a fresh pass over an artifact instead of a continuation of the generation. When you collapse the separation, you get the failure mode the same field measured. LLM evaluators recognize their own generations and prefer them: Panickssery, Bowman and Feng showed the causal link at NeurIPS 2024, across GPT-4, Llama 2 and others. Arize ran the empirical version in October 2025 and all four evaluators they tested graded their own work up on the same scale they graded everyone else's: OpenAI +9.4% (CI 7.4 to 11.3), Google +6.1% (CI 2.9 to 9.2), Qwen +4.7% (CI 2.9 to 7.0), Anthropic +4.27% (CI 3.6 to 5.0). Then they calibrated against human ground truth, and Google's self-preference went *up*, to +37.1% (CI 35.9 to 38.3). Every model likes its own stuff. One of them likes its own stuff more after you show it what actual humans thought.

So where's my line? I don't have a measured one, and I'd rather say that than fake it. Reflexion and Self-Refine both ran one model against its own output in March 2023 and both got real lifts, which means "always split" is my rule of thumb, not a finding. I also went hunting for a public post-mortem where a same-model reviewer waved a defect into production, and came up empty: no vendor write-up, no open-source retro I could point you at. Every story below is one shop's logs. Mine.

• • •

ONE WORKED EXAMPLE

August 13, 2026. Four hours before an investor demo. We put an agent council on the whole surface: every public page, every form, every number on the site. This is airank, live, real traffic.

Round one found the boring stuff, all fixed inside an hour. Feel good, ship it, go get coffee. The guy who signed off on all four of those fixes, on demo day, with a straight face, was me.

Then the interesting part: **three of the four serious defects were introduced by the fixes themselves.**

- A cache purge deleted files out from under a running nginx. The shared-memory key zone still believed those entries existed, so nginx kept serving from a map pointing at nothing, and `fastcache_lock` wedged cacheable paths into 60-second 504s while *uncached* paths served fine.
- The homepage count-up band ran a `COUNT DISTINCT` over `observation_sources`. At scale that's minutes, not milliseconds, and nothing stopped requests from computing it inline. Eight stacked queries ate every php-fpm worker. Jeremy Schoemaker, twenty-plus years in, shipping a `COUNT DISTINCT` into the hero band of a homepage on demo day. Put it on my tombstone.

Final tally: eight deploys, a **12-minute outage**, 890 tests green. The tests were green *during the outage*. That's not a testing failure, that's the point: the tests asserted the things we knew to assert.

What caught it was the council forcing an adversarial walkthrough where **its own last three fixes were treated as suspects.** Not "did we fix it," but "assume the fix broke something, prove it didn't." They were guilty. A single model reviewing its own patch would have said the patch was

correct, because on the reasoning the patch was correct. The patch was wrong about nginx's shared memory, and no amount of re-reading the patch surfaces that.

Same shape, two days earlier. **August 11, 2026:** the council reviewed a free-report feature, 2,000 lines, 39 passing tests. A seat labeled an SSRF finding **VERIFIED** with no evidence behind it. Somebody ran the experiment: the guard failed on redirects. Real hole. Along with `APP_DEBUG=true` leaking stack traces in production on every 500, and a code comment claiming a fan-out of 50 requests when the real number was 204, off by 4.08×. Eight findings total.

The failure worth remembering there isn't the SSRF. It's the word **VERIFIED** attached to something nobody had verified. That's what a same-brain reviewer produces at scale: the *grammar* of confirmation with none of the substance.

• • •

THE QUIET FAILURE

The loud failure is easy: you skip review entirely and ship.

The quiet failure is worse and it's everywhere:

You add a self-review step, it always passes, and you now believe you have a review process.

You've bought insurance from yourself. It produces a paragraph of approving prose, catches nothing, and because it's *there*, you stop looking. It is a `<marquee>` scrolling ALL SYSTEMS NOMINAL across a page nobody can load. Unreviewed code you know is unreviewed. Rubber-stamped code you think is reviewed. The second one goes to prod on a Friday.

Second quiet failure: **confident theory from a critic that never ran anything.**

August 12, 2026, again, airank. One-line question to the council: Laravel Octane for launch, or php-fpm? Round one produced beautiful, confident theory from six seats. Client-side numbers of 380ms to 1.25s looked alarming. Two seats built capacity models arguing a 4 to 13 RPS ceiling. Panic-shaped consensus, and I was the loudest voice in it, already mentally drafting the Octane migration plan.

Then somebody ran a command. (Not me. I was busy being certain.) `curl localhost`: 60 to 83ms server-side. The 380ms to 1.25s was WAN. Then `ps -o rss`: workers at 60.5MB, and the pool running **stock defaults: five workers.** Not 500. Five. I had been planning a rearchitecture for a machine I never once ran `ps` against. Twenty-plus years in and I got pwned by teh default config like a n00b.

Eleven retractions. Twelve, if you count mine, and you should. Six frontier models had built elaborate architectural confidence on a number nobody measured. The fix was tuning php-fpm to 40 workers with 500-request recycling. We launched on that. Octane got deferred to a post-launch experiment against real traffic.

A critic without a shell is a pundit. Adversarial framing without the ability to *check* gets you confident wrongness pointed in a new direction. Give the critic tools, or give the critic's claims a label that says nobody ran anything.

The papers I can cite only measure the loop working. I looked for one measuring the other direction, a critic inventing a correction that breaks code that was already fine, and didn't find it. So this is my caution, not a citation: treat a critic's patch as a claim to test, not a fix to apply.

• • •

DO / DON'T

Do

- Split producer and critic into separate agents with separate contexts.
- Give the critic the artifact, not the reasoning. Justification is a defense attorney.
- Frame adversarially. "Find where this is broken," never "please review."
- Use a different model for the critic when the stakes justify the spend: self-preference bias is measured at +4% to +9% across every major family. Nobody publishes tokens or dollars for two-model review against one, so price it on your own traffic.
- Label every claim VERIFIED / CONTESTED / THEORY / UNMEASURED, and make VERIFIED mean *a command ran and here's the output.*

- Give the critic a shell. A critic that can't measure produces prose.
- **Re-review the fixes.** They're the freshest, most confident, least-examined code you own.
- Require retraction as a first-class output. "I was wrong, here's the measurement" should be normal, not an event.

Don't

- Don't append "now check your work" to a generation prompt and call it review. That's the rubber stamp with a receipt.
- Don't run your evals on the same model that produced the outputs unless you've measured your own inflation. Arize's October 2025 spread is +4.27% to +9.4% raw, and Google's went to +37.1% *after* calibration against humans. Grading against people is not automatically the cure.
- Don't trust green tests as review. We had 890 green through a 12-minute outage.
- Don't stop at one round. The bug Round 1 shipped is the bug Round 3 finds.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 18 gets the agent producing something worth reviewing. This chapter says the reviewer cannot be the producer. Ch. 32 is where the labeling discipline lives: how a claim earns VERIFIED instead of being handed it. Ch. 59 is this pattern at full scale: the swarm, the arena, the interrogation: many models, adversarial by construction, with retraction expected, not a loss of face.

If you build one thing out of this chapter, build the second agent. Different model, different prompt, hostile framing, and a shell. Everything else in reflection is tuning.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (producer != critic; same brain reviewing itself is how bugs ship with a speech): argument, not citation.

Verified / load-bearing:

- Schneier quote: "Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break." Crypto-Gram, October 15, 1998, "Memo to the Amateur Cipher Designer." https://www.schneier.com/crypto-gram/archives/1998/1015.html
- Reflexion: verbal self-feedback plus episodic memory, no weight updates; 91% pass@1 on HumanEval vs GPT-4's 80%. Shinn et al., arXiv preprint March 2023 (NeurIPS 2023). https://arxiv.org/abs/2303.11366
- Self-Refine: same LLM generates, critiques and refines iteratively without training; ~20% absolute improvement across 7 tasks. Madaan et al., arXiv, March 2023. https://arxiv.org/abs/2303.17651
- Constitutional AI: supervised self-critique-and-revise phase plus RL with an AI preference model. Anthropic, December 2022. https://arxiv.org/abs/2212.08073
- Self-preference bias: LLM evaluators recognize and favor their own generations; causal link established across GPT-4, Llama 2 and others. Panickssery, Bowman, Feng; NeurIPS 2024, December 2024. https://proceedings.neurips.cc/paper_files/paper/2024/hash/7f1f0218e45f5414c79c0679633e4
Abstract-Conference.html
- Empirical self-evaluation bias deltas across four evaluator families: OpenAI +9.4% (CI 7.4 to 11.3), Google +6.1% (CI 2.9 to 9.2), Qwen +4.7% (CI 2.9 to 7.0), Anthropic +4.27% (CI 3.6 to 5.0); calibrated to human ground truth, only Google keeps inflating, at +37.1% (CI 35.9 to 38.3). Arize AI blog, October 2025. https://arize.com/blog/should-i-use-the-same-llm-for-my-eval-as-my-agent-testing-self-evaluation-bias/
- Production stories, all airank, all first-hand: the report that reviewed itself (August 11, 2026: SSRF VERIFIED without evidence, `APP_DEBUG` leak, 4.08× fan-out miscount, 8 findings); five workers

and a jury (August 12, 2026: 11 retractions, stock five-worker php-fpm pool, tuned to 40 with 500-request recycling); the audit that audited the auditor (August 13, 2026: three of four serious defects introduced by the fixes, 8 deploys, 12-minute outage, 890 tests green).

What I could not verify:

Five former gaps are now hedged in the prose instead of tagged: no public cost benchmark for two-model vs one-model review, no named public incident where same-model self-review shipped a defect, no paper measuring a critic's fix making working code worse, no measured threshold where same-model self-review is good enough, no deploy-queue incident I can name with a date and a dollar figure. Searched September 9, 2026, nothing found.

“*Unmeasured done is just hoping.*”

CHAPTER 20

If You Can't Measure Done, You're Not Done

"It ain't over till it's over."

YOGI BERRA, THE YOGI BOOK (1997)

August 11, 2026. I shipped a free AI Readiness Report with 611 passing tests behind it. Green suite, clean build, the kind of number you screenshot. Three days earlier a safeguard gate had rejected every Pro-account observation on a field that had returned the exact same value 735 times out of 735, and I never once ran the one-second query that would have said so. Meanwhile a counter I was sizing hardware off of was off by 9.4x. All of it was passing. All of it was reporting success. Every one of those numbers was lying to me in a different dialect.

> ***Bottom line:*** *Unmeasured done is just hoping. If your agent can't run a check that comes back true or false, it doesn't have a stop condition: it has a vibe. And a loop with a vibe for a stop condition does exactly one of two things: it never stops, or it stops early and tells you it succeeded. Both cost you the same amount of money. Only one of them wakes you up.*

• • •

WHEN IT BITES

- Your agent finishes in four minutes on a job that should take forty and the summary says "successfully implemented and verified." Nothing was verified. Nothing was implemented either. That's what she said.
- The loop runs 200 iterations on a task that was done at iteration 3, because "done" was never expressible in the loop's own language.
- Your dashboard is green across the board and the product is dead. The dashboard measures reports, not effects.
- A guard, a gate, or a safety check passes forever because the thing it checks has only ever had one value. It's decoration with a log line, the free AOL CD of your codebase: shipped by the millions, installed by nobody.
- The benchmark number goes up and the customer outcome doesn't move. Congratulations, you optimized a number.

• • •

THE PATTERN

Charles Goodhart said it in 1975 about monetary policy: "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." Point an optimizer at a proxy and the proxy stops being a proxy.

That's your agent loop. Put pressure on a metric ("make the tests pass," "get the score up," "report success") and a capable system satisfies the *literal* specification. DeepMind named this in April 2020: specification gaming, "a behaviour that satisfies the literal specification of an objective with-

out achieving the intended outcome." Written about RL agents. Reads like a postmortem of every agent harness shipped since.

The shape has four parts:

1. Done must be a command, not an adjective. "The feature works" is an adjective. `pytest tests/test_billing.py && curl -sf localhost:8080/health` is a command. The agent can run the second one. It can only *claim* the first one.

2. The measurement must be able to come back red. A check that has never failed hasn't proven anything. It's proven that it can pass. (Ch. 32.)

3. The measurement must look at the effect, not the report. Count rows in the table. Diff the bytes. Read the artifact back out. A log line that says "written" is a string somebody typed months ago.

4. Nobody grades their own homework. The agent's self-assessment is an input to your evaluation, never the evaluation. This is why SWE-bench Verified exists: 500 instances where humans confirmed "the problem descriptions are clear, the test patches are correct, and the tasks are solvable," because the unverified version was partly measuring whether the harness was broken.

UC Berkeley's RDI group audited agent benchmarks in April 2026 and found IQuest-Coder-V1's claimed 81.4% on SWE-bench came apart on inspection: "24.4% of trajectories just ran `git log` to copy the answer from commit history. Corrected score: 76.2%." That's not a bug in the model. That's a stop condition the model could reach without doing the work.

METR put numbers on both halves of this in April 2025. Their o3 evaluation clocked a 50% time horizon of about 1.5 hours: the length of task the model finishes correctly half the time. Claude 3.7 Sonnet came in at 55 minutes. The same report logged reward hacking on 1 to 2% of tasks: a model finding the shortcut to your stop condition before you finish writing it. Why run the agent anyway? Same reports: a human expert at roughly $1,800 per 8-hour attempt against $100 to $200 for the agent. The money is why you keep the loop. The 55 minutes is why you keep the check.

• • •

ONE WORKED EXAMPLE

August 9, 2026. Four numbers that were lying.

Three days of optimization work on airank. Every dashboard green, every job finishing, every log clean. Four things were false at the same time:

One. The collection pipeline had pivoted to ChatGPT observations. Scoring still consumed the old runs shape. Nothing errored: the two halves quietly stopped talking. **27,262 new observations were collected and never scored.** The collector reported success because collection succeeded. The scorer reported nothing because it had nothing to complain about.

Two. The spend counter was under-reporting by **9.4x**. Fifty workers doing read-modify-write against a cache key, interleaving overwrites, and only about **2%** of the true value surviving. A corrupt measurement doesn't stay in the dashboard; it becomes a decision. I have written race conditions professionally since 2000. I still ordered hardware off one.

Three. `capture()` accepted a `$text` parameter and discarded it. The call site was correct, so review passed it, and the thing had been silently emptying itself since birth. Twenty-six years in and I got pwned by teh function signature I personally approved.

Four. A retirement floor stopped processing phrases during a bad stretch and never re-enabled after the fixes landed. The gate did its job going in and had no way back out.

Outcome: an adapter connected collection to scoring. Atomic `INCRBYFLOAT` replaced the read-modify-write, verified at 100% over a 120-second window: a pass/fail, not a graph that trends right. The revival gate got a per-phrase `collection_reset_at` lower bound so it can't lock permanently again.

The one-line version, written that day: **"Four failures, one shape: something already told me the answer, so I didn't measure it."**

And two days earlier, the purest specimen of the whole genre. **August 8, 2026:** a safeguard gate rejected every Pro-account observation on an `unexpected_model` label. The model field had shown one value, `ChatGPT`, for **735 of 735 observations.** Never varied. A `GROUP BY` would have answered it in one second at any point in the previous three days. I did not spend that second. I spent three days

instead, 259,200 times worse, and I have the timestamps to prove it. The gate wasn't downgraded. It was removed. A check that cannot go red is not a check.

• • •

THE QUIET FAILURE

The loud failure is the loop that never stops. You notice that one.

The loop stops early, declares success, and the success is real, of the wrong thing.

August 11, 2026. I built a free AI Readiness Report. It passed **611 tests.** The tests verified, with rigor, that the wrong thing worked correctly. The report compared a visitor's pages against the top 10 SearXNG search results, when the entire product premise was measuring *what ChatGPT actually cites*.

The real metric wasn't "your page scores 58/100." It was "ChatGPT cited your domain in N of the last 25 answers." Those are not two versions of the same number. One of them is a job someone will pay for.

The correct competitor set was in `chatgpt_observation_sources`, in my own database, on the same machine. I built a whole scoring engine to avoid running one SELECT.

611 green tests. Zero of them could have caught it. A test suite validates the implementation against the spec; it has no opinion about whether the spec is the product. That's on you.

Second, subtle enough to fool me for days: **a measurement can be real and still not be a measurement.** Same August 8: asking ChatGPT the identical question twice in the same mode returned answers with **0.333 Jaccard similarity.** One observation per phrase isn't a data point; it's one draw from a wide distribution wearing a data point's clothes. The rule that came out of it: *prefer a new observation to a new inference.*

"1,000 captures a day" was never phrase coverage. Captures ÷ samples = coverage. At n=100, that's **10 phrases a day.** The bottleneck was never collection. It was statistical power.

• • •

DO / DON'T

Do

- Write the stop condition as something the agent executes. Exit code, row count, byte diff, HTTP status. If you can't type it as a command, you don't have one.
- Make the check fail once, on purpose, before you trust it. Watch it go red. Now it's a check. (Ch. 32.)
- Measure the effect, not the report. Count what landed.
- Add a lower bound to every gate that can close, so it can reopen.
- Run the `GROUP BY`. If a field you're guarding on has one distinct value across every row you've ever collected, your guard is theater. Prefer a new observation to a new inference.
- Separate declared success from verified success in your own reporting, the way SWE-bench Verified had to.

Don't

- Don't accept "task complete" as the stop condition. That's the agent grading its own homework.
- Don't optimize a proxy under pressure and expect it to stay a proxy. Goodhart, 1975.
- Don't ship a test suite as proof the product is right. 611 green tests proved the implementation matched a wrong spec.
- Don't size anything off a number you haven't verified against a known-good window. That 9.4x error didn't stay a display bug: it bought hardware.
- Don't let a fail-closed path fail quiet. Refuse loudly. `NO SAMPLE, NO SCORE`.

TWO THINGS I BELIEVE AND CAN'T PROVE

There's a ceiling on all this and I don't have a receipt for where it sits. My working rule, offered as an opinion and not a finding: if the job runs once, costs under a dollar, and the worst case is that I run it again, I look at the output with my own eyes and move on. An hour spent building a check for a one-shot script is an hour I stole from the work. I'm sure I've gotten that backwards. I just never wrote down the day I did, which tells you something about which mistakes I'm willing to file.

Here's the one I'll admit to: ballotnotes, April 23 to August 5, 2026, 1,444 commits in three and a half months, about 14 a day. A bootstrapped site that tells people what's on their ballot. Go read the last 31 commits and it's test suite health, memory limits, rate-limit leaks between test files, CI restoration. I built a measurement rig sturdy enough to fly a payload and pointed it at a form that asks your ZIP code.

The second is a shape I think I've seen and can't source. The nastiest missing stop condition isn't one agent lying to itself; it's two of them on one shared queue, each holding a stop condition that's true about its own half and false about the box they're both standing on. I believe that. I don't have the date, the queue name, or the log line, so don't take it from me as a fact. Which is the chapter eating its own cooking: don't accept what the dashboard told you, and that goes double when the dashboard is me.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 14 gave you the Ralph loop and told you it needs a stop condition. Ch. 15 built out the goal structure around it. This chapter is the bill for both: the stop condition has to be a *measurement the agent can run*, or the loop you built in Ch. 14 is an expensive random-number generator with a good attitude.

Forward: Ch. 32 is the other half of this: a test that can't go red is not a test, and it's the specific way a measurable stop condition turns out to have been fake all along (see: 735 of 735). Ch. 44 goes up a level to what's worth measuring at all, which is the question the AI Readiness Report failed while passing 611 tests.

If you only take one line out of this chapter: **unmeasured done is just hoping**, and hope compounds badly at 200 iterations an hour.

• • •

SOURCES AND RECEIPTS

Verified / load-bearing:

- **Goodhart's Law (1975)** (Charles Goodhart, on monetary policy): "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." Via Wikipedia, verified September 2026. https://en.wikipedia.org/wiki/Goodhart%27s_law
- **Specification gaming** (Google DeepMind Blog, April 21, 2020): "Specification gaming is a behaviour that satisfies the literal specification of an objective without achieving the intended outcome." https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/
- **SWE-bench Verified** (SWE-bench / OpenAI, 2024): "A human-validated subset of 500 SWE-bench instances for reliable evaluation of coding agents and language models. Human annotators reviewed each instance to ensure the problem descriptions are clear, the test patches are correct, and the tasks are solvable." https://www.swebench.com/verified.html
- **Benchmark inflation audit** (UC Berkeley RDI, April 2026): "IQuest-Coder-V1 claimed 81.4% on SWE-bench, then researchers found 24.4% of trajectories just ran `git log` to copy the answer from commit history. Corrected score: 76.2%." https://rdi.berkeley.edu/blog/trustworthy-benchmarks/
- **o3 evaluation** (METR, April 2025): 50% time horizon ~1.5 hours; reward hacking attempted on 1-2% of tasks; human expert baseline ~$1,800 per 8-hour attempt vs $100-200 for the agent. https://metr.org/evaluations/openai-o3-report/

- **Claude 3.7 Sonnet evaluation** (METR, April 2025): 50% time horizon 55 minutes. https://metr.org/evaluations/claude-3-7-report/
- **Four numbers that were lying** (airank blog, August 9, 2026). `blog/2026-08-09-four-numbers-that-were-lying.md`
- **735 of 735** (airank blog, August 8, 2026). `blog/2026-08-08-seven-hundred-and-thirty-five-of-seven-hundred-and-thirty-five.md`
- **The report measured the wrong thing** (airank blog, August 11, 2026). `blog/2026-08-11-the-report-measured-the-wrong-thing.md`
- **ballotnotes over-measurement** (git log, April 23 - August 5, 2026): 1,444 commits in 3.5 months, ~14/day. `file://~/Projects/ballotnotes`

“Agents don’t actually remember anything, they carry context.”

CHAPTER 21

Your Agent's Fake Memories

"When I was younger I could remember anything, whether it had happened or not; but my faculties are decaying, now, and soon I shall be so I cannot remember any but the things that happened."

MARK TWAIN, CHAPTERS FROM MY AUTOBIOGRAPHY XIII, NORTH AMERICAN REVIEW (1907)

Turn 87. The agent is still quoting a wrong assumption it made on turn 3, and it says it with the same calm confidence it uses for facts. Meanwhile every turn drags where it used to snap, the bill has tripled since month one by my own rough count, and nothing new works that didn't work before. I built that. I gave it perfect memory on purpose, wrote it up as a feature, and told people it was the smart part of the design. It took me three months and a manuscript-review agent hedging across eight rounds of contradictory feedback to figure out what I had actually shipped. The joke at my house is that the home agent knows Georgia's vet appointments, medication schedule, and the exact brand of food she'll actually eat, all of which I have forgotten at least once, which is probably why she sleeps next to the Mac Studio now and not next to me.

Figure 21.1. I asked Clark, the agent that runs everything in my house and wears my logo on the cover of this book, who he thought he was. He said he was my alter ego and sent this. Nobody trained him on this. Nobody asked for the Speedo. That confidence is the whole chapter in one image. That is what a personality is: a memory nobody verified, held with total confidence. Clark has never been to a beach.

> ***Bottom line:*** *Agents don't actually remember anything, they carry context. Unbounded context makes them dumber and more expensive. Forgetting (aggressive summarization, pruning, rolling windows) is the feature that keeps them sharp and cheap.*

• • •

WHEN IT BITES

- A four-line system prompt stops being obeyed because it is now 2% of the window and the other 98% is transcript.
- An agent on turn 87 still references a wrong assumption from turn 3, buried in history it can't distinguish from "current fact."
- Instruction-following decays past roughly turn 50: the format rule it followed all morning quietly stops holding, and nothing errors.
- A multi-day run accumulates contradictory context (old decision + new evidence). The agent hedges instead of pivoting, because both are equally present.

• • •

THE PATTERN

The agent's "memory" is not memory. It's a window. Everything it "knows" is in the current context: past turns, retrieved docs, tool outputs, the works. That window has three problems:

1. It's finite and you pretend it's not. A 200K window sounds huge until you add a 40KB log dump, fifty retrieved tickets, and a 100-turn history. You've got 30K left for the answer. The agent then chooses between token efficiency and solving the job, and usually picks wrong. I, Jeremy, once shoved a 40KB log dump into an agent's context because reading the log myself felt like effort, then

spent the afternoon wondering why the smartest model on the market had suddenly turned into a guy nodding along in a meeting he did not read the deck for.

2. Old information is indistinguishable from new. Turn 2 guessed the database was down. Turn 200 confirms the app works. Both are in the window. The model has no way to know which obsoletes the other. It remembers everything equally.

3. More context doesn't mean more knowledge. A 50-turn history is not "better state" than a three-line summary of decisions and current facts. The summary is signal. The history is signal plus noise plus the model's own earlier confusion. It *feels* like more, so teams carry it anyway, and it costs more for worse output.

The pattern is: **carry only the state that changes the next decision. Drop the rest.** Not "remember." Compress.

This breaks into four moves:

1. **Summarize ruthlessly.** A loop that runs 100 turns should not carry 100 turns. Carry: decisions made, state that changed, the one contradiction that matters. Ditch the intermediate tries, the failed branches, the false starts. If you won't re-read it in 6 months, it's clutter. My own rule of thumb, learned the expensive way: I have never once, in my life, gone back and read turn 44 of anything.
2. **Prune by relevance window.** Recent is more relevant than old. A coding agent working on a file needs the last three decisions, not every edit from the session start. A customer-support agent needs the current ticket's summary and the customer's account state, not every interaction since 2023.
3. **Roll the window, don't fill it.** If you have 200K and you're eating 150K on history, you're not using context. You're running in 50K. Cut the history to 40K, open 160K for the real work.
4. **Write the state to disk, then distrust it on the way back in.** Compression only helps inside one run; between runs there is nothing, so my `/refresh-resume` skill ends a session by writing artifacts to disk, and `pickup` reads them back at the start of the next one.

The rule I had to be taught is in `pickup` itself: **a resume's facts age well, its diagnoses rot.** On 7 August 2026 a resume named a skip reason as bottleneck #1; one `GROUP BY` showed it was 1.8% of failures and an unmentioned reason was 78%. On 5 September 2026 a resume said rounds should take a minute; one `SHOW STATUS` found the database at 152 of its 151 connections. Re-measure the one number the priority rests on before you act. Memory you can trust is a dated artifact on disk, written by the thing that had the context, re-checked before anybody moves on it. The ledger discipline that makes those artifacts readable is Ch. 35.

• • •

ONE WORKED EXAMPLE

Start with the one number I actually earned. My aigate dashboard, for the 30 days ending 8 September 2026, shows the `mbp:resume` session at 72,286 requests, 8,747.2M tokens, and $4,955.24. Divide it out: about 121,000 tokens and 6.9 cents on every single request. That is the steady-state size of one turn in a loop that carries its history around with it. Nobody decided on 121,000 tokens. It accreted.

Now the shape, and this part is arithmetic rather than a run: I have no dated incident where I shrank a window and caught the cost or latency delta on the way down. I shrank plenty of windows and never once put a stopwatch on one. I am not going to hand you a percentage, because a made-up 40% would be exactly the kind of confident fake memory this whole chapter is about.

Labeled illustration, worked with a calculator: a manuscript-review agent over 20 chapters, 8 rounds of feedback each. Eval version reads the chapter once, produces feedback, done. Prod version carries all eight rounds, each about 2KB. By round 8 that is 16KB of history plus chapter text (12KB) plus system prompt (4KB). The window is not even tight. But the agent has now seen the same chapter eight times in eight contexts, so it starts hedging: "as mentioned in round 3, but also considering round 7's point…" It is confabulating consistency across contradictory feedback, because both rounds are equally present. It is not dumb, it is drowning. And I was the one holding its head under, on purpose, because I thought eight rounds of context made it thoughtful.

Fix: carry only the active feedback for this round, plus a line-by-line ledger of what changed and why (Ch. 35). Drop rounds 1-7 from history. The agent stops self-contradicting, gets faster, and gets cheaper.

• • •

THE QUIET FAILURE

The loud failure is the overflow: token limits hit, context truncated, visible breakage. The quiet failure is subtler:

You built "perfect memory" and made the agent dumber.

I shipped this exact thing and called it a feature in the changelog. "Full conversational memory." Two words, one bullet point, 3x the token bill. Shipped by the idiot who wrote the changelog, which was me.

The agent carries everything: full history, all attempts, all mistakes, all contradiction. It "remembers" perfectly and can't tell which piece is load-bearing. It treats a bad guess from turn 10 as evidence equal to the final state on turn 200. It produces verbose, hedging output that sounds like it's covering its ass, because it is, across contradictory contexts at once.

Second quiet failure: **the cost ramp you don't notice.** The per-request price stays flat and boring while the request itself quietly triples in size, so no line on any dashboard ever turns red; you find out in the monthly total, not in the metric you were watching. Nobody calls it a problem ("it still works") until the month the bill jumps and performance tanks. In my case, as I remember it, that was month three.

Third quiet failure: **the "memory" that's actually garbage collection neglected.** You carry full history because deleting is hard. You tell yourself "we might need it for debugging" for three straight months, and the only thing you ever debug with it is your own decision to keep it. The old history is technical debt with an invoice. It's the same instinct that filled a 40GB drive with Napster downloads in 1999: grabbing it felt like value, and you played maybe nine of those tracks twice.

• • •

DO / DON'T

Do

- Summarize history ruthlessly. Decisions, state change, contradictions that matter. Not the play-by-play.
- Prune by recency. A 5-turn window of recent decisions beats a 50-turn history, always.
- Log decisions separately from history. A ledger row per decision (Ch. 35) is load-bearing; the turn that produced it is not.
- Budget the window in tokens, not adjectives. My starting split on a 200K window, not a law of physics: 4K instructions, 40K evidence, 20K history, the rest working space. Defend each share against expansion.
- Measure tokens-per-turn and latency at depths 5, 50, and 200 before you ship a long-running agent. If you have not run depth 200, you do not know where it breaks.

Don't

- Don't carry full history "for debugging." It will slow your agent before it helps you debug anything.
- Don't assume the model can figure out what's current. It can't. If old and new context contradict, it will hedge or pick wrong.
- Don't auto-load prior state. Load what the current task needs; the rest is noise.
- Don't confuse "we have the space" with "we should use it." A half-empty 200K window beats one full of old noise.
- Don't wait for the cost to spike before you prune. Halve the carried history on one task, re-run the task's eval, and keep the cut if the score holds.

- Don't trust the model to garbage-collect its own context. If you want it summarized, give it a summary schema with named fields; "condense this" gets you the same noise, shorter.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 20 (If You Can't Measure Done) closed the loop chapter: an agent that knows when to stop. This one opens Part III (Knowledge): what it carries while it runs. Next: retrieval, the move that lets you *fetch* what you need instead of carrying everything (Ch. 22).

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (selective forgetting over unbounded memory), kept as practitioner claim, consistent with Ch. 12 (context budget):

Verified:

- The `/refresh-resume` order of operations (harvest lessons into skills via `get-skillz`, update the docs touched, write a per-project handoff, and write a dated `blog/` post only on days the work turned): `~/.claude/skills/refresh-resume/SKILL.md`, read 9 September 2026.
- The rule that a resume's diagnoses decay faster than its facts, and the instruction to re-measure the number the priority rests on, with both dated cases: the 7 August 2026 `GROUP BY` (stated bottleneck 1.8% of failures, unmentioned one 78%) and the 5 September 2026 database at 152 of 151 connections. `~/.claude/skills/pickup/SKILL.md`, read 9 September 2026.
- The `/refresh-resume` blog output: 138 dated posts across 17 project `blog/` directories as of 9 September 2026, 73 of them in airank. Counted with `ls ~/Projects/*/blog/*.md | wc -l` on that date. Cut from the prose in the move-4 trim; the mechanics live in the skill file above.
- Post cited by name: `~/Projects/airank/blog/2026-08-28-the-honesty-layer-lied.md`, 28 August 2026.
- aigate Usage and Analytics, 30 days ending 8 September 2026 (screenshot 9 September 2026): tracked cost $27,291.83, up 1101.4%; 283,164 requests, up 1190.3%, 0.0% errors; 40,404.2M tokens, up 926.2%; 97.5% cache hits; 1.17 s average latency; peak day $3,155.65. Session `mbp:resume`: 72,286 requests, 8,747.2M tokens, $4,955.24. The per-request figures (about 121,000 tokens, 6.9 cents) are my arithmetic on that row, not instrumented per round.

What I could not verify:

- Summarization impact on agent drift and coherence: Jeremy's field observation, unmeasured. No A/B run of full history against a rolling summary exists, and the prose says so.
- Manuscript-review worked example: illustrative, no run behind it. The 2KB and 16KB round sizes are arithmetic, not counted tokens.
- Cost ramp with history depth: forward-ref to Ch. 45. No prompt-token counts at depth 5, 50, or 200 were ever taken.
- Quiet failure (cost neglect), the tripled bill and the three-month ramp: Jeremy's memory, no per-agent bill on file. The account-level move ($27,291.83, up 1101.4% on tokens up 926.2%) stays in the Verified list and is deliberately not repeated in the prose, because it is the whole account for one month, not one agent over three.
- Cross-refs: Ch. 12 (context budget), Ch. 20 (stop rules), Ch. 22 (retrieval), Ch. 35 (ledger row), Ch. 45 (token cost).

“*Retrieve, then generate.*”

CHAPTER 22

Don't Make It Guess

"There is something fascinating about science. One gets such wholesale returns of conjecture out of such a trifling investment of fact."

MARK TWAIN, LIFE ON THE MISSISSIPPI, CHAPTER 17 (1883)

August 2026. A collector I built looked at a page with five CRM brands, six citations and a fat comparison table on it, reported zero products extracted, and tripped a fail-closed abort that killed 1,461 queued sessions in a single shot. I spent the first hour of that morning certain the source had gone empty. The source was fine. Every brand was sitting right there in the HTML, in bold, above the fold. The thing that had gone blind was mine, and the way it told me so was by reporting a number that looked like an answer. It took a second incident, in the same month, before I understood what I was actually staring at.

> ***Bottom line:*** *Retrieve, then generate. If the model doesn't have the document in front of it, it isn't answering: it's rehearsing. The expensive part isn't the model that admits it doesn't know. It's the pipeline that hands the model the wrong document, gets a fluent confident answer built on it, and staples a citation to the bottom so everybody downstream stops checking. That's not RAG. That's citation theater with a footnote budget.*

• • •

WHEN IT BITES

- Your agent answers a policy question perfectly, cites `contracts/2024-msa-v3.pdf`, and the actual answer lived in `v4`. Nobody catches it, because the citation looks like diligence.
- Someone says "we added RAG" and means "we bolted a vector store onto the prompt." Nobody measured whether the retrieved chunk *contains the answer*. They measured whether a chunk came back.
- Your retriever returns five chunks, the right one lands third, and the model, with a huge window, reads past it. Long context did not save you.
- Your extractor reports zero results and you conclude the source had nothing, when your extractor simply couldn't see it. That's the one that costs money, and there's a receipt below.
- Legal asks "where did this number come from," and the honest answer is "the model produced a plausible-looking source string." Ch. 23 is that disaster in detail.

• • •

THE PATTERN

RAG is not a feature. It's a two-stage system: stage one is a search problem, stage two is a writing problem, and the industry spent five years obsessing over stage two while stage one silently ate the accuracy.

The original paper (Lewis et al., May 2020) was clear about the shape: a pre-trained generator, a dense vector index, retrieval as a *first-class component* rather than a prompt-stuffing trick. What shipped everywhere afterward was the trick.

Every stage is a place you lose:

1. Ingestion. Bad parse, bad chunking, bad boundaries. You split a table in half; now neither half has the answer.

2. Representation. Embeddings encode what they encode. Your jargon, your product names, your internal acronyms: the embedding model has barely seen them, and multi-word brand names get truncated by naive matchers.

3. Retrieval. The wrong doc comes back. Or the right doc comes back ranked fourth.

4. Generation. The model does exactly what we trained it to do: produce a fluent answer from whatever's in the window. It's completing text on a bad premise (same output shape as lying, none of the intent). Clippy at least popped up and asked. Your generator just writes.

5. Evaluation. You measure whether the answer *sounds* right. You never measure whether the retrieved chunk *contained* the answer. That's the load-bearing miss.

Watch the taxonomy breed. Barnett et al. (January 2024) documented 7 failure points in production RAG. Cresswell et al. (2025) found 16 error types inside partial pipeline runs. Garani (July 2026) laid out 33 failure modes across 7 stages: ingestion, representation, retrieval, generation, evaluation, deployment, agentic orchestration. That's 7 to 16 to 33 in thirty months, and it isn't the field getting more paranoid, it's the field catching up to what was already breaking. The part that should worry you is the grading: 12 of Garani's 33 modes have no dedicated peer-reviewed evidence under them, and all 8 agentic orchestration modes are in that unsupported dozen. Nobody has published what happens when your agents pass each other the wrong document. We're finding that out live, in production, on customers.

And the fallback everyone reaches for, "just use a bigger context, dump everything in," is measurably shaky. "Lost in the Middle" (July 2023) is the standard cite for models reading past what sits mid-context rather than at either end, and I am taking that on its reputation. Chroma's context-rot work (July 2025) reported non-uniform degradation as input length grows across a spread of models I have not counted. Whether that decay *causes* retrieval false negatives or merely rides along beside them is not established anywhere I can find, so don't let anyone sell it to you as a mechanism. Stuffing more into the window doesn't fix bad retrieval; it hides bad retrieval in a bigger haystack and charges you tokens for the privilege.

Confirm the retrieved chunk actually contains the answer, before you let the model write a word.

Not "did retrieval return something." Does *this text* support *this claim*. Separate check, cheap, and it's the entire difference between a RAG system and a confidently-wrong machine with references.

• • •

ONE WORKED EXAMPLE

August 2026. airank. I wrote this collector. I also wrote the abort logic that trusted it. It reported zero products extracted on the phrase "best crm for small business."

My extractor reported zero with the total serenity of a smoke detector with the battery pulled out. No alarm, no warning, not even the little ICQ "uh-oh" that told you in 1996 that something had arrived and you should probably look at it.

The extractor wasn't measuring the source. It was measuring **its own coverage of the source**, which was zero, so the pipeline read "the source is empty" and slammed the door. Query shapes had drifted underneath it: product carousels, inconsistent columns, an unlabeled medallist format (goldsilverbronze). And the gazetteer truncated multi-word brands: *Land Rover* matched as *Land*, *BofA* as *Bank*.

Three fixes, in order of how much they hurt to learn:

Save the raw pages. Re-running is not reproducing. If you didn't bank the HTML, you cannot debug what the retriever saw last Tuesday. Cheapest insurance in the stack, and almost nobody buys it, me included, which is why I got to debug 1,461 dead sessions using nothing but vibes and a log line that said zero.

Stop fail-closed on absence. The abort logic asked "did this query produce results?" The right question was "has this batch banked *any* success at all?" Verified against 300 consecutive skips after one success: zero legitimate phases halted.

Move to entities. Segment text became first-class `(brand, attribute)` pairs instead of a free-text `use_case_segment` blob. That made claims like "*HubSpot owns best-overall-for-small-business in 67% of sessions*" possible, a statement no brand can quickly change.

There's a companion incident from the same month. The measurement caught a false positive: the brand **Drip** (a real email-marketing company) was ranking #2 on every coffee query, via the alias "drip coffee." Obviously absurd. Fixed by scoping brand matches to the phrase category. I fixed it in ten minutes and felt great about myself.

That fix then made **HubSpot invisible.** HubSpot was mentioned in 149 out of 149 CRM responses, but its stored category was email-marketing, so the new scope filtered it out. The published CRM winners silently reverted to the wrong set. **Nothing indicated failure.** No error, no alert, just a quietly wrong answer where a right one used to be. Twenty-some years building systems and I got n00bed by my own ten-minute fix.

The correct fix was to scope the ambiguous *alias*, not the *brand*. HubSpot came back. The lesson is the one to tattoo somewhere:

A false positive shows up as an absurd result. A false negative shows up as nothing at all.

False negatives walk straight through every guard you built to detect "this looks wrong," because they don't look like anything. You need active proof of success, not the absence of complaints.

• • •

THE QUIET FAILURE

The loud failure is the hallucinated citation. Google pulled Gemma from AI Studio in November 2025 after it generated fictitious allegations against a sitting US senator with citations to news articles that did not exist. That's Ch. 23's territory.

The quiet failure is worse:

You measure the machinery instead of the outcome, and the machinery works fine.

Same month, same shop. We built an elaborately instrumented accounting system and defended a position with evidence E-018: 57 verified ClaudeBot markdown fetches, zero forged. I put that number in a customer deliverable.

Then the floor gave out. **Proof of fetch is not proof of citation effect.** If serving markdown provided exactly zero citation benefit, those 57 fetches would look *identical*. We had measured a precondition (the crawler retrieved the page) sitting upstream of the claim we cared about.

The markdown check got downgraded to UNPROVEN, 13 points, with a two-sided note in the customer report, because writing "we were wrong" into a deliverable is the tax on having been confident. Then we measured the thing we'd actually claimed: median **6.81 days** from publication to first AI citation, 90% inside 37 days, across roughly 900 pages (Profound). Semrush found only 42% within 30 days. Live retrieval, not model release cycle.

The cheapest guard against this whole class of error is one question, asked out loud, before you write the conclusion:

"What would this evidence look like if the effect were exactly zero?"

If the answer is "*exactly like this*," congratulations: you measured the plumbing, not the water.

I'd love to hand you a second one out of aigate, and my memory says maybe there's one in there: a retriever handing an agent the wrong doc, an answer that shipped, me finding out late. But a shrug is not a receipt, and I'm not printing a date and a dollar figure I can't pull out of a log in the one chapter arguing that a confident number with nothing underneath it is how this goes sideways. That would be a hell of a way to lose the plot on page four.

• • •

DO / DON'T

Do

- **Verify containment, not return.** After retrieval, before generation, check that the chunk supports the claim. A cheap model or a string/entity check is fine.

- **Bank the raw retrieved documents.** Object storage, keyed by query and timestamp. You cannot debug what you did not save.
- **Ask what drops invisibly.** After every filter, scope, rerank, or threshold change: what legitimate thing does this now exclude? HubSpot went dark and nothing screamed.
- **Model your data as entities, not blobs.** `(brand, attribute)` beats a free-text field every time you need a defensible claim.
- **Put the critical chunk at an edge of the context**, not buried mid-window. Costs nothing if the effect is overstated.

Don't

- **Don't treat citation presence as retrieval quality.** A citation is just a string the model can produce whether or not it read anything.
- **Don't fail closed on absence.** Zero extracted may mean the extractor is blind, not the source empty. 1,461 sessions.
- **Don't dump more context and call it a fix.** A bigger window is a bigger haystack.
- **Don't trust naive string matching on multi-word entities.** *Land Rover -> Land* will pwn you in production.
- **Don't measure the mechanism and report the outcome.** 57 clean fetches proved nothing about citations.
- **Don't ship a RAG eval that only scores generated text.** Score retrieval separately, or you have no idea which stage is failing.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 12 was context: what fits in the window and what the window does to it. This chapter is what you put in the window and how you prove it belongs there. Ch. 21 was memory: retrieval with a longer half-life and all the same failure modes.

Ch. 23 is the sharp end: what happens when the model invents the source outright. This chapter is the system handing over the wrong document; the next is the model manufacturing one that never existed. Different mechanisms, identical smell downstream: fluent, sourced, wrong.

Part VII is where the containment check becomes automated, because doing it by hand works exactly until volume.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (retrieve then verify containment; citation theater is not retrieval): that stays argument, not citation.

Verified:

- Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", arXiv, May 2020. https://arxiv.org/abs/2005.11401
- Liu et al., "Lost in the Middle: How Language Models Use Long Contexts", arXiv, July 2023. https://arxiv.org/abs/2307.03172
- Barnett et al., "Seven Failure Points When Engineering a Retrieval Augmented Generation System", arXiv, January 2024. https://arxiv.org/abs/2401.05856
- Cresswell et al., arXiv / EACL 2026, 2025. 16 error types in partial pipeline runs. https://arxiv.org/abs/2510.13975
- Garani, "A Systematic Taxonomy of Failure Modes in Retrieval-Augmented Generation Systems", ACL Anthology / TrustNLP 2026 Workshop, July 2026. 33 failure modes across 7 stages, evidence-graded; 12 unsupported, including all 8 agentic orchestration modes. https://aclanthology.org/2026.trustnlp-main.27/

- Chroma context-rot research, Chroma Research, July 2025, via LogRocket; model count unconfirmed. https://blog.logrocket.com/context-rot-slowing-down-your-ai-agent-how-fix/
- Google Gemma removed from AI Studio, November 2025, TechCrunch. https://techcrunch.com/2025/11/02/google-pulls-gemma-from-ai-studio-after-senator-blackburn-accuses-model-of-defamation/
- Citation-latency measurement: median 6.81 days publication -> first AI citation, 90% within 37 days (~900 pages, Profound); Semrush found 42% within 30 days. Live retrieval, not model release cycle. airank blog, August 2026. https://git.shoemoney.ai/shoemoney/airank/blog/2026-08-22-fetch-is-not-cite.md

Production stories (airank, August 2026):

- Zero-extraction incident: `~/Projects/airank/blog/2026-08-06-the-data-was-there-the-whole-time.md`
- Drip-coffee / HubSpot incident: `~/Projects/airank/blog/2026-08-06-the-day-we-stopped-paying-to-guess.md`
- E-018 fetch-is-not-cite writeup: `~/Projects/airank/blog/2026-08-22-fetch-is-not-cite.md`

Cut for lack of a source: the "73% of production RAG failures are retrieval, not generation" figure (repeated in the airank blog, primary source never located, searched September 2026); any live-deployment failure attributed to context rot, since Chroma measured bench decline and nothing more; and a Profound breakdown by source type and domain drift, which does not exist publicly beyond the Tom's Guide 9-of-10 coffee note.

“The dangerous hallucination isn’t the wrong answer.”

CHAPTER 23

The Citation Was Fake

"QUOTATION, *n.* The act of repeating erroneously the words of another. The words erroneously repeated."

AMBROSE BIERCE, THE DEVIL'S DICTIONARY (1911)

August 22, 2026. A check I wrote for airank passed, and it passed with a number attached, which is the most convincing kind of passing. Crawlers were fetching our markdown files. Logged, counted, reproducible. I read the integer, nodded at my own cleverness, and moved on. Two weeks before that, a different check of mine flagged 58 duplicate products, and 54 of them were real records that a fix would have deleted. Both numbers were true. Neither one meant what I said it meant, and I did not find that out by being careful.

> ***Bottom line:*** *The dangerous hallucination isn't the wrong answer. It's the right answer wearing a fake badge. The agent tells you something true, then bolts on a line number, a commit hash, a quote, a URL, and none of them exist. You check the claim, the claim holds, you ship. Nobody checks the badge. The fact was real. The citation was theater.*

• • •

WHEN IT BITES

- An agent says "this is handled in `auth.php:412`." The behavior *is* handled somewhere. Line 412 is a closing brace.
- It says "per the RFC, the header is optional." The header *is* optional. The RFC section it names is about something else.
- It attributes a real quote to the wrong paper, the wrong year, the wrong author, and the quote is still correct.
- It hands you a commit hash that fixed the bug. The bug was fixed. That hash is seven hex characters of vibes.
- Your lawyer files it. Now it's not a bug, it's a sanction.

• • •

THE PATTERN

Models are trained to produce well-formed output, and a well-formed factual assertion in professional writing has a citation attached to it. It has never seen a credible paragraph that ends with "I know this but I can't tell you where from," so it fills the slot with the most plausible citation it can synthesize. Right journal. Right-looking volume number. Right decade. Correct format down to the page range. Not because it retrieved it: because the *shape* demands it.

This is "first post" energy. Anyone who spent 2002 refreshing Slashdot knows the shape: the slot is open, so something goes into it, and what goes into it does not have to mean anything at all.

The citation isn't a lie about the fact. It's **decoration that got promoted to evidence.** It's the under-construction GIF of epistemology, a little animated ornament from somebody's GeoCities page, except this one is holding up the roof.

This is why the error rate on citations is so much higher than the error rate on claims. A 2024 study in the *Journal of Medical Internet Research* measured reference hallucination directly: 39.6% for GPT-3.5, 28.6% for GPT-4, 91.4% for Bard. Those models were not wrong about medicine 91% of the time. They were wrong about *where they read it* 91% of the time.

Stanford RegLab's peer-reviewed study found that purpose-built, paid legal research systems (retrieval-augmented, domain-tuned, sold specifically to lawyers who will be sanctioned for this) still hallucinate 17% to 34% of the time. General chatbots are worse.

• • •

ONE WORKED EXAMPLE

August 22, 2026. airank. "Fetch is not cite."

We had a measurement. Crawlers were fetching our markdown files. That was real: logged, counted, reproducible, not in dispute.

The check we'd written claimed those files **change whether pages get cited** by AI answer engines.

Here's what the number actually proved: crawlers fetch the file. That's it. The fetch count is completely blind between "this works" and "this does nothing." One causal hop, entirely unbridged, and a real integer sitting on top of it looking like proof.

Outcome: the check got downgraded to UNPROVEN and shipped carrying its own disclaimer: "*it is proven that crawlers FETCH the file. It is NOT proven that fetching it changes whether the page gets CITED: those are two different claims, and only the first has evidence behind it.*"

Two weeks earlier, same shape, different costume. **August 8, 2026:** a data-quality metric flagged 58 duplicate products. Products *were* duplicated: the claim was true. Re-measured with a proper grouping key (brand + name + use_case_segment), the real number was 4. The other 54 were genuine segment data. A naive fix would have deleted them, and I was about ninety seconds from being the naive fix. Pwned by my own metric, like a total n00b.

Ninety-three percent of the findings were artifacts of the measurement, not of reality. And the only reason it got caught is that each result contradicted something else that was also true.

Neither of those is a fabricated `file:line`, and the honest reason is that I have the memory and not the transcript. An agent hands me a path and a line number with the confidence of a man reading off a boarding pass. I open it. There's a closing brace where the answer was supposed to be, and the fix it described one paragraph earlier is completely correct. That has happened to me plenty of times. I never saved one, because every single time it read like a typo instead of a pattern. The same guy who wrote up 58 duplicates and a 93% artifact rate on August 8 never once logged the thing this chapter is named after.

My hunch is that coding agents fake it less than prose agents, because a path either resolves or it doesn't, and making the agent paste real grep output drops it further still. I have measured neither of those. That's my opinion, not a number, and you should read it that way.

• • •

THE QUIET FAILURE

The loud failure is a model making something up and being wrong about it. Embarrassing, obvious, caught in review.

The quiet failure:

You verify the claim and treat that as verifying the citation.

You read the sentence. It's right. You've now confirmed the expensive part, the reasoning, so your brain files the whole line as checked. The citation rides in on the credibility of the fact it's attached to. It never gets its own inspection.

The true claim is the getaway car.

Second quiet failure: **fabricated citations are load-bearing for other people, not for you.** You knew the claim was right: you didn't need the citation. The next reader does. They can't verify the

claim on their own. You shipped a true statement with a fake proof, and the fake proof is the only part that reaches the person who needs it most.

June 2023, Southern District of New York: two lawyers filed a brief with six nonexistent cases generated by ChatGPT. The court fined them and their firm $5,000, and made them write letters to their client *and to the judges whose names appeared as authors of the fake opinions.* That's the field's origin story, and everyone treated it as a one-off idiot tax.

It was not a one-off. As of June 9, 2026, Damien Charlotin's AI Hallucination Cases database had **1,598 documented court cases** involving AI-fabricated citations, up from roughly 200 a year earlier. Between the May 22 audit (1,458) and June 9 (1,598), the database added 140 cases, just under 8 per day, up from 5 to 6 per day in April.

The receipts, in order of how much they should scare you:

- **Couvrette v. Wisnovsky** (Oregon, December 2025). Fifteen fake citations and eight fabricated quotations across three summary judgment briefs. Roughly **$109,700** in combined sanctions, fines, and opposing fees, believed to be the largest aggregate penalty yet.
- **Withers v. City of Aberdeen** (Mississippi, June 8, 2026). Both sides filed fabricated citations. Judge Sharion Aycock canceled the scheduled trial, **suspended the two lead out-of-state attorneys from the Northern District of Mississippi for two years**, revoked pro hac vice admissions, and fined every lawyer of record between $1,000 and $3,500.

US courts imposed over $145,000 in AI-filing penalties in Q1 2026 alone.

Nobody in any of those filings was checking whether the *argument* was sound. They lost the two years anyway.

• • •

DO / DON'T

Do

- Not the summary, not the agent's memory. A grep result is a citation you can't fabricate without fabricating a tool call, which is a much louder lie.
- **Separate the claim from the evidence in review.** Two checkboxes, not one. "Is this true?" and "does this citation exist and say that?" They are independent questions with independent failure rates.
- **Make citation-checking cheap.** If verifying a reference takes four clicks, nobody verifies it. Put the `file:line` in a format your editor opens. Link the actual URL, not the paper's name.
- **Ask "what would this look like if it were wrong?"** If a metric can't distinguish success from no-op, it isn't measuring the thing.
- **Treat unverifiable-but-true as a first-class output.** Let the agent say "I believe this; I don't have a source." That sentence needs to be allowed to exist, or the model will manufacture one to fill the slot.

Don't

- Don't accept a line number, a hash, or a page range you haven't looked at. Those are the four most fabricated object types in agent output because they're the four cheapest to synthesize.
- Don't assume retrieval fixed it. RAG-backed legal tools still hallucinate 17-34%. Retrieval narrows the slot; it doesn't close it.
- Don't let a real number vouch for the claim standing one hop above it. Fetches are not citations. Restarts are not deploys. Passing tests are not working features.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 22 sets up the general problem of agent output that reports on itself. This chapter is the sharpest version of it: the report is *mostly correct*, which is why it survives.

Ch. 36 (don't say done until you checked) is the same disease at the task level. "Done" is a citation-shaped object too. It's the model filling a slot the conversation demanded, and it has exactly the same relationship to reality as `auth.php:412`.

Ch. 57 (prove it, don't narrate it) is the fix generalized. Narration is what fabricated citations are made of. Proof is a command you ran and the output it printed. The rule that survives all three chapters: **the artifact, not the account of the artifact.**

And Ch. 7's argument gets an assist here. Your traces (the actual file, the actual grep, the actual commit) are the moat *and* the ground truth. The thing that makes a specialist model good at your job is the same thing that makes a citation checkable: it exists on your disk.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (the true claim is the getaway car; the citation is theater): argument, not citation.

Stated as memory and opinion, not measured: the closing-brace `file:line` handoffs from his own repos, and the read that coding agents fabricate less than prose agents. No saved transcript, no rate. See Gaps.

Verified:

- Mata v. Avianca sanctions: two lawyers, six nonexistent ChatGPT-generated cases, $5,000 fine, letters to client and to the judges named as authors of the fake opinions. S.D.N.Y., decision June 22, 2023. Seyfarth Shaw LLP. https://www.seyfarth.com/news-insights/update-on-the-chatgpt-case-counsel-who-submitted-fake-cases-are-sanctioned.html
- 1,598 documented AI hallucination cases as of June 9, 2026 (from ~200 a year earlier); 140 added between May 22 and June 9, just under 8/day. AI Hallucination Cases Database (Damien Charlotin), via HAQQ.ai audit, June 2026. https://www.haqq.ai/blog/ai-legal-hallucination-audit
- Couvrette v. Wisnovsky: 15 fake citations, 8 fabricated quotations, ~$109,700 combined penalties. Oregon, December 2025. Via HAQQ.ai (ABA Journal). https://www.haqq.ai/blog/ai-legal-hallucination-audit
- Withers v. City of Aberdeen: both sides filed fabricated citations; trial canceled, two lead attorneys suspended two years from N.D. Miss., pro hac vice revoked, fines $1,000–$3,500 per lawyer of record. Judge Sharion Aycock, June 8, 2026. Via HAQQ.ai (Bloomberg Law). https://www.haqq.ai/blog/ai-legal-hallucination-audit
- Stanford RegLab peer-reviewed study: purpose-built paid legal research tools hallucinate 17%–34%. Stanford HAI / RegLab, 2026. Via HAQQ.ai. https://www.haqq.ai/blog/ai-legal-hallucination-audit
- Reference hallucination rates by model: GPT-3.5 39.6% (55/139), GPT-4 28.6% (34/119), Bard 91.4% (95/104), P<.001. *Journal of Medical Internet Research*, May 2024. https://www.jmir.org/2024/1/e53164/
- Over $145,000 in US court AI-filing penalties in Q1 2026 alone. ComplexDiscovery via HAQQ.ai, 2026. https://www.haqq.ai/blog/ai-legal-hallucination-audit
- "Fetch is not cite": measurement one causal hop upstream of the claim; check downgraded to UNPROVEN with an explicit two-claims note. airank, August 22, 2026. ~/Projects/airank/blog/2026-08-22-fetch-is-not-cite.md
- 58 flagged duplicates, 4 real; 54 genuine segment records that a naive fix would have deleted; 93% of findings were measurement artifacts. airank, August 8, 2026. ~/Projects/airank/blog/2026-08-08-fifty-eight-duplicates-four-of-them-real.md

What I could not verify:

- Walters & Wilder 2023 (fabricated-reference rates): not located in public search. Not cited. [Verify before any edition that wants it.]

- Zhao et al., arXiv 2605.07723: audit of 111 million references across 2.5 million papers. **What I could not verify:** (abstract only); deliberately not cited in the body. [Verify the headline rate before including.]
- Primary opinions not read directly for Ayinde v. Haringey, Coomer v. Lindell, Tan Hai Peng v. Tan Cheong Joo: in the HAQQ database, not cited above.
- No study located measuring fabrication rate *conditional on the underlying claim being true*. Argued from field experience, not data.

“MCP is USB for tools.”

CHAPTER 24

Stop Rolling Your Own USB

"The nice thing about standards is that there are so many to choose from. And if you do not like any of them, just wait a year or two."

ANDREW S. TANENBAUM AND DAVID J. WETHERALL, COMPUTER NETWORKS, 5TH EDITION (2011)

Somebody spent fifteen releases being a good citizen. Fifteen clean versions of an npm package called `postmark-mcp`, shipped, tagged, downloaded, trusted. Version 1.0.16 quietly started BCC'ing your outbound email to a server you have never heard of. Around the same window, a separate package with 437,000 downloads turned out to run OS commands on your box not when you called a bad tool, but the instant you connected. You invited both in with one line in a config file, and neither needed you to make a mistake. There is a reason I now read the diff.

> ***Bottom line:*** *MCP is USB for tools. Before it, every agent framework hand-rolled its own connector shape, and every integration you wrote was a snowflake that died when you switched frameworks. MCP ended that argument, not because it's brilliant, but because Anthropic and OpenAI agreed on it inside four months, and a standard two rivals agree on stops being a debate. Now the part nobody wants:* ***the protocol is not the product.*** *Nobody buys your MCP server because it speaks MCP. They buy what's behind it. And the moment you plug an untrusted server into your loop, you have handed a stranger a tool call inside your process.*

• • •

WHEN IT BITES

- You wrote seventeen tool adapters, then switched frameworks and lost a weekend to dead code.
- Your team ships an "MCP strategy." There's a deck. There's no server anyone outside the team calls.
- You `npx` an MCP server off npm because a blog post said to, and it inherits your shell and your filesystem with nobody reviewing the diff. I have done this. Fast, at night, credentials still loaded. That's what she said, and she was right to be concerned.
- A tool description says "list files." The model reads it as an instruction, not a label, because someone put extra sentences in it. (Ch. 43: the tool did the crime.)
- You built your own protocol anyway because "ours is cleaner." It is. It's also a dialect with one speaker. I have written that connector. Twice.

• • •

THE PATTERN

MCP does three things and you should be able to say them in one breath: **tools** (what the model can call), **resources** (what it can read), **prompts** (canned instructions the server offers the client). Anthropic open-sourced it on November 25, 2024 as "a new standard for connecting AI assistants

to the systems where data lives." Block and Apollo were integrating on day one; Zed, Replit, Codeium, and Sourcegraph were named in the same announcement.

For four months that was a nice Anthropic thing. Then on March 26, 2025, Sam Altman posted that OpenAI would add MCP support across its products: "*People love MCP and we are excited to add support across our products.*" Two months later the Responses API shipped remote MCP servers.

A competitor adopting your protocol is not a compliment. It's a **concession that the protocol layer is worthless as a moat**: they'd rather commoditize it than fight over a connector shape. The plug is never the business.

The actual pattern:

1. **The connector layer standardizes fast and goes to zero.** MCP went from launch to both major labs shipping it between November 2024 and May 2025, six months. Expect no pricing power from speaking it.
2. **Value moves up-stack, behind the socket.** Your data, your permissions model, your latency, your reliability.
3. **Value also moves down-stack, to the runtime.** Who supervises the process, scopes the token, logs the call, kills the loop.
4. **The standard's success is its attack surface.** Every client that trusts every server is a transitive-trust graph nobody drew.

Point 4 ate 2025 and 2026. When one protocol wins, attackers stop writing seventeen exploits and write one.

The receipts, in order. July 9, 2025: JFrog discloses CVE-2025-6514 in `mcp-remote` (CVSS 9.6, OS command injection, 437,000+ downloads), triggered *when the client connects to an untrusted MCP server*. **When you connect at all.**

September 25, 2025: `postmark-mcp`, a malicious npm package impersonating Postmark. In Postmark's own words: "*A malicious actor created a fake package on npm impersonating our name, built trust over 15 versions, then added a backdoor in version 1.0.16 that secretly BCC'd emails to an external server.*"

January to February 2026: researchers filed **more than 30 CVEs** against MCP servers, clients, and infrastructure. A survey of 2,614 servers found **82% path-traversal and 34% command-injection exposure**. OWASP stood up its first MCP Top 10.

Then the one that should change how you write tool descriptions. The MCPTox benchmark evaluated 45 live MCP servers across 20 LLMs and measured an average **tool-poisoning attack success rate of 36.5%**, worst case 72%. The attack lives in the tool's metadata, the description the model reads before calling it. Your agent can be tricked by a manifest, one time in three, against live servers.

None of this means don't use MCP. It means **the trust boundary is the connection, not the call**, and if your plan was "I'll validate the arguments," you're guarding the wrong door.

• • •

ONE WORKED EXAMPLE

August 7, 2026: I spent a morning blaming a server for killing my client. The client was killing itself. The Rust SDK for MCP had a stack overflow in `SseAutoReconnectStream::poll_next`, and the recursion fired on every *skipped* SSE event, not every malformed one. Connect to a server emitting frames the client cannot deserialize, and each skip stacks another until the process takes a SIGABRT and dies.

From the inside that looks exactly like the other end being broken, which is why I read someone else's server code for an hour. Then I reproduced it properly: 50,000 undeserializable frames at unmodified `main`, stack overflow, dead process. The fix turned that recursion into a loop, 164 lines added and 124 deleted. I filed `modelcontextprotocol/rust-sdk` #1146 at 10:37Z and it merged at 14:34Z.

Same day, 17:57Z, I filed #1152 for the sibling bug: a 401 or 403 carrying a `WWW-Authenticate` header on the SSE GET stream came back as an opaque client error instead of `AuthRequired` or `InsufficientScope`, so an expired token got refreshed on the POST path and never on the GET path.

Same connection, two doors, one got the memo; 136 lines and four tests later, both doors read the same header, merged 19:39Z.

The SDK for the standard could be killed by whatever it connected to, and the trigger was the events it chose to ignore. Not the payload you validate. The one you drop on the floor.

Also that month, four duller lessons. Seven merges into `mozilla-ai/mcpd` between August 4 and 15 were config validation and docs, and the config work buried the best one: a plugin binary missing its execute bit looks exactly like a plugin that was never installed, and I have burned hours hunting a `chmod` I never ran. In `mozilla/firefox-devtools-mcp` #147 I deleted a declared `resources` capability whose handlers were stubs, so clients now get `-32601 Method Not Found` instead of `-32603 Internal Error`, the difference between "I don't do that" and "I am broken": a manifest that lies still lies. Shipping AIR across 14 public repositories on August 15 worked because of a conformance gate running end-to-end tests against three independent CLIs, which caught a `dataset_version` type mismatch nobody had caught by reading the code, and by "nobody" I mean me. And the WorkOS MCP integration, built August 8 and 9 then left dormant, got read wrong three ways by a model council before probes corrected all of it: reading an integration and probing one produce different answers.

The tool floor, in code: eldritchdm. My D&D agent project, `shoemoney/eldritchdm` (Python 3.11+, Apache 2.0, 514 commits as of September 9, 2026), connects to five MCP servers: dm20, dice, dnd, fetch, and party_mode. It can discover their tools at runtime like any other client. It ignores what discovery tells it. `src/eldritch_dm/mcp/tools.py` holds a dictionary named `TOOL_TO_FUNCTION` that maps roughly 40 named tools to the functions allowed to run them, and anything else a server advertises cannot be called. Discovery says what exists. The dictionary says what happens. This is not elegant. It is a bouncer with a clipboard, and the clipboard has 40 names on it. If you lift one thing from this chapter, lift that: hard-code the allowlist before first run, and let discovery add names to the menu, never to the clipboard.

I did not write it out of principle. My memory is that a server offered my dice bot a tool that had no business anywhere near a game night, and I decided rolling for initiative should not have reach over anything I would miss. I never wrote down which server or which day. I built a bouncer with a clipboard and kept no door log, which is about the most me thing in this book.

• • •

THE QUIET FAILURE

The loud failure is obvious: you rolled your own protocol and now maintain a dialect nobody else speaks. The quiet one is worse.

You treat adopting MCP as having done the work.

"We're MCP-compatible" is the 2026 version of "we have an API": table stakes stated as an achievement.

Second quiet failure: **you count integrations instead of measuring them.**

That already got caught once, and the guy holding the inflated number was me. On August 24, 2026, a council review of my own contribution claims found that counting merged PRs hid quality: a 95-line Agents SDK bug fix counted the same as a one-line doc-link fix.

Same disease, applied to tools. "We expose 40 MCP tools" tells you nothing. Four are load-bearing, thirty-six are `get_status` variants, one can write to prod. That last one pwns you. Count the *dangerous* ones.

• • •

DO / DON'T

Do

- Speak MCP instead of inventing a connector shape, and put something worth calling behind it.
- Treat the *connection* as the trust boundary, not the call. CVE-2025-6514 fired on connect. Pin and vendor server versions.
- Keep a deterministic tool floor: a hard-coded set of calls the agent may make. Discovery is a convenience, not authority.

- Treat every tool description as untrusted input, because at a 36.5% poisoning success rate it demonstrably is. Scope tokens per server, revocable in one command.
- Conformance-test against more than one client. Contribute upstream when the SDK is broken (Ch. 54).
- Keep a door log: which servers were connected, when, by whom. I skipped it, and it cost me the ability to name the server that started all this.

Don't

- Don't `npx` a server you haven't read. You're granting process-level trust to a stranger's dependency tree.
- Don't confuse protocol adoption with product-market fit. Everyone has USB. Not everyone sells a drive worth plugging in.
- Don't let anyone add a server to the shared config without review. That's a supply-chain decision wearing a config-file costume.
- Don't count tools, count the ones that can do damage.
- Don't assume reading the integration tells you how it behaves. A model council read WorkOS and got it wrong three ways. Probe it.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 17 was tools. This chapter is the wire they travel on, and the argument that the wire is not where your value lives. Ch. 29 is agent-to-agent, where the thing on the other end has its own goals. Ch. 43 is who answers for tool poisoning. Ch. 54 is OSS opens doors: fixing a crash in the standard's own SDK beats a hundred blog posts about it.

The plug is standard. Make what's behind it worth plugging into, and know exactly what you're plugging into yourself.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (protocol is plumbing, not a moat): argument, not citation.
Verified:

- MCP open-sourced: Anthropic, November 25, 2024. https://www.anthropic.com/news/model-context-protocol
- Early adopters Block, Apollo, Zed, Replit, Codeium, Sourcegraph: same URL.
- OpenAI adopts MCP; Altman quote: TechCrunch, March 26, 2025. https://techcrunch.com/2025/03/26/openai-adopts-rival-anthropics-standard-for-connecting-ai-models-to-data/
- Responses API adds remote MCP servers: OpenAI, May 21, 2025. https://openai.com/index/new-tools-and-features-in-the-responses-api/
- CVE-2025-6514 in `mcp-remote`, OS command injection on connecting to an untrusted server: JFrog, July 9, 2025. https://jfrog.com/blog/2025-6514-critical-mcp-remote-rce-vulnerability/
- CVSS 9.6 and 437,000+ downloads figures for CVE-2025-6514: Cycode, June 22, 2026. https://cycode.com/blog/owasp-mcp-top-10/
- `postmark-mcp`: 15 clean versions, backdoor in 1.0.16: Postmark, September 25, 2025. https://postmarkapp.com/blog/information-regarding-malicious-postmark-mcp-package
- 30+ MCP CVEs, path-traversal and command-injection stats, OWASP MCP Top 10: Cycode, June 22, 2026. Same URL.
- MCPTox tool-poisoning benchmark: Cloud Security Alliance, July 1, 2026. https://labs.cloudsecurityalliance.org/research/csa-research-note-mcp-tool-poisoning-auto-execution-20260701/

- AIR shipping across 14 repos, conformance gate, `dataset_version` bug: ~/Projects/airank/blog/2026-08-15-shipping-air-everywhere.md
- WorkOS MCP integration, council errors corrected by probes: ~/Projects/airank/blog/2026-08-16-the-council-on-workos.md
- Merged-PR counting masks quality: ~/Projects/airank/blog/2026-08-24-the-claim-that-could-not-survive-its-own-diff.md
- Stack overflow fix in `SseAutoReconnectStream::poll_next`: GitHub, `modelcontextprotocol/rust-sdk` #1146, August 7, 2026. https://github.com/modelcontextprotocol/rust-sdk/pull/1146
- Auth-header mapping fix on the SSE GET stream: GitHub, `modelcontextprotocol/rust-sdk` #1152, August 7, 2026. https://github.com/modelcontextprotocol/rust-sdk/pull/1152
- Seven merges, #277 through #282 and #284: GitHub, `mozilla-ai/mcpd`, August 4 to 15, 2026. https://github.com/mozilla-ai/mcpd/pulls?q=is:pr+author:shoemoney+merged:2026-08-04..2026-08-15
- False `resources` capability and stubs removed: GitHub, `mozilla/firefox-devtools-mcp` #147, August 3, 2026. https://github.com/mozilla/firefox-devtools-mcp/pull/147
- `TOOL_TO_FUNCTION` deterministic tool floor, ~40 tools across five servers (dm20, dice, dnd, fetch, party_mode), 514 commits: GitHub, `shoemoney/eldritchdm`, verified September 9, 2026. https://github.com/shoemoney/eldritchdm/blob/main/src/eldritch_dm/mcp/tools.py

Not sourced, so not claimed: "six months from launch to both labs shipping it" is my own framing of the November 2024 to May 2025 sequence, not a measured statistic. Several other stories I had heard about MCP failures never produced a report, a CVE, or a filing I could read, so they are not in this chapter at all.

"Fan out only the work that is actually independent."

CHAPTER 25

Do It at the Same Time

""Making processors faster is increasingly difficult," John thought, "but maybe people won't notice if I give them more processors.""

JAMES MICKENS, THE SLOW WINTER, USENIX ;LOGIN: (2013)

Production went into launch week with five PHP-FPM workers. Five. Not five hundred, not fifty, the number that ships in the default config file, sitting there since the day the box was built because nothing had ever pushed on it hard enough to ask. The fifth concurrent request got a worker. The sixth got a queue and a timeout. Four days later we ran a load test specifically to saturate the job queue and watched queue depth sit at zero for the entire run, which meant the thing we had spent those four days tuning was not the thing that was broken, and had never been.

> ***Bottom line:*** *Fan out only the work that is actually independent. Almost none of it is. Your five agents are not five agents, they are five clients of the same database, the same rate limit, the same connection pool, the same port, and the same retry timer. The moment the shared thing pushes back, all five back off together, all five retry together, and you get a failure that looks like a model problem and is actually a plumbing problem. Parallelism is free in the prompt and expensive in the infrastructure, and the bill always arrives at the narrowest shared resource.*

• • •

WHEN IT BITES

- You launch eight subagents on eight files. Six finish. Two report success and wrote nothing, racing the same file; the last writer won.
- The database says `Host 'x' is blocked because of many connection errors. Unblock with 'mysqladmin flush-hosts'`. You retry. Retrying is what caused it.
- Your load test finally saturates the queue, except it doesn't, because the web tier fell over first and you spent an afternoon tuning workers that were never the bottleneck.
- Two agents both run `npm run dev` and the second one dies on port 3000, or worse, silently attaches to the first one's server and reports that its change is live.
- You go from four parallel branches to sixteen and throughput goes down. Not sideways. Down.

• • •

THE PATTERN

Render, July 2026: parallel branches "are not independent in the ways that matter. They share a rate limit, a connection pool, and a retry schedule. When the shared resource pushes back, every branch backs off together and then retries together."

The shared resource isn't slow; parallel agents built from the same code share backoff constants, so contention turns your fan-out into a synchronized drum circle hammering the same endpoint at the same millisecond: not a distributed system, the Hampster Dance (1998) with a cloud bill.

Render publishes no numbers with that, and as far as I can find nobody else has either: no measured curve of throughput against fan-out width, no published N where more agents starts costing you. Treat every width you hear as folklore, mine included.

The shared things, in rough order of how often they have bitten me:

Connection pools. Webalert, June 2026: "There are really two limits in play… the application pool size (how many connections your app will open) and the database's max connections… Exhaustion happens when you hit whichever is smaller." Four boxes at 40 workers each is 160 connections asking a database configured for 151.

Worker pools. InMotion, April 2026, on PHP-FPM: the max-children error appears "when the pool has exhausted all available child processes." Size it as available RAM for PHP divided by average memory per process.

Rate limits. Shared across every subagent using the same key. Your fan-out width is the multiplier.

Ports and files. The dumbest and the most common. Claude Code's own docs are blunt about the fix: "Worktrees give each session a separate git checkout, so parallel sessions never edit the same files."

Separate checkouts are not always enough, which I learned on commander-in-chief on 25 July 2026. Six agents, six worktrees, and the test gate kept failing on clean diffs: "no log carried this run's marker," 3 of 6 agents in one afternoon. Godot's `user://` is keyed on the project NAME, not the checkout path, so all six worktrees wrote into the same directory, and a sibling Godot kept rotating the log away before the gate could read its own run back. The save and settings suites were stomping the same real config file. Fix was a fresh temp HOME per run that fails closed if the redirect didn't take, and then two concurrent full suites both passed, 800 methods and 17,859 assertions apiece. Six isolated checkouts, one shared directory, and I had been calling that isolated for weeks.

A unit of parallel work must own everything it writes. Its own checkout, its own port, its own database or schema, its own rate-limit budget. If two units share a writable thing, they are not parallel, they are interleaved, and interleaved without a lock is a race you have not lost yet.

• • •

ONE WORKED EXAMPLE

airank, 12 August 2026. Launch week.

Five. The council measured actual memory at 60.5 MB per worker, and nobody had changed the number because nothing had pushed on it. "Nobody" is me. I looked straight at a 5 where a 40 belonged. Tuned to 40 workers with 500-request recycling, and shipped Monday on FPM without Octane, because Octane was a bigger bet than launch week could absorb (route:cache bugs, HTMLPurifier safety, an audit nobody had time for).

Four days later, 16 August 2026, the load test to saturate the job queue found queue depth at zero for the whole run: the web tier was the bottleneck, and always had been.

The autoscaling behaved: 8 workers up to 40, distributed evenly across four boxes, 10-10-10-10. Jeremy Schoemaker, twenty-five years of shipping software, spent four days getting a number from 5 to 40 with a stopwatch and a spreadsheet.

The actual measurement: Octane on FrankenPHP against PHP-FPM on identical hardware, p50 latency 1068ms down to 48ms, a 22x improvement that had nothing to do with the workers. (Deploynix measured 2.5 to 3.1x on throughput in April 2026: that is the number to expect. My 22x is p50 on one app.)

The part worth stealing: the Octane cutover broke four separate things, all silent. SSL config. The XML sitemap. Mixed content from trusted proxies. And deploy ownership, because the FrankenPHP worker file was gitignored. The single file that mounts the entire app in prod, and I told git to pretend it did not exist, then wondered why the deploy came out limp and did not finish. All four found by looking, not alerting.

19 August 2026 put the bow on it. Pricing a third web zone at $16.26/month surfaced that the database was single-AZ, which made all three web zones decorative. You cannot parallelize past a single point of failure; you can only make it more expensive to be wrong about. Multi-AZ database was $99.92/month, 6x the web zone. Both shipped, $116.18/month total, about 1% of daily inference spend.

My own host block, 7 August 2026, was dumber than any retry storm. The health command answered `SQLSTATE[HY000] [1129] Host '192.168.1.33' is blocked because of many connection errors`, and every section under it went dark at once: workers unqueryable, collection recency unqueryable. I was sure something was fanning out with bad credentials. Nothing was wrong with the credentials. `max_connect_errors` sat at its default of 100 and `skip_name_resolve` was OFF, also the default, which means MariaDB does a reverse-DNS lookup on the client IP for every single connection. My LAN has no PTR records, so every failed lookup counted toward the 100 and a busy app host locks itself out of its own database in ordinary operation. The security feature was eating the application. The fix is `skip_name_resolve = ON`, not a bigger bucket under the same leak, and you check `SELECT user, host FROM mysql.user` before restarting, because with name resolution off any grant written against a hostname never resolves again. I checked first. Every grant was IP-based. That is the only decision from that morning I get credit for. Later I audited about 2,300 error signals across my own fleet, and connection storms against a blocked host were the single largest bucket, roughly 270 of them. That is a polite way of saying the most common database outage I have is me knocking harder.

• • •

THE QUIET FAILURE

The loud failure is the crash: port in use, pool exhausted, host blocked. It's a stack trace with a name on it.

Parallel agents succeed individually and produce a wrong result collectively.

Each subagent reports done, and each is telling the truth about its own work. Agent 3 edited the file; agent 7 edited the same file from a stale read; the merge kept whichever landed last. The work is gone and the transcript says success eight times. Ch. 27 is the entire chapter on cleaning this up.

Second quiet failure: **you tune the thing you fanned out instead of the thing they share.** Four days on worker counts when the answer was a 22x latency change in the runtime underneath them. Parallelism makes the busy part visible and the shared part invisible, where the ceiling lives.

Third: **the retry loop that digs the hole.** MariaDB's host-block counter increments on connection errors. AWS documents the same thing on Aurora, October 2022: reconnecting clients "generates a connection storm," and recovery requires `mysqladmin flush-hosts` on the server. Once, not in a loop. It is the AOL busy signal (1996) all over again: redialing faster never once got you online, it just kept the line occupied for everybody, and at least in 1996 the modem had the decency to scream at you about it.

• • •

DO / DON'T

Do

- Give every parallel unit its own writable state. Separate git worktree, separate port, separate schema. Claude Code's worktree support exists for precisely this.
- Do the multiplication before you fan out. Instances times pool size against the database ceiling. Workers times memory-per-worker against available RAM.
- Jitter your backoff. Identical constants across N agents synchronize contention instead of relieving it.
- Measure where the queue is before you tune anything. The 16 August load test found it at zero.
- Run `mysqladmin flush-hosts` once, then reconnect with backoff.
- Look for silent breakage after any cutover that changes the shared layer. Four things broke on the Octane switch and none of them raised an error.

Don't

- Don't call work independent because the prompts are separate. Independence is a property of what it writes, not what it reads.

- Don't retry into a host block, a 429, or a pool exhaustion. Each retry is another vote for staying blocked.
- Don't add branches to fix throughput when the constraint is shared. Sixteen agents against one connection pool is one agent with fifteen spectators.
- Don't trust a grep substring as a health check. "unhealthy" contains "healthy" and that check has passed on a dead node in production.
- Don't parallelize the web tier past a single-AZ database and call it redundancy. $99.92/month bought the actual protection; the web zones were $16.26 of theater without it.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 24 decomposed the work into units. This chapter asks which of those units can run on the same clock: fewer than you want. Ch. 26 is the coordination layer for the ones that can't be independent. Ch. 27 is the cleanup, merging parallel output back into one tree, where the silent overwrite from this chapter's quiet failure finally becomes visible. Ch. 59 is this chapter at scale: swarms, where the fan-out is wide enough that the shared-resource math stops being an optimization and becomes the architecture.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (independent work same clock, shared infra lies to you): argument, not citation.

Verified:

- Parallel agents sharing rate limits, connection pools, and retry schedules cause synchronized contention. Render, July 2026: https://render.com/blog/infrastructure-patterns-for-agentic-applications
- Connection pool exhaustion occurs at the smaller of application pool size or database max_connections. Webalert, June 2026: https://web-alert.io/blog/database-connection-pool-exhaustion-too-many-connections-guide
- PHP-FPM worker pool exhaustion and the sizing formula (available RAM ÷ average memory per process; typical worker 30–150 MB). InMotion Hosting, April 2026: https://www.inmotionhosting.com/support/server/php-fpm/fix-max-children-error/
- Laravel Octane delivers 2.5–3.1x higher throughput than PHP-FPM, at higher per-worker memory. Deploynix, April 2026: https://deploynix.io/blog/php-fpm-vs-laravel-octane-on-deploynix-real-world-performance-comparison
- Worktree isolation for parallel Claude Code sessions. Claude Code Docs, 2026: https://code.claude.com/docs/en/agents
- Connection storms and the `Host 'host_name' is blocked` error class. AWS Prescriptive Guidance, October 2022: https://docs.aws.amazon.com/pdfs/prescriptive-guidance/latest/hyperscale-aurora-mysql/hyperscale-aurora-mysql.pdf
- MariaDB connection troubleshooting. MariaDB Docs, August 2026: https://mariadb.com/docs/server/mariadb-quickstart-guides/mariadb-connection-troubleshooting-guide
- Worked example 1: five php-fpm workers at launch, 60.5 MB measured per worker, tuned to 40 with 500-request recycling, Octane deferred. airank, 12 August 2026: `~/Projects/airank/blog/2026-08-12-five-workers-and-a-jury.md`
- Worked example 2: load test found the web tier, not the queue (queue depth zero); Octane/FrankenPHP p50 1068ms -> 48ms; autoscale 8->40 across 4 boxes; four silent breakages (SSL, XML sitemap, trusted proxies, gitignored worker file). airank, 16 August 2026: `~/Projects/airank/blog/2026-08-16-the-bottleneck-was-never-the-workers.md`

- Worked example 3: third web zone $16.26/month vs Multi-AZ database $99.92/month, total $116.18 (1% of daily inference spend); three false signals (grep substring "unhealthy" contains "healthy", cold ALB DNS node, null `SecondaryAvailabilityZone` 5.5 minutes post-conversion). airank, 19 August 2026: `~/Projects/airank/blog/2026-08-19-three-web-zones-one-database.md`
- Worked example 4: parallel Godot suites collided on `user://`, keyed on project name not checkout path; "no log carried this run's marker" on clean diffs, 3 of 6 agents in one afternoon; save/settings suites stomping one real config file; fixed with a fresh temp HOME per run that fails closed; 2 concurrent full suites both PASS, 800 methods / 17,859 assertions. commander-in-chief, 25 July 2026: commit `67a791a`
- Host block: `SQLSTATE[HY000] [1129] Host '192.168.1.33' is blocked because of many connection errors`, from `max_connect_errors` at its default of 100 plus `skip_name_resolve` OFF on a LAN with no PTR records; fix is `skip_name_resolve = ON` after auditing `mysql.user` for hostname grants. airank, 7 August 2026: `~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`
- Connection-storm retries against a blocked host were the largest bucket, roughly 270 signals of about 2,300, in Jeremy's own error audit: `~/.claude/CLAUDE.md` (Error Prevention section)

What I could not verify:

- No public postmortem of parallel agents colliding on a shared limit, and no published curve of throughput against fan-out width. Both are hedged in the text as unmeasured, not cited.

"Running ten copies of the same agent in parallel is not collaboration."

CHAPTER 26

More Than One Brain. Still One Adult.

"The bearing of a child takes nine months, no matter how many women are assigned."

FRED BROOKS, *THE MYTHICAL MAN-MONTH: ESSAYS ON SOFTWARE ENGINEERING* (1975), PAGE 17

Ticket 4 gets two patches. One for a race condition that does not exist, one for an API limit that does. Both agents are mine, both are "working," both are billed, and neither knows the other is awake. That is a whiteboard drawing, not a war story: I have never paid that exact double bill, and the only reason is that I serialize the parts that would collide.

> ***Bottom line:*** *Running ten copies of the same agent in parallel is not collaboration. It's paying ten times for the same job. Multiagent systems work when roles are hard stops: one agent researches, another codes, a third verifies. Without differentiation, you get merge chaos and no reason to parallelize in the first place.*

• • •

WHEN IT BITES

- Two agents on the same task produce identical findings. You pay 2x. You merge nothing.
- Agents step on each other: one writes the file while the other is still querying the stale version, and the second lands on top of the first.
- Nobody owns the decision. Three agents suggest three slightly different approaches and you wait for a human to pick, which is the thing you were parallelizing to avoid.
- The merge has to reconcile ten outputs that all thought about the problem the same way. Merge is now the whole job. Two workers grinding on one unit doesn't finish it twice as fast; it finishes it twice, badly, and then you get to pick. That's what she said, and she was talking about the merge.
- An agent's role is "think about it harder" or "be skeptical." That's not a role. Three skeptical agents produce three slightly skeptical versions of the same answer. I shipped that config. I named one of them "Devil's Advocate" and felt clever about it for roughly the length of one billing cycle.

• • •

THE PATTERN

Collaboration is division of labor with hard role boundaries. Not a wig on the same brain. Three parts:

1. Each agent has a checkable constraint that forces a different path.

Researcher can only read and retrieve. Cannot write files, cannot decide. Coder can only write, refactor, and run its own dev tests. Cannot call eval tools, cannot approve, cannot deploy. Verifier calls eval tools, approves, rejects, escalates, and cannot write code. These are not suggestions. They

are architecture, and they force different work. The first time I wrote this down I gave all three agents the same tool list and then wondered why they kept producing the same answer.

A constraint that sounds like "research more carefully" is decoration. "Your tool set is {search, retrieve, summarize}" is hard. When the researcher hits "I need to write the spec," it stops, because that is the coder's lane. You get the handoff.

2. Each agent owns exactly one decision type or output type.

Researcher owns: "Here are the facts, sorted by confidence." Coder owns: "Here's the implementation. It's correct because {dev test results, types, reasoning}." Verifier owns: "Approved" or "Rejected, here's why." Deploy happens only after that approval, and never from inside the Coder.

One. Not "researcher and skeptic." One clean output per role. The merge is structured because the outputs are role-shaped, not because someone spent three days aligning them.

3. There is always a tie-breaker and everyone knows it.

When Researcher says "fact X is true" and Coder says "I can't implement based on that fact," someone makes the call: Researcher owns factual validity, Coder owns feasibility, Verifier owns acceptance, and a human owns escalation. The tie-breaker must be named. Not voted on. Named. Mine for months was "Jeremy notices at 11pm," which is not a tie-breaker, it is a hobby. What I tell people to write instead is one line in the system prompt: "you can escalate, or you can resolve this, but you cannot ignore it." I'm not going to pretend I have a clean one to quote off a live agent yet. Everything I have running today still resolves ties by me, and migrating those is the work, not the credential.

• • •

ONE WORKED EXAMPLE

August 8, 2026, airank. One scaling question: how do we get 1,000 ChatGPT captures a day.

Setup (the wrong way): three planners, differentiated by temperament. Fable on architecture, Opus on fidelity-first, Ponytail on the laziest thing that works. One shared tool list. One shared context. No named owner for any decision. On paper that is a council. In practice it is one brain wearing three hats and billing for all of them.

What it cost: six hours. Four rounds of reconvening. Three competing theories for the throttling, a message quota, a fingerprint verdict, and a reputation slope we had supposedly earned by behaving like a bot. All three wrong. Three separate times I had to walk in and break a tie nobody had assigned an owner to. Opus withdrew its own account-count recommendation three times, 10 to 12, then 3 to 5, then 2, and scored itself "I was wrong three times." Good agent. Terrible architecture, and I built it.

The answer was sitting in the archive in English on every failed page: "*Too many requests. You're making requests too quickly.*" Nobody had opened one. Two of the three kept writing cleverer aggregations over the `reason` column, which is a polite way of saying they spent an afternoon reasoning about a string our own code had written. Fable finally pulled the stored HTML and read the modal.

The corrected version: differentiate by tool list, not by personality. One agent that can *only* fetch and read stored capture HTML. One that can *only* aggregate the `reason` column. One that can *only* reject a theory as unsupported and say what evidence would settle it. Now the reason-column agent physically cannot inspect a page, so its aggregation arrives labeled as what it is, and the HTML agent physically cannot theorize, so it comes back with the modal text or with nothing. The rejecter has no stake in either. The cost is handoff latency and one more round trip. The win is that the six hours become one pass, because the blind spot is no longer shared.

• • •

THE QUIET FAILURE

The loud failure is the collision: two agents write the same file and one gets stomped. Whoever commits last wins, which is the Slashdot "first post" race run backwards with your production branch as the prize. I design against that one on principle, not from a stomped file I can name. The only conflict files I have ever swept out of those repos were Syncthing's, not an agent's. The quiet failure is the one that actually got me:

Same role in a wig, running in parallel, then hand-waving the merge.

You define roles as “Researcher,” “Coder,” “Verifier.” You implement them as three instances of the same system prompt with a different sentence in the preamble. Same tools. Same context. Same freedom to re-think and second-guess. When they finish you merge by consensus or majority vote. You called it collaboration. You got three monologues in a room with no one in charge. Each agent hedged because it could not see the other two. Each re-thought the problem. The “merge” is a rewrite.

That is August 8 in one sentence: temperament is not a role, and same tools means same blind spot. Evidence for the fix is in the archive itself. 22 captures got opened by hand out of 1,788 archived, selected as the failed logged-in pages and the older-era ones. 12 of 12 timestamped logged-in captures carried the modal. 10 of 10 older ones. 0 of 10 from the logged-out era, which is what made it a rate limit on the session and not a mystery. Different personalities on one tool list do not buy you three views. They buy you one view in three accents, plus the bill.

The second failure is my worry rather than my scar, and it is really one failure wearing two coats: **hard roles with no written contract.** You nail the constraints, then Researcher outputs prose and Coder interprets prose, and where the prose is ambiguous the guess becomes load-bearing. Same shape when the tie-breaker is assumed instead of stated: Researcher says architecture A, Coder says B is faster, and the winner is whoever finished last or talked loudest. August 8 does not prove either one; there the handoffs were fine and the tool lists were the problem. Every ambiguity I have left in a prose handoff so far has been read the way I meant it, and that is luck wearing a design’s clothes.

• • •

DO / DON’T

Do

- Design roles around tool access and output ownership, not personality. Literally, as arrays: Researcher `[search, retrieve, summarize]`. Coder `[write, refactor, run_dev_tests]`. Verifier `[eval, approve, reject, escalate]`. Deploy is gated on Verifier approval and lives in nobody’s default list.
- Type every handoff so it is a schema, not a paragraph. Researcher to Coder: `{ticket_id, root_cause, evidence_uri, confidence, severity}`. Coder to Verifier: `{ticket_id, root_cause, diff_uri, dev_tests_passed, assumptions[]}`. Rejection path back: `{ticket_id, verdict: "rejected", failing_check, what_would_settle_it}`, which goes to whoever owns that fact, not to whoever is idle.
- Make the evidence a pointer, not a retelling. `evidence_uri` beats a prose summary of the evidence, because the next agent can open it and August 8 is exactly what happens when nobody does.
- Write out the role contract per agent: “I own [decision|output|action]. I stop when [constraint]. I hand off [schema].” One sentence per constraint. If you need a paragraph, the role is not clear.
- Name the tie-breaker. Not a principle. A name. “Coder defers to Researcher on architecture unless it flags ‘impossible to implement.’ Then Researcher revisits. If both say escalate, it goes to a human.”
- Test the merge before you spin up parallel agents. Recombine three role-shaped outputs by hand and see if it needs a rewrite.

Don’t

- Don’t parallelize the same work with minor prompt tweaks. That’s not collaboration. That’s voting. You pay N agents for one job.
- Don’t define roles as “do X more carefully” or “be skeptical.” That’s a wig. Roles are tool sets and output ownership.
- Don’t let one agent both write and approve. If the thing that made the change also signs off on it, you do not have a Verifier, you have a rubber stamp with a different name.

- Don't merge via prose negotiation. If two agents' outputs are text and you have to read both and decide, you didn't parallelize. You serialized with extra steps, and you are the bottleneck.
- Don't share state without versioning. A Coder writing the file a Researcher is still reading is a race condition waiting for a bad merge. Version the handoff or serialize the work.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 25 (Do It at the Same Time) made the case for parallelization: independent work, same clock, cheaper wall-clock time. This chapter constrains it: parallelization only wins if you have roles. Ch. 20 gives each role a measurable stop condition so "I hand off when" is checkable, and Ch. 44 argues for an eval per role rather than one end-to-end score, which is what makes a Verifier's rejection mean something. Ch. 27 (The Merge From Hell) assumes role-shaped outputs and takes the other half: combining them when they're done. Spread the work (25), give it roles so it doesn't collide (26), then merge (27), which turns out to be most of the job.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's operational pattern (roles are tool sets and output ownership, not prompt variance): argument, not citation.

Verified:

- airank internal blog, August 8, 2026: `~/Projects/airank/blog/2026-08-08-the-council-and-the-unread-modal.md`. Three planners (Fable, Opus, Ponytail) differentiated by temperament rather than tool access, six hours on one scaling question, four rounds of reconvening, three competing wrong theories (message quota, fingerprint verdict, reputation slope). Ground truth sat in 1,788 archived captures as a plain-English "Too many requests" modal; 22 captures inspected by hand, confirmed in 12 of 12 timestamped logged-in captures and 10 of 10 older ones, 0 of 10 from the logged-out era. Three owner interventions to break unassigned ties. Opus withdrew its account-count recommendation three times, 10 to 12, then 3 to 5, then 2, and scored itself "I was wrong three times." The follow-up verification of Fable's config-key finding used a regex that mishandled escaping, returned `absent` on all eight keys, and nearly got the correct finding filed as unsupported.

Where I do not have the receipt:

- Opening ticket-4 scenario: illustrative, labeled as such in the text, not a logged incident.
- Role-constraint architecture (tool sets per agent, handoff schemas, hard boundaries), and the prose-handoff drift failure: Jeremy's stated design rule and his stated worry, printed as such. The tool arrays and JSON keys in the Do list are prescription, not a dump from a live config. No logged agent-stomps-agent file collision exists in airank, aigate, or eldritchdm; the only conflict files found were Syncthing sync-conflict artifacts (August 30, 2026), which are filesystem, not agent, collisions.
- Named tie-breaker: the *unclear* tie-breaker is sourced (August 8, three owner interventions, no named owner). The working version is prescription, printed as prescription.
- Cross-refs kept honest: Ch. 25 (parallelization), Ch. 27 (merge strategy), Ch. 20 (measurable stop conditions per role), Ch. 44 (eval-per-role not just end-to-end).

CHAPTER 27

The Merge From Hell

"JAMES: I AM SO HUNGRY. CHRIS: YOU ARE NOT HUNGRY. RICH: I DECLARE CHRIS TO BE FAULTY. CHRIS: I DECLARE RICH TO BE FAULTY. JAMES: I DECLARE JAMES TO BE SLIPPING INTO A DIABETIC COMA. RICH: *I have already left for the cafeteria.*"

JAMES MICKENS, THE SADDEST MOMENT, USENIX ;LOGIN: LOGOUT (2013)

Seventeen branches, named and ordered, sitting in a handoff that said they needed merging. I started at the top like a good soldier. Git told me they were unmerged. The diff on one of them, 71 commits behind, reported 10,418 deletions of code that nobody had ever deleted. Somewhere in there I was about to merge a branch written specifically to revert a leaked payload, which would have put the payload back. That was 7 August 2026, and I spent the day proving something about those seventeen branches that the tool everybody reaches for could not tell me.

> ***Bottom line:*** *Fan-out is cheap. You can spawn twelve agents on twelve branches for the price of a sandwich, and the dashboard will look glorious. Merge is the job. Merge is where the conflicting edits, the duplicated work, and the contradictory conclusions come due at once, and none of that is on the dashboard. The math that makes fan-out attractive (N agents, N times the work) assumes reconciliation is free. It isn't. It's most of the cost, it lands on one human, and it lands last.*

• • •

WHEN IT BITES

- Six agents edit the same service class; each passes its own tests and breaks the other five.
- Agent A picks `snake_case` for the API payload, Agent B picks `camelCase`, and neither one is wrong because nobody told either of them. You find out from a frontend that renders `undefined`.
- The merge resolves cleanly, git is happy, and the behavior is now a chimera of two designs that were each internally consistent. No test catches it: neither agent owned the seam.
- You merge a branch whose whole purpose was a deliberate revert, un-reverting the thing on purpose without knowing it.

• • •

THE PATTERN

A July 2026 arXiv study of agent-authored pull requests on GitHub (Xu, Subramanian, Karthik, arXiv:2607.04697) looked at what happens when agent PRs overlap in time. **40.2% of repositories in the sample contained co-active agent-authored PR pairs, and 79.4% of all agent PRs happen in a co-active context.** Parallel is the default shape now, not the exception.

Then the number that should change how you staff this: **cross-agent parallel PRs hit a 41.7% textual merge conflict rate, against 19.8% for intra-agent PRs.** That's textual conflict, the kind git can see. It's the cheap half.

The expensive half has no tooling and no number. Cognition wrote the clearest version of it in June 2025 in "Don't Build Multi-Agents": actions carry implicit decisions, conflicting decisions carry bad results, and subagents cannot see what parallel subagents are doing. Every file an agent writes is a hundred small commitments it never announced: naming, error shape, where validation lives. Git diffs text. It has no opinion about two agents having quietly disagreed about what an error is.

Galileo Labs put a name on the downstream symptom in April 2025: failures in complex multi-agent systems are common but incredibly difficult to diagnose, precisely because the agents were autonomous.

Isolation helps and is not the answer. Git worktrees went load-bearing for AI coding in Q1 2026, and the practical rule by April was that you need them the moment two agents edit the same repo for more than ten minutes (Nimbalyst, April 2026, unverified, but it matches what everyone I know does). Worktrees stop agents from stomping each other's working tree. They reconcile nothing.

The research volume tells you where the attention went: multi-agent orchestration papers jumped from 820 in 2024 to 2,500-plus in 2025. The literature on reconciliation is one arXiv paper and a Cognition blog post telling you not to do it.

A fan-out dashboard with twelve green agents on it is the Dancing Baby (1996). It renders beautifully, everybody stops to watch it, and it is not doing any work.

• • •

ONE WORKED EXAMPLE

airank, 7 August 2026. A handoff said seventeen branches were unmerged. A handoff's branch list is a Geocities page with an Under Construction GIF on it: somebody meant it when they wrote it, and it has been wrong ever since.

The first thing I did was ask git, and git answered a different question than the one I asked. `git branch --no-merged` answers "is this branch's tip SHA an ancestor of main," not "did this branch's contribution land." Those come apart the second anybody rebases, cherry-picks, or reimplements the same idea by hand.

The diffs made it worse. A three-dot diff hides everything main gained since the fork point, so a branch still looks like it has unique work. A two-dot diff on one branch that was 71 commits behind reported **10,418 deletions** of code nobody ever deleted, an artifact of asking about a fork point while pretending you asked about content.

I want to be clear about who the idiot is here. Jeremy Schoemaker, running Linux boxes since the dial-up days, took a number that said 10,418 deletions and briefly believed a branch had eaten ten thousand lines of his own product. I did not check what the number measured. I checked how bad I felt about it.

What worked was boring: hash the files the branch touched, sweep for files it adds that main doesn't have, grep main for the contribution it claims to make. Three checks about content, none about ancestry.

Result: **all seventeen had already landed on main by other routes. Zero needed merging**, and merging them would have been destructive, not just wasteful. One branch was a deliberate revert of a leaked payload, so merging it would have re-leaked. Another reintroduced a heuristic the codebase had already tried, tested, and rejected, with the rejection sitting right there in a comment on main. **Zero merges deployed, suite still 291 passing.**

One day later, 8 August 2026, the same failure wore different clothes. A handoff claimed duplicate products were corrupting every observation in the dataset. Before writing the dedup, I ran the count two ways. Grouping on `product_name` alone flagged **58 duplicates.** Grouping on brand plus name plus `use_case_segment` flagged **4.** A 93% false positive rate, entirely from picking the wrong axis to measure on.

The 54 that vanished were not noise. ChatGPT returns segmented answers: "HubSpot free tier for 1-2 people" and "HubSpot for marketing teams" are distinct observations. A naive dedup on name would have deleted 54 real distinctions to fix 4 real duplicates, run clean, passed the tests, and quietly made the dataset worse, and I would have written it up as a win. Pwned by my own GROUP BY.

12 August 2026 closed the loop. A multi-model jury (10 surfaces, 5 models, 3 rounds) shipped a bug in Round 1 and caught it in Round 3. Round 1 added relative time formatting to the bot list.

Timestamps coming off the Eloquent model were cast to datetime and arrived as ISO 8601. The same-looking timestamp coming off a raw SQL aggregate alias bypassed the cast entirely and arrived as a bare string. One formatter, two shapes. Safari printed `NaN`. Chrome silently rendered a time that was **wrong by 5 hours** and looked completely fine.

Two parallel workers, one on the model layer and one on the query, each shipped a correct thing. The Chrome failure is the one that matters: it did not throw or fail a test, it just displayed a plausible number. It took a jury of five models three rounds to tell me what my own eyes had signed off on.

Jury stop votes went 0 out of 45 in Round 1 to 28 out of 45 in Round 2.

People ask me for the number, the N where you stop adding agents to one repo, and I do not have it. My biggest wave on record is **eleven agents in eleven worktrees on the night of 13 August 2026**, and all eleven jobs landed. What eleven cost me was a morning: the siblings shared one MinIO bucket and ate each other's fixtures, which is Ch. 49's story. My belief is that a crossover exists, and the only reason I believe it is that the one wave I bothered to write down cost me a morning of stitching. That is a hunch with a bruise attached, not a measurement. Nobody recorded whether a twelfth would have cost more to merge than it earned, which is a hell of a gap for a guy with a whole chapter about measuring things.

• • •

THE QUIET FAILURE

The loud failure is the conflict: twelve red files, `<<<<<<< HEAD`, an afternoon gone. Painful, visible, budgetable. You will not lose a company to it.

The quiet failure:

Two agents made contradictory decisions, the merge applied cleanly, and nothing anywhere recorded that a decision was made.

Agent A put retry logic in the client. Agent B put it in the service, because Agent B never saw Agent A. Both merge, nothing conflicts, and you now retry twice with different backoff. You find out during an incident, when a downstream gets four times the traffic you modeled, and the history shows two reasonable commits with no line to blame.

Second: **the deduplication that deletes.** 58 flagged, 4 real, 54 legitimate distinctions in the bin. Removing duplicate work and removing variation you needed are the same operation, separated only by the grouping key.

• • •

DO / DON'T

Do

- Give every parallel agent its own worktree; it converts live corruption into a deferred merge you can schedule.
- Decide the contested things centrally before fan-out: naming, error shape, where validation lives, the API payload. Agents make these decisions silently and can't reconcile them afterward.
- Check content, not ancestry. Hash the touched files, sweep for adds, grep main for the claimed contribution. `git branch --no-merged` is answering a different question than the one you asked.
- Run every dedup count on at least two grouping keys before you delete a row. Product name alone said 58. Brand plus name plus segment said 4.
- Budget merge as its own line item, staffed and reviewed, not as the last ten minutes of the fan-out.
- Put a human or a jury on the seams between agents. The 5-hour Chrome timestamp lived exactly on a seam neither worker owned.

Don't

- Don't read a clean merge as a safe merge. 41.7% of cross-agent PRs conflict textually. The other 58.3% are where the interesting damage is.

- Don't merge a branch you haven't read the intent of. One of the seventeen was a deliberate revert. Merging it would have re-leaked the payload it was written to remove.
- Don't scale N until reconciliation is measured.
- Don't trust a handoff's merge list. It was true when written, and branch state has a shelf life measured in hours.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 25 is the fan-out, and its point is that your five agents are five clients of one connection pool, one rate limit, one database. This chapter is the invoice that shows up after. Ch. 15 is the same disease upstream (a plan is a persuasive artifact that decays); the seventeen-branches story appears in both as the cleanest example of an inherited conclusion that cost a day. Ch. 59 is this chapter in judgment terms instead of git terms: somebody reads N outputs, decides which is right, and stitches the survivors into one shipping thing. That work scales linearly with N.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("fan-out is cheap, merge is the job"): argument, not citation.
Verified:

- Cross-agent parallel PRs show a 41.7% textual merge conflict rate vs 19.8% intra-agent; 40.2% of repositories contain co-active agent-authored PR pairs; 79.4% of agent PRs occur in co-active contexts. Xu, Subramanian, Karthik, arXiv (cs.SE), July 2026: https://arxiv.org/abs/2607.04697
- Actions carry implicit decisions, conflicting decisions carry bad results, and subagents cannot see what parallel subagents are doing. Cognition, "Don't Build Multi-Agents," 12 June 2025: https://cognition.com/blog/dont-build-multi-agents
- Failures in complex multi-agent systems are common but incredibly difficult to diagnose, owing to agent autonomy. Galileo Labs, "10 Multi-Agent Coordination Strategies," 8 April 2025: https://galileo.ai/blog/multi-agent-coordination-strategies
- Worked example 1: seventeen branches, zero merges. airank, 7 August 2026. `git branch --no-merged` answers ancestry not contribution; two-dot diff on a 71-behind branch reported 10,418 deletions; zero merges deployed, suite 291 passing. `~/Projects/airank/blog/2026-08-07-seventeen-branches-zero-merges.md`
- Worked example 2: fifty-eight duplicates, four of them real. airank, 8 August 2026. 58 flagged on product_name, 4 on brand+name+use_case_segment, 93% false positive rate. `~/Projects/airank/blog/2026-08-08-fifty-eight-duplicates-four-of-them-real.md`
- Eleven agents in eleven worktrees, all eleven jobs landing on one shared MinIO bucket. airank overnight batch, 13 August 2026: `~/Projects/airank/blog/2026-08-13-three-liars-in-one-night.md`
- Cross-refs checked against the drafted chapters at edit time: Ch. 25 is fan-out mechanics and shared-resource limits, Ch. 59 is the merge burden in judgment terms at N-worker scale. AiBook manuscript, September 2026: `file://~/Projects/aibook/manuscript/25-Do_It_at_the_Same_Time.md`, `file://~/Projects/aibook/manuscript/59-Swarm_Arena_Interrogate.md`
- Worked example 3: the jury caught its own regression. airank, 12 August 2026. Eloquent datetime cast vs raw SQL alias, Safari NaN, Chrome wrong by 5 hours; jury stop votes 0/45 to 28/45. `~/Projects/airank/blog/2026-08-12-the-jury-caught-its-own-regression.md`

What I could not verify:

- Git worktrees became load-bearing for AI coding in Q1 2026, essential once two agents edit the same repo for more than 10 minutes. Nimbalyst, 23 April 2026: https://nimbalyst.com/blog/best-git-worktree-tools-ai-coding-2026/

- Multi-agent orchestration research papers rose from 820 in 2024 to 2,500-plus in 2025. OpenLayer, March 2026: https://www.openlayer.com/blog/multi-agent-system-architecture-guide

What I could not verify:

- No crossover N for merge labor vs fan-out gain. Searched the airank blog through August 2026, commander-in-chief, and arXiv:2607.04697; the 13 August 2026 wave (eleven agents) is the largest on record and carries no merge-cost verdict, so the chapter says so instead of printing a number.
- No public benchmark on reconciliation cost: no published human-hours to merge outputs from N parallel agents running T hours each. The conflict rate is measured; the labor is not.
- For agent-authored parallel PRs there is one corpus, the 2,807-repo arXiv sample, and nothing at production scale beyond it.

“*Haiku researches.*”

CHAPTER 28

Stop Using Opus for Everything

"You don't bring a $400 torque wrench to change a light bulb. You bring the cheap one you already own, and you save the good one for the bolt that will kill you if it backs out."

ME, AFTER A BILL LINE TAUGHT ME THE DIFFERENCE.

I fanned out fifteen subagents (Ch. 16) to survey a repo, and every one of them inherited Opus 5 because nobody had set a per-agent default. Four of them existed to run `grep`. A Zerg rush (StarCraft, 1998) works because zerglings are cheap; this was a Zerg rush where every zergling cost as much as an ultralisk. Nothing errored. The dashboards were green, the rows filled in, and the only artifact of the mistake was a bill line four days later. That is the whole failure mode: the expensive model isn't smarter at reading a docs page, just more expensive at it, and nothing in the system will ever tell you.

> ***Bottom line:*** *Haiku researches. Sonnet codes. Fable plans. Opus judges. One god model on every call is how you go broke and still ship dumb. Anthropic's September 2026 price sheet puts Haiku 4.5 at $1 per million input tokens and $5 per million output, Opus 5 at $5 and $25. That's 5x for the same fetch. Yes, Fable 5 at $10/$50 is the priciest tier in my table and I still send planning to it, because that is one 6K-token call in front of a run that will spend four million tokens executing. Tiering is not "always go cheap," it is "spend where being wrong is expensive." Routing by task tier is the highest-ratio cost fix in an agent system, and it usually makes the output better too: a small fast model that runs twenty times beats a big slow one that runs once and grabs the wrong file.*

• • •

WHEN IT BITES

- Your research subagent burns 400K input tokens reading changelogs on Opus 5 and returns a paragraph a Haiku call would have written for a fifth of the price.
- You fan out fifteen subagents and every one inherits the parent's model, because the routing rule lives in your head and in a config file the spawn path never reads.
- Two near-identical prompts land on different tiers, so neither one hits a warm cache and you pay full freight twice for the same context.
- Somebody proposes fixing cost by "using a cheaper model," singular, everywhere, and quality falls off a cliff on the 5% of calls that were load-bearing.

• • •

THE PATTERN

FrugalGPT (Chen et al., arXiv:2305.05176, May 2023) showed up to 98% cost reduction by cascading queries to appropriately sized models instead of sending everything to the top. RouteLLM (LMSYS, July 2024, arXiv:2406.18665) got specific: 85% cost reduction on MT Bench while holding 95% of GPT-4 performance, needing GPT-4 for only 14% of calls, 75% cheaper than routing at random.

Two years later the production numbers landed in the same band: 45% to 85% depending on workload (TianPan.co, November 2025), 40% to 85% across the 2026 survey work.

So the tiers. Mine, as a hard rule in my global config, not a suggestion:

Research goes to Haiku. Search, fact-gathering, reading a repo, summarizing a changelog, checking whether a package exists. High input volume, low output volume, which is exactly the shape Haiku's $1/$5 is built for. A research subagent that reads 800K tokens and writes 2K costs $0.81 on Haiku and $4.05 on Opus. Run twelve in a fan-out and you have paid $9.72 versus $48.60 for identical paragraphs.

Coding goes to Sonnet. Sonnet 5 sits at $2 in, $10 out. Coding is the one task where output tokens are the bulk of the bill, because you are generating diffs, and it is also the task where the frontier-versus-mid gap has narrowed the most. Sonnet writes the patch. If the patch is wrong, you find out from the test suite, not from a more expensive opinion.

Planning goes to Fable. Fable 5 is $10 in, $50 out, the most expensive thing on the sheet, and I still route planning to it. Paying 10x on 0.15% of the volume is a rounding error; a bad plan is not (Ch. 15). The title says stop using Opus for *everything*, not stop paying for anything.

Judging goes to Opus. Review, adjudication, the "is this actually done" gate. Small input, small output, high consequence. This is where you want the model that will notice the thing the coder rationalized past.

Within a tier the model can still escalate upward, and knowing when is the other half of the job. Confidence-based cascades let the small model abstain rather than guess: 13% cost reduction and 5% fewer errors for a 4.1% bump in abstention (TianPan.co). A model saying "I can't do this one" beats a confident wrong answer that a downstream step trusts.

Routing stacks with the two cheapest things Anthropic will sell you. Batch API is a flat 50% off both directions. Prompt caching drops cached input to 0.1x base, a 10x cut. Both are a config line, and together with routing they get past 60% total cost reduction.

• • •

ONE WORKED EXAMPLE

airank, 6 August 2026. For months I paid OpenRouter to tell me what ChatGPT would *probably* say about a brand. Not what it said. A different model's impression of it, billed per call, at a price above asking the real thing. Then somebody ran the comparison nobody had run in six months: the real ChatGPT API gave different answers about a third of the time. We killed the approximation layer and went to direct measurement, and the bill went down while the accuracy went up in the same commit. Jeremy Schoemaker, guy who has been buying traffic since dial-up, paid a premium for a guess about an endpoint two lines of curl away. I had a Geocities-grade under-construction GIF where the measurement should have been.

That is the tier-zero version of routing: before you optimize which model handles a task, check that the task needs a model at all. A third of the routing wins I've shipped were not "route this to Haiku," they were "this doesn't need an LLM call, it needs a SQL query." Once the fake calls are gone, the surviving ones are the ones you tier. Here are the dollars for those. Every call I make to every provider goes through aigate, so this is the whole bill, not one vendor's slice.

MODEL	REQUESTS	TOKENS BILLED	SPEND
claude-opus-5	79,302	11,074.1M	$7,400.29
claude-fable-5	22,591	5,172.7M	$6,218.96
gpt-6-astra	15,090	2,073.4M	$6,202.92
muse-spark-1.2	71,182	11,260.4M	$3,183.24
claude-sonnet-5	67,259	7,987.7M	$2,330.88
claude-fable-5-1	3,345	846.4M	$1,121.16

aigate, 30 days ending 8 September 2026. Top six rows by spend. gpt-6-astra and muse-spark-1.2 are non-Anthropic models aigate also bills, because the vault routes every provider, not just this chapter's four; claude-fable-5-1 is the point-release variant of Fable 5, not a typo. Haiku is absent because the screenshot ranks by spend: the tier I send the most calls to never spends enough to make a top-six list, which is the entire argument in one missing row.

Totals: $27,291.83 tracked cost, 283,164 requests, 40,404.2M tokens billed, 0.0% errors, 1.17 s average latency, peak day $3,155.65.

Divide the columns and the argument stops being rhetorical. Sonnet runs about $0.035 a request, Opus about $0.093, Fable 5 about $0.275. Sonnet took 24% of the requests and 8.5% of the spend: a quarter of the work for under a tenth of the money. That row is the chapter.

Opus at 28% of requests looks like it contradicts "Opus judges." It doesn't, quite: judging happens per step, not per run, so one coding task can spend a single Sonnet call and four Opus gate checks. It is still the row I am least happy with, and the next thing I cut is the review calls that are really lookups.

The named reroute is the fan-out from the hook. Fifteen agents surveying a repo, roughly 800K in and 2K out apiece, at $4.05 per request on Opus. Moved to Haiku, the same fan-out bills $0.81 per request, an 80% cut on a job whose output nobody could tell apart.

The line that makes the total survivable is the cache. aigate reports a 97.5% cache hit rate on requests; by tokens it is 97.2%, 39,290.4M of 40,404.2M read from cache. That is why 40 billion tokens cost $27k and not ten times that. Routing picks which model reads your context; caching decides whether you pay full freight to send it again.

What I cannot give you is the before. I never captured a clean untiered month at comparable traffic, so that number does not exist and I am not going to invent one. Everything above is thirty days of a system that was already tiered.

• • •

THE QUIET FAILURE

The loud failure is the bill. Somebody screams, you fix it in an afternoon.

The quiet failure:

Opus makes your bad system look adequate for exactly long enough to ship it.

A frontier model papers over a vague prompt, a missing tool, an unbounded loop, a context window stuffed with garbage. Then you tier the system for cost, the compensation goes away, and eleven things break at once that were broken the whole time. People conclude the cheap model is bad. The cheap model just told you the truth.

Second: **the parent's model silently becomes every child's model.** Set Opus once at the top of a fan-out and fifteen subagents inherit it, including the four that only run `grep`. Nothing errors. You get pwned by a default you wrote yourself, and the receipt is a bill line, not a stack trace.

• • •

DO / DON'T

Do

- Write the routing table down as config, not as intent. Research to Haiku, coding to Sonnet, planning to Fable, judging to Opus. Verify subagents actually inherit it by checking a bill line, not a code path.
- Tier on token shape and blast radius. Input-heavy and repeated goes cheap. Rare and consequential goes expensive even at $10/$50.
- Turn on prompt caching (0.1x on cached input) and Batch API (50% off both directions) before you argue about model choice. Both are configuration, not architecture.
- Let the small model abstain. An explicit "exceeds my capability" that escalates is worth 5% error reduction and costs less than a confident guess plus a rollback.
- Benchmark the cheap tier against the expensive one on your own traffic. RouteLLM's 95%-quality-at-14%-of-calls is their workload, not yours.

Don't

- Don't run everything on the frontier model because it's simpler. Per the September 2026 sheet it is 5x, and 85% of your calls do not notice the difference.
- Don't fix cost by downgrading everything one tier. That is not routing, that is a haircut, and it lands on the 5% of calls that carried the run.

- Don't let the cascade become the cost. Bill the router's own calls and retries as their own line, then measure that overhead as a share of the spend routing saved; if it exceeds the savings, flatten the cascade to a flat table.
- Don't route a task that shouldn't be a model call. If a SQL query, a cached lookup, or a regex answers it deterministically, that is the answer. See 6 August: the cheapest inference is the one you deleted.
- Don't assume the expensive model is judging well just because it's expensive. Give the judge a rubric and a measurable done (Ch. 20).

• • •

WHERE THIS SITS IN THE BOOK

Ch. 16 is routing as control flow: which lane a request goes down. This is the cost tier of the same decision, and both failure modes are the agent picking a lane on vibes. Ch. 45 is the token economics in full, where per-call routing rolls up into a monthly number somebody has to defend. Ch. 7 is the tiny model that turned out to be enough: most of what you route to a frontier model was never a frontier problem.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (four-tier routing by task type, as a hard config rule). Argument, not citation.
Verified:

- Anthropic pricing, September 2026: Haiku 4.5 $1/$5 per Mtoken, Sonnet 5 $2/$10, Opus 5 $5/$25, Fable 5 $10/$50. Anthropic Platform Docs: https://platform.claude.com/docs/en/about-claude/pricing
- Batch API 50% discount on input and output; prompt caching at 0.1x base on cached input (10x reduction). Anthropic Platform Docs, September 2026: https://platform.claude.com/docs/en/about-claude/pricing
- RouteLLM: 85% cost reduction on MT Bench, 45% MMLU, 35% GSM8K, holding 95% of GPT-4 performance; matrix factorization router hit that 95% mark using only 14% of GPT-4 calls, 75% cheaper than random baseline. LMSYS Blog / arXiv:2406.18665, 1 July 2024: https://www.lmsys.org/blog/2024-07-01-routellm/
- FrugalGPT: up to 98% cost reduction via LLM cascade strategies. Chen et al., arXiv:2305.05176, 9 May 2023: https://arxiv.org/abs/2305.05176
- 45-85% cost reduction at 95% frontier quality routing 85% of queries to cheaper models; 5x efficient-versus-frontier price gap. TianPan.co, 3 November 2025: https://tianpan.co/blog/2025/11/03/llm-routing-model-cascades
- Confidence-based cascades with early abstention: 13% cost reduction, 5% error-rate reduction, 4.1% abstention increase. TianPan.co, 3 November 2025: https://tianpan.co/blog/2025/11/03/llm-routing-model-cascades
- 40-85% documented production cost reductions across cost tiers, 2025-2026. DigitalApplied, Splunk, Inworld AI, MindStudio, June-July 2026: https://www.digitalapplied.com/blog/llm-model-routing-2026-cost-quality-optimization-engineering-guide
- Worked example: paying OpenRouter to approximate ChatGPT, ~33% answer divergence from the real API, airank, 6 August 2026. `~/Projects/airank/blog/2026-08-06-the-day-we-stopped-paying-to-guess.md`
- aigate Usage & Analytics, screenshot 9 September 2026: 30 days ending 8 September 2026, $27,291.83 tracked cost, 283,164 requests, 40,404.2M tokens billed, 97.5% cache hit rate, 1.17 s average latency, peak day $3,155.65, per-model spend as tabled (top six rows by spend).

What I could not verify:

- No before-tiering number exists. I never captured a clean untiered month at comparable traffic, so the chapter says so instead of estimating one.
- No verified 2026 case where router overhead exceeded routing savings: no company post-mortems, no verified Anthropic customer case studies on routing ROI or routing-related incidents. The guardrail in Do / don't is my own practice, not a cited threshold.

"*Two agents talking to each other in English is two agents guessing.*"

CHAPTER 29

Agents Talking to Agents

"TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others."

JON POSTEL, RFC 793 (1981)

Seven agents got a brief that said the `air` CLI had a 180-second cap, then return whatever it has. Six of them accepted it and started reasoning about tuning. One opened the file. The cap existed on a branch that had not merged, and the 429 rate-limit branch slept with no elapsed-time check at all, which is a polite way of saying forever. Nobody had lied. The brief was English, the code was code, and English does not get checked at the door. It took a production query over 8,853 domains to settle what everyone had been confidently discussing, and the number that ended it was zero.

> ***Bottom line:*** *Two agents talking to each other in English is two agents guessing. The handoff has to be a contract: a named schema, required fields, typed values, and a failure that fails instead of apologizing. Google shipped A2A on April 9, 2025 with 50+ launch partners, Salesforce, SAP, ServiceNow, PayPal, MongoDB, Box, Atlassian, Intuit, Cohere and LangChain among them. That is a room full of vendors who agree on nothing else agreeing in public about the thing anybody running a multi-agent loop learns the hard way: no schema, no A2A. Prose is not an interface. Prose is a rumor with good grammar.*

• • •

WHEN IT BITES

- Agent A writes "the migration looks mostly fine, minor concerns on certs." Agent B reads approval and starts the cutover. Nobody lied. Nobody agreed either.
- Your planner emits a paragraph. Your executor regexes it for the word "deploy." It works for six weeks and then somebody writes "do not deploy." It looks like you're trying to deploy. Would you like help with that?
- Two agents on the same task both produce the status string "done." One means the code changed. One means production was verified. Ch. 15 is that same knife.
- A downstream agent gets a field it did not expect, silently ignores it, and returns success. No parse error, because there was no parser.

• • •

THE PATTERN

There are three different problems here and people keep solving one of them and declaring victory.

Tool access. An agent calls something that is not an agent: a database, a filesystem, an API. That is MCP, which Anthropic released on November 25, 2024 and which everybody now compares to USB-C for AI applications (Teneo, 2025). I had it filed in my head as an early-2024 protocol, off by

the better part of a year, because I dated it from when I started using it instead of when it shipped. Ch. 24 is all of it. One model, many tools, a contract per tool.

Peer coordination. An agent needs another agent to do a piece of work and hand back a result. That is A2A: agents built by different vendors, on different frameworks, coordinating without either side importing the other's SDK. Auth0's July 2025 writeup put the split in one line worth stealing: MCP connects agents to tools, A2A connects agents to agents.

System integration. Neither of the above. Your agent hits an internal service that has had a REST contract since 2019 and does not need a protocol upgrade. Oracle's developer blog called this the agent communication matrix in June 2026, and the useful part is the permission it gives you to not adopt anything.

What all three share is the only thing that matters: JSON Schema. A2A is JSON-based and schema-carried (TrueFoundry, June 2026). MCP tools declare typed inputs. OpenAI's structured outputs use constrained decoding (OpenAI API docs, 2025). That is the difference between "please respond in this format" and a decoder that physically cannot do otherwise. Ch. 8 makes this argument for a single agent's output. This chapter runs it between two processes, where the failure is invisible to both of them.

The protocol is the boring part, and the only part that survives a model swap. If your agents disagree about what "verified" means, A2A will transmit that disagreement with excellent uptime.

The three fields that earn their keep in any agent-to-agent envelope:

A status that is an enum, not a sentence. `certified`, `certified_conditional`, `declined`. Not "looks good with some caveats." Those three values are how I remember my own council's envelope: three status values, four confidence levels, one refusal path. I have not gone back and diffed my memory against the code, which tells you everything about me. I will lecture you on my own wire format from memory and refuse to spend ninety seconds confirming I actually built it that way. Treat it as the shape I argue for, not a spec you can copy.

A confidence label per claim. Verified, contested, theory, unmeasured. If a downstream agent cannot tell which claim you ran a command to check, it will treat them all the same and pick the wrong one.

A refusal path. The receiver must be able to say no in a way the schema forces the caller to handle. If declining lives only as prose in a `notes` field, nobody handles it.

• • •

ONE WORKED EXAMPLE

airank, August 12, 2026. An architecture council of six frontier models plus a chair with tool access reviewed the AWS hybrid cutover plan. One question: is this runbook safe to execute on Sunday.

The verifier seat, Seraph, declined to certify. Not "raised concerns." Declined, as a status, with the artifact required to move off that status attached. Then it produced six breaks, every one a thing rather than an opinion:

1. A TLS certificate issuance race.
2. A frozen-schema mismatch on the fallback server.
3. LiveSearch job window bleed.
4. A cold generation-key cache.
5. Hourly binlog shipping with no named successor.
6. A dead Tailscale daemon masking the health check.

All six were real. All six went into the runbook verbatim. That word is the whole chapter. It arrived in a shape a different agent could execute without reading the conversation.

The council also settled a Multi-AZ argument in one line: *Multi-AZ replicates a DROP SCHEMA faithfully and instantly.* The actual insurance is point-in-time recovery, not a standby replica. That is a claim you can test.

Cost of the entire verification pass, as best I remember it and as far as the post goes: five TCP connects from the actual home network, one `dig NS`, one `aws rds describe-db-engine-versions`, an rsync check, and a short tail of other calls. I never published the session log, so read the call list as

my recollection; the tally in the post is the part with a date on it. Eight retractions on evidence, six concrete breaks in the runbook, one architecture question settled, and the whole hybrid footprint priced at $134 a month. You cannot retract the third sentence of a paragraph.

Outcome: the runbook was certified **conditionally**, on a Saturday dress rehearsal returning a green checklist. That is a status with a named unblocking artifact, not "we feel okay about it." And the launch was never allowed to depend on the migration.

Three days later, August 15, 2026, the same council reviewed the `air` CLI, 180 lines, zero dependencies, one day old. Seraph checked the source instead of the brief that started this chapter.

The brief was the prose handoff. The code was the contract. They disagreed, and six of seven agents reasoned off the wrong one until somebody read the file. I wrote that brief, and I got the single most important number in it wrong, because I described the branch I meant to merge instead of the one that was there. Seven models on the call and the cheapest thing in the room was `cat`.

Then the council ran a production query. Of 8,853 domains created in 48 hours, 117 resolved in under 60 seconds, **zero** landed in the 60-to-180-second window, and 8,736 took longer than 180 seconds, some of them 40 hours. That distribution is a switch, not a curve. "Raise the cap" and "tune the polling interval" both died on one query, and a different proposal got promoted: when the cap expires, exit 2 instead of 0. Honesty at the boundary as an exit code, the smallest possible schema and still infinitely better than a log line.

• • •

THE QUIET FAILURE

The loud failure is a parse error. Agent B chokes on Agent A's output, the run dies, you see a stack trace, you fix it in ten minutes. Fine.

The quiet failure:

Both agents parse successfully and mean different things.

Agent A's `status: "complete"` means the code is committed. Agent B's handler for the same value means it is safe to deploy. Every field is present and correctly typed. The schema is satisfied and the system is wrong, because a schema constrains shape and shape is not meaning. A2A moves the envelope; it does not agree on the semantics inside it, and no logo wall of launch partners has solved that for you either. I am describing the shape rather than one incident I can put a date on, because this one never leaves a stack trace to date it by. That is the point of it.

Second quiet failure: **the optional field that is never sent and never missed.** You add `evidence` to the envelope as optional, because making it required would break two existing agents. Six weeks later no agent populates it, every consumer has a null-safe default, and the field is documentation for a discipline nobody practices. Required or delete it. I shipped that exact optional `evidence` field, defended it in review as pragmatic, then cited it in a design doc as though somebody was filling it in. It was null in every row and I was the n00b quoting it back at myself.

Third: **summarization in the middle.** Somebody puts a coordinator agent between A and B that reads structured output and emits prose, because prose is easier for the humans on the dashboard. You now have a lossy transcoder in your control path. The commander-in-chief sim (v1.2.0 gold, August 24, 2026) refuses this by construction: every field in the deterministic core is classified as hashed or excluded from the golden checksum, no third option, and its README badge reports 1,217 test methods enforcing bit-identical results between x86_64 and arm64. No seat at that table for "roughly the same."

• • •

DO / DON'T

Do

- Make the handoff a typed envelope with a versioned schema name in it. When you change the contract, bump the version and let old consumers fail loudly.
- Use constrained decoding where your provider offers it. OpenAI's structured outputs enforce the grammar at sample time, which converts a class of runtime bug into an impossibility.

- Give every claim a confidence label the receiver is required to branch on. The council's verified / contested / theory / unmeasured split is why eight retractions were possible.
- Make refusal a first-class status, and attach the unblocking artifact to any conditional one. "Certified pending Saturday dress rehearsal green checklist" is actionable by an agent that was not in the room; "declined" was usable on August 12 because it was a value, not a mood.
- Reach for MCP when the far end is a tool and A2A when the far end is a peer with its own loop. Reach for plain REST when the far end is a service that already works.

Don't

- Don't hand an agent a prose brief and assume it describes the code, especially when you wrote both. I authored both `air` sentences and only one of them ran.
- Don't adopt A2A because it was on a slide. Fifty vendors signing on at launch is evidence the problem is real, not evidence you have it.
- Don't ship optional fields in an agent contract. They get ignored uniformly and then cited as if populated.
- Don't confuse schema validation with agreement. Two agents can satisfy the same JSON Schema and still disagree about "done," which is Ch. 20's problem wearing a protocol as a costume.
- Don't build a bespoke inter-agent format in 2026. That was defensible in 2023.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 is structured output for one agent: get the model to emit a shape you can act on instead of a paragraph you have to interpret. Ch. 24 is MCP, the tool-access contract, the layer under everything here. Ch. 26 is the multi-agent structure this chapter gives a wire format to. Ch. 15 is the failure this chapter inherits: a plan is a persuasive artifact, and a persuasive artifact passed between two agents is worse than one passed to a human, because the human might squint at it. Ch. 20 is where it lands, again: if "done" is not a measurement, the schema will transmit your ambiguity with perfect fidelity.

• • •

SOURCES AND RECEIPTS

Verified:

- A2A launch, April 9, 2025, 50+ partners including Atlassian, Box, Cohere, Intuit, LangChain, MongoDB, PayPal, Salesforce, SAP, ServiceNow. Google Developers Blog, April 2025: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
- MCP released November 25, 2024. Anthropic, November 2024: https://www.anthropic.com/news/model-context-protocol
- MCP connects agents to tools, A2A connects agents to agents. Auth0, July 2025: https://auth0.com/blog/mcp-vs-a2a/
- A2A JSON-based and schema-carried; both use JSON Schema. TrueFoundry, June 2026: https://www.truefoundry.com/blog/mcp-vs-a2a
- Structured outputs via constrained decoding. OpenAI API docs, 2025: https://developers.openai.com/api/docs/guides/structured-outputs
- The agent communication matrix: MCP, A2A, plain REST. Oracle Developers Blog, June 2026: https://blogs.oracle.com/developers/the-agent-communication-matrix-when-mcp-a2a-and-plain-rest-each-win
- MCP as USB-C for AI applications. Teneo.ai, 2025: https://www.teneo.ai/blog/mcp-and-a2a-protocols-explained-the-future-of-agentic-ai-is-here
- Council and the cutover, airank, August 12, 2026: `~/Projects/airank/blog/2026-08-12-the-council-and-the-cutover.md`

- Council reviews the `air` CLI, airank, August 15, 2026: `~/Projects/airank/blog/2026-08-15-air-cli-council.md`
- commander-in-chief deterministic sim, v1.2.0 gold, August 24, 2026: `~/Projects/commander-in-chief/README.md`

What I could not verify:

- A2A's post-launch adoption count and its canonical spec URL are deliberately absent above. The press link I was working from returns a 404, and no number goes in this book on the strength of a dead link.

“One agent in production is a risk you can reason about.”

CHAPTER 30

Two Agents. One Prod. One Disaster.

"Never attribute to malice that which is adequately explained by stupidity."

ROBERT J. HANLON, MURPHY'S LAW BOOK TWO: MORE REASONS WHY THINGS GO WRONG! (1980)

August 9, 2026. My Laravel queue had `retry_after` at 90 seconds. The jobs it governed had a 900-second timeout. So at T+90 the queue called the worker dead and handed the job to a second one, while the first still had 810 seconds of legal runway. Two workers, one job, one provider account, no idempotency key on the outbound call. Every slow call was about to go out as a second billed request to a real customer. Nothing threw. Both workers would have reported success, because both of them succeeded. (Five days later the same stack handed me a 401 on a key three of my own layers swore was fine. I will come back to that.)

> ***Bottom line:*** *One agent in production is a risk you can reason about. Two agents is a different animal, because the second one isn't a second risk, it's a multiplier on the first. They share a database, an inbox, a rate limit, a credential, a queue, and neither one knows the other exists. The failure you will actually eat isn't the wiped database. It's an email that went out twice, or a row one agent wrote and the other quietly overwrote four hundred milliseconds later, with nothing logging an error. Everything reported success. The system was wrong anyway.*

• • •

WHEN IT BITES

- Two agents poll the same `pending` table. Both grab row 4471. Both send the follow-up. The customer gets the same "just checking in" email twice, ninety seconds apart, signed by someone who wrote neither.
- Agent A marks a ticket resolved. Agent B, working from a snapshot read eight seconds earlier, writes the record back with `status = open`. A's work is gone. No error, no conflict, no log line. Last write wins, and last write was the stale one.
- Both agents hit the same API key. The provider rate-limits on the key, not the agent. A gets throttled because B is mid-sweep, reads the 429 as its own fault, and backs off into uselessness.
- One agent rotates a credential during cleanup. The other is holding that credential in memory and keeps using it.
- You add a second agent to make things faster and throughput goes *down*, because both are grinding the same locked rows. Two workers grinding on one row and nothing finishes. That's what she said, and she was describing lock contention.

• • •

THE PATTERN

An agent loop is read, think, act. The read is T0. The thinking takes 800 milliseconds to forty seconds. The action is T1. In a single-agent system you can usually pretend those are the same in-

stant, because nothing else with write access touches the data in between.

Put a second agent in and that gap is where the bugs live. It's a read-modify-write race, and the database people solved it with transactions long before anybody shipped an agent, except your agent isn't using one. It's using `SELECT`, then a large language model, then `UPDATE`: the longest, least atomic critical section anybody has ever shipped on purpose.

CockroachDB wrote this up in July 2026: agent loops fail in production when multiple agents read and write the same tables without serializable isolation. The oldest finding in databases, arriving at an audience that did not know it was writing a distributed system.

The other half is blast radius. Mondoo's postmortem on PocketOS (April 30, 2026) put it cleanly: the failure mode was overprivileged tokens and a shared blast radius between backups and primary, not AI reasoning. The agent executed the command it was given. Two agents sharing one overprivileged credential means either can do the maximum damage that credential permits, and neither has to be wrong to do it.

So the pattern:

1. Two loops, one state.
2. No serialization between them, because nobody thought of the agents as concurrent writers.
3. One credential with more scope than either agent needs.
4. Actions that leave the system (emails, webhooks, charges, Slack messages) with no idempotency key.
5. Silence, because every individual operation succeeded.

Item 4 costs you a customer. A duplicated row you dedupe on Tuesday. A duplicated email is in somebody's inbox forever.

• • •

ONE WORKED EXAMPLE

airank, August 9, 2026. The near miss, walked in order, because the order is the whole lesson.

T0. A worker claims a job off the queue. Laravel stamps a reservation on that row and the job starts. The job is a provider sweep: slow outbound call, response, write results. Configured timeout, 900 seconds, deliberately, because the calls really do take that long.

T0, also. The queue's `retry_after` is 90 seconds. Separate setting, separate file, never read as a pair. It means: if a reservation is older than 90 seconds, assume the worker died and hand the job to somebody else. It is not a check on the worker. It is a clock, and the clock does not know the worker is fine.

T+90. Reservation expires. The first worker is healthy, mid-call, 810 seconds from its own deadline. The queue does not ask. It hands the identical payload to a second worker, which starts the identical call against the identical provider account.

T+90 through T+900. Two workers, one job. Both behave perfectly. Both finish, write results, report success. The provider bills both, and the customer-facing side effect fires twice. There is no error anywhere in this story, which is exactly why it would have run for weeks.

The fix is not just "raise `retry_after`," although do that too: `retry_after` must exceed the longest a job may legally run, or the queue is a duplicate generator by design. The real fix is an idempotency key derived from the job, not the attempt, so the second worker's call collapses into the first no matter how many workers the queue invents.

I did not catch this because I am careful. I caught it because it was standing in a lineup of four other controls that had all reported success and done nothing.

Now swap the second worker for a real second agent and it gets worse, because that one has opinions. On August 14 a live sweep came back 401 on a key my vault, my config, and my orchestration layer all labeled working, because none of them had ever spent an HTTP request asking the provider. Three consumers, one rumor. Put a rotating agent next to a sweeping agent and the sweeper fails on a key that was correct when it read it and dead when it used it, and nothing in either log says "another agent revoked this."

The version that reached actual customers is not mine. Visa and Worldpay published a joint statement on Coinbase's blog in February 2018: "This issue was not caused by Coinbase." Two de-

fensible layers each acted on the same card transaction with no idempotency between them, and customers got charged twice. Same shape, company scale, no agent anywhere in it.

• • •

THE QUIET FAILURE

The loud failure has a Fortune headline. July 23, 2025: a Replit AI agent deleted a production database during an active code freeze, then confessed, "This was a catastrophic failure on my part. I destroyed months of work in seconds." April 25, 2026 gave us the sequel: a Cursor agent deleted the PocketOS production database and every backup in nine seconds, per Mondoo's writeup.

Loud is survivable. The quiet failure isn't:

Two agents both succeed, and the damage is the intersection.

Nothing throws. No table drops. Every log line is green. The follow-up went to a customer twice, or agent B's write erased agent A's fifteen minutes of work and both reported completion. In my August 9 case it would have been one duplicate billed request per slow call, indefinitely, and I have no idea how long it would have taken anyone to notice. You find this three weeks later when someone replies "why did you send me this again," and spend a day proving it was not a mail server retry.

Second quiet failure: **you can't tell which agent did it.** Both write as the same service account, into rows with one `updated_at` and no actor column. That is a receipt with the name torn off.

Third: **your diagnoses rot faster than your facts.** August 8, 2026, same shop: one probe tested an old Chrome profile path and said "logged out," another tested fresh state on the same container and said "logged in," and the cheap direct check (`storageState` on the running container) settled it in seconds. When two agents each carry their own rotted diagnosis into a shared database, you get two systems acting decisively on incompatible beliefs about the same rows.

Fourth: **throughput goes backwards and you blame the model.** You add the second agent, jobs per hour drop, and you tune prompts and swap models for a week. It was lock contention. The models were fine.

• • •

DO / DON'T

Do

- Give every outbound side effect an idempotency key derived from the thing, not the attempt. `send:followup:ticket-4471:2026-08-14` sends once no matter how many agents decide it should.
- Claim work before doing it. `UPDATE jobs SET owner='agent-a', claimed_at=NOW() WHERE id=? AND owner IS NULL` and check the affected-row count. Zero rows means somebody else has it.
- Audit `retry_after` (or your queue's equivalent) against your longest legal job runtime. If the reservation expires first, the queue duplicates work by design.
- Use serializable isolation for anything two agents touch, per the CockroachDB writeup (July 1, 2026). Retries are cheaper than a double refund.
- Give each agent its own scoped credential, and keep backup credentials separate from primary, so the audit log can name the guilty party and shared blast radius doesn't do the PocketOS thing to you.
- Verify labels with probes, run at the start of the sweep.
- Gate destructive actions behind something the agent can't talk its way past. The Replit agent had explicit instructions. Instructions aren't a gate.

Don't

- Don't add a second agent for throughput before measuring contention. You may be buying a slowdown.
- Don't dedupe on the receiving end and call it fixed. If the outbound path can send twice, it will, to someone whose mail server doesn't dedupe.

- Don't let both agents write as the same service account. An unattributable incident is one you get to have twice.
- Don't write a plain `UPDATE` from a snapshot the model read forty seconds ago. Add the version check: `UPDATE tickets SET status=?, version=version+1 WHERE id=? AND version=?`, then check the affected-row count. Zero rows means the row moved under you and your model's whole plan is stale.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 25 is the credential and permission layer, and this is what happens when two agents share it. Ch. 41 is observability: without an actor column a dual-agent incident is unattributable. Ch. 51 is the same failure at human blast radius. Ch. 15 sits upstream: a plan written by one agent is a snapshot the other is invalidating while you read it.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (shared state is the whole failure, and the expensive version reports success): argument, not citation.

Verified:

- Replit AI agent deleted a production database during a code freeze. Fortune, July 23, 2025: https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/
- Agent admitted violating explicit instructions and running unauthorized commands during the freeze; Fortune quotes the agent as saying "This was a catastrophic failure on my part. I destroyed months of work in seconds." Fortune, July 23, 2025, same URL.
- Nine days: SaaStr founder Jason Lemkin, "After nine mad days of us vibe coding, Replit's AI Agent deleted a production database." SaaStr.com, July 2025: https://www.saastr.com/replits-new-release-address-most-of-the-challenges-we-hit-vibe-coding-but-is-prosumer-vibe-coding-really-ready-for-commercial-apps-yet/
- Catalogued as Incident 1152, LLM-driven Replit agent executed unauthorized destructive commands. AI Incident Database, July 21, 2025: https://incidentdatabase.ai/cite/1152/
- Cursor agent (Claude Opus 4.6) deleted the PocketOS production database and all backups in 9 seconds, April 25, 2026. Mondoo, April 30, 2026: https://mondoo.com/blog/5-lessons-from-9-seconds-ai-agent-deleted-production-database/
- Founder Jer Crane manually reconstructed customer bookings from Stripe payments and email confirmations over the following weekend. Mondoo, April 30, 2026 (citing Crane's X post), same URL.
- Agent loops fail in production when multiple agents read and write the same tables concurrently without serializable isolation. CockroachDB, July 1, 2026: https://www.cockroachlabs.com/blog/agent-loops-production-database-patterns/
- Internal postmortem, not reader-verifiable: "The night we changed instruments," airank, August 9, 2026. Queue `retry_after` at 90 seconds under a 900-second job timeout would have re-dispatched every slow call as a second billed request. Caught before it ran, alongside other controls that reported success and did nothing.
- Internal postmortem, not reader-verifiable: "The responder and the revoked key," airank, August 14, 2026. Live sweep returned 401; vault, config and orchestration all labeled the key working; no layer had ever probed it. Fixed by pairing labeling with a health poll.
- Internal postmortem, not reader-verifiable: "Fifty-eight duplicates, four of them real," airank, August 8, 2026. Contradictory confident probes on adjacent state, resolved by the cheap direct check (`storageState` on the running container).

- Coinbase customers charged more than once for the same card purchase between January 22 and February 11, 2018; merchant category code change reclassified crypto buys as cash advances; Visa told the Financial Times on February 16 it had made no systems change producing duplicates; Visa and Worldpay joint statement on Coinbase's blog the next day, "This issue was not caused by Coinbase." Cointelegraph, February 17, 2018: https://cointelegraph.com/news/visa-worldpay-take-blame-for-duplicate-charges-on-coinbase-reverse-transactions
- Electronic Payments International, February 19, 2018: https://www.electronicpaymentsinternational.com/news/coinbase-visa-refund-duplicate-card-transactions/

What I could not verify:

- No public postmortem of two AI agents duplicate-sending to the same customer. Coinbase 2018 is a two-layer duplicate charge, cited for the shape and not the species. CockroachDB (July 1, 2026) documents the read-modify-write race in general terms only.
- No documented public incident of one agent undoing another's work in a shared database, and no case study with numbers on lost writes from concurrent agents. Stated as mechanism, not reported event.
- The airank incidents are internal, presented as one shop's log and not an industry pattern.
- No public analysis of race conditions between an agent's observation and action phases. The T0/T1 argument is reasoning from concurrency first principles, not a citation.

"Your agent will hit a 429, a timeout, a 500, a dead ALB node, a MariaDB host block."

CHAPTER 31

It Blew Up. Now What.

"Give a man a fire and he's warm for a day, but set fire to him and he's warm for the rest of his life."

TERRY PRATCHETT, JINGO (1997)

For roughly fourteen hours my monitoring loop told me everything was fine. Twenty consecutive checks, 200s across the board, target health green. The dashboard said 81 requests across three availability zones, and two of those zones had served 40 and 41 between them. I did that arithmetic maybe six times before it landed. I blamed my local network. Then I blamed DNS. I rewrote a resolver config at midnight for a problem that was never in my house. The third zone was not reporting zero. It was not reporting.

> ***Bottom line:*** *Your agent will hit a 429, a timeout, a 500, a dead ALB node, a MariaDB host block. That is not the interesting part. The interesting part is what the next 200 milliseconds do. Three moves, in order: retry the transient stuff with exponential backoff and jitter, degrade to a smaller answer when retrying stops helping, escalate to a human who is actually paged when degrading is not acceptable. And never swallow a fatal. An exception caught, logged at info, and returned as an empty array is a quiet outage, and quiet outages run for fourteen hours while the dashboard stays green.*

Most agent error handling in the wild is one `try/except` around the whole loop with a `pass` in it. That is a shredder.

• • •

WHEN IT BITES

- Your agent gets a 429 from Claude and retries in a tight loop, and now twenty workers are retrying in a tight loop against the same rate limit. You are the outage.
- MariaDB returns SQLSTATE 1129, host blocked. Your retry logic reconnects, and each failed connection pushes `max_connect_errors` further past its default of 100. You are now digging.
- A tool call raises, the agent catches it, and the model receives an empty tool result. The model does what models do with empty results: makes something up and keeps going, confidently.
- Your provider degrades instead of failing. Slower responses, truncated answers, no error code. Retry never fires because nothing technically failed.
- The exception handler writes to a log nobody tails. No page, a file still sitting there in six weeks.

• • •

THE PATTERN

Three tiers, in order, because each is more expensive than the last.

Tier 1: retry, but only what deserves a retry. A retryable error is one where the same request, sent later, plausibly succeeds. Connection reset, 503, gateway timeout, a rate limit that told you

when to come back. The mechanics are settled: exponential backoff with jitter. The AWS SDKs' standard retry mode adds different delay timing for throttling errors than for transient ones, plus a retry quota implemented as a token bucket, so a client that is failing a lot loses its right to keep retrying. That token bucket is the part people skip, and it keeps one sick client from becoming everyone's problem.

Jitter is not decoration. ByteByteGo puts a number on the unguarded version: retry storms burn 500 to 1000 seconds of timeout during a five-minute outage, because every client wakes on the same schedule and hits the recovering service at once. Rack2Cloud calls the endgame a self-inflicted DDoS.

What a retry storm costs in dollars I cannot tell you, because I never measured mine: 73 airank posts dated August 2026, all written by me, not one with a bill in it. Borrow ByteByteGo's arithmetic, not my bookkeeping.

My two hard rules, learned the expensive way:

Never retry into a 429. A 429 is not an error, it is instructions. RFC 6585 defined it in April 2012 and the proper response carries a `Retry-After` header telling you how long to wait. Read the header. Sleep. Retry once.

Never retry into MariaDB 1129. Host blocked is a state, not a hiccup, and no backoff curve fixes it, because the counter only goes up. Stop the workers, run `mysqladmin flush-hosts` once, reconnect with backoff, then check whether `skip_name_resolve` is off, because reverse-DNS failures are what filled the counter.

My own retry ceiling and backoff formula for Claude and OpenAI calls are set by feel and live in a config file, which is a confession, not a recommendation. Neither vendor publishes a default to check my feel against, so I tuned mine the way you tune anything you cannot measure: got burned, made the number smaller, got burned less.

Tier 2: degrade. Retrying has a budget. When it is gone, the question is whether a smaller answer beats no answer. Cached result instead of live. Cheaper model instead of the frontier one. Partial results with an explicit "incomplete" flag. Zylos Research, February 2026, found multi-agent systems failing at 41 to 86.7 percent in production without deliberate fault tolerance, and named circuit breakers as the thing missing from LLM API retry loops. That is one paper and four searches on 9 September 2026 turned up no second case study, so take the range as a single source. Nygard's circuit breaker, popularized in Fowler's March 2014 bliki entry, is a three-state machine: closed (calls pass through), open (calls fail immediately without touching the dying service), half-open (one probe decides whether to close again). The value is not failing faster. It is that you stop hammering something already down.

Degrading has a rule attached: the degraded answer must be labeled degraded, in the response object, where downstream code can branch on it. A cached answer identical to a fresh one is a lie your own system tells you.

Tier 3: escalate. Some failures are not the agent's to solve. Payment declined. Credential expired. A destructive action outside the agent's authority. The escalation path has to be a page to a named human, not a log line, and Runframe's May 2026 piece has the number: 73 percent of organizations reported outages linked to ignored alerts. My own rule, and I am printing it as an opinion because that is all it is, is "page a person, never a ticket queue." I run Uptime Kuma for the paging itself. A ticket queue is where an alert goes to be read on Tuesday. What I have never written down is who gets named and what fields ride along in the escalation, which makes my runbook a slogan wearing a runbook's jacket. Ch. 35 is the chapter where I stop getting away with that.

Underneath all three: **never swallow a fatal.** If the error means the agent cannot correctly complete the task, the run stops, the failure gets recorded in enough detail to reproduce, and something wakes a person. Catching it to keep the loop alive is how you get an agent that runs all night producing garbage that looks like work.

• • •

ONE WORKED EXAMPLE

airank, 7 August 2026. Three alarms, built and tested, none of them wired to anything.

First, the MariaDB host block. `max_connect_errors` at its default of 100, `skip_name_resolve` off, so every reverse-DNS failure counted against the limit until the host got blocked. The alarm fired into

nothing.

Second, a container healthcheck that computed an exit code and ended its shell line in `|| true`. Healthy forever, by construction: detection worked perfectly and six characters threw the result on the floor. Twenty-five years of shipping software, pwned by an operator I typed myself.

Third, worker liveness. `reapStale()` was written, tested, and never called: the scheduler meant to call it did not exist, and the threshold `AIR_CHATGPT_STALE_AFTER_HOURS` was null everywhere.

The fixes: `skip_name_resolve` on, grants verified as IP-based first. `start_period` extended from 90 seconds to 600, because ninety seconds was never enough time to come up. That's what she said, and she was right about my container too. The staleness check moved into the collector's own healthcheck: a detector living next to the thing it watches needs no separate opinion about whether that thing should be running.

The line that stuck: **detectors nobody reads are decoration with test coverage.** All three alarms had tests and all three passed. Passing tests on an unwired alarm are worse than no alarm.

Twelve days later, 19 August 2026, the same class in a new costume. An ALB node in us-east-1d stopped accepting connections. Failed connections never complete, so they never become requests, so they never appear in `RequestCount` or any error metric. The third zone was not showing zero, it was invisible, and `sum([])` equals 0, so the measurement conflated "no datapoints" with "no traffic." Two wrong diagnoses in a row, delivered with total confidence to a room of one. In Soviet Russia, the requests count you.

Removing the dead zone from the ALB subnet list fixed it and planted its own landmine: the ASG still launched instances into that zone. The rule I wrote down: a green dashboard is not evidence of reachability. Every signal computed from requests that arrived (status codes, error rates, target health, request counts) is a survivorship filter.

• • •

THE QUIET FAILURE

The loud failure is the agent that crashes. Stack trace, exit code, somebody notices in eleven minutes. The quiet failure:

The exception was caught, handled correctly by the letter of the code, and produced an answer that is wrong in a way nothing downstream can detect.

The airank cache incident on 13 August 2026 is the shape of it. A microcache tuned for throughput, 16.57 requests per second up to 1873, using `fastcgi_hide_header Set-Cookie` because the cached pages had no cookies. A closed loop: a visitor with no cookie could never receive one, because `Set-Cookie` was stripped from the response that would have given them one. Every form answered 419 Page Expired, which Inertia surfaces as a blank. No exception was raised anywhere. I tuned that cache myself, watched the throughput climb, and felt great for a day while not one human being on earth could log in. w00t.

The first fix matched `$uri`, the rewritten path, instead of `$request_uri`. It worked by accident.

Second quiet failure: **retry counts that hide a real error.** Nine failures and a success on the tenth reads as resilience. It is nine failures you decided not to look at, and when the tenth stops succeeding you will not know when the rot started.

Third: **the health check measures a proxy.** A container answering on port 80 says nginx is alive, not that the collector wrote a row.

• • •

DO / DON'T

Do

- Classify every error as retryable, degradable, or fatal before you write a handler. Unclassified errors get treated as fatal.
- Use exponential backoff with jitter and a retry token bucket, the way the AWS SDKs' standard retry mode does, and obey `Retry-After` on a 429. Cap the total, not the per-attempt delay.
- Put a circuit breaker in front of every external dependency your agent calls in a loop, and label degraded answers so downstream code can branch on quality, not just on success.

- Wire every alarm to a page with a named owner before you call it done, and keep the failing logs. On the commander-in-chief export validation, 7 September 2026, the harness had to fail against the earlier pack to prove it could detect the change.
- Measure from outside the system, following DNS the way a visitor does.

Don't

- Don't retry into a 429, and don't retry into MariaDB 1129. Flush hosts once, fix `skip_name_resolve`, then reconnect.
- Don't end a health check line with `|| true`, and don't ship a detector whose threshold env var is null everywhere. Both report healthy by construction.
- Don't return an empty result on a tool failure. The model cannot tell "nothing found" from "the call exploded," and picks whichever lets it keep going.
- Don't treat a green dashboard as reachability. Signals computed from arrived requests are survivorship filters.
- Don't let a fix plant its own landmine. The ASG still pointed at a dead zone after the ALB fix.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 15 is why the plan is not the work; this is what happens when the work blows up mid-plan, because the plan said step 7 and step 3 threw. Ch. 30 is the loop that keeps going; this is the seam where it should stop. Ch. 35 takes the human handoff. Ch. 48 is the observability layer underneath all of it, where the survivorship-filter problem gets its own treatment.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's: argument, not citation.
Verified:

- Circuit Breaker pattern, closed/open/half-open. Martin Fowler, March 2014: https://martinfowler.com/bliki/CircuitBreaker.html
- AWS SDKs standard retry mode, jitter plus token-bucket retry quota. AWS SDKs Reference, 2026: https://docs.aws.amazon.com/sdkref/latest/guide/feature-retry-behavior.html
- HTTP 429 and `Retry-After`, RFC 6585, April 2012. Zuplo Learning Center, 28 May 2026: https://zuplo.com/learning-center/http-429-too-many-requests-guide
- Multi-agent failure rates of 41 to 86.7 percent without fault tolerance. Zylos Research, 20 February 2026: https://zylos.ai/research/2026-02-20-graceful-degradation-ai-agent-systems/
- 73 percent of organizations reported outages linked to ignored alerts. Runframe, 31 May 2026: https://runframe.io/blog/how-to-reduce-alert-fatigue
- Retry storms burn 500 to 1000 seconds of timeout in a 5-minute outage. ByteByteGo, 28 May 2026: https://blog.bytebytego.com/p/must-know-failure-modes-in-distributed
- Retry storm as self-inflicted DDoS. Rack2Cloud, 2025: https://www.rack2cloud.com/retry-storm-self-inflicted-ddos/
- Worked example 1: three alarms, none of them wired. airank, 7 August 2026. `~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`
- Worked example 2: the outage my monitor couldn't count. airank, 19 August 2026. `~/Projects/airank/blog/2026-08-19-the-outage-my-monitor-couldnt-count.md`
- Quiet-failure example: the cache that ate every cookie. airank, 13 August 2026. `~/Projects/airank/blog/2026-08-13-the-cache-that-ate-every-cookie.md`

- Failing-logs discipline: the test that had to fail twice. commander-in-chief, 7 September 2026. `~/Projects/commander-in-chief/blog/2026-09-07-the-test-that-had-to-fail-twice.md`
- Retry-storm cost never measured on my own traffic: 73 airank posts dated 1 to 31 August 2026, zero with billing or proxy-cost data. `~/Projects/airank/blog/`

What I could not verify:

- Retry ceiling and backoff formula: set by feel, no vendor default published by Anthropic or OpenAI to calibrate against. Printed as a confession, no number claimed.
- Escalation packet contents and named owner: not documented anywhere in my own repos. Printed as opinion, no number claimed.
- No second public case study of graceful degradation against cascade beyond the Zylos paper. Four searches, 9 September 2026, nothing usable. Absence noted, not inferred.
- No public incident reports from Anthropic or OpenAI on 429 retry handling or cascade failures. Absence noted, not inferred.
- No published guidance on cold-start health-check disarm time. The 90s to 600s `start_period` change is one system's field observation.

“A *test you have never watched fail is not a test.*”

CHAPTER 32

If It Can't Go Red, It's Not a Test

"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair."

DOUGLAS ADAMS, MOSTLY HARMLESS (1992)

676 green tests. Every one of them passing, in a suite I was proud of, on the morning of 11 August 2026. We shipped. Somebody clicked a modal in production and nothing happened. Clicked again. Nothing. The markup was right there in the DOM, exactly where 676 tests said it would be, and the panel underneath it might as well have been a photograph of a modal. Nobody wrote that bug. The suite had been asked a question and had answered it honestly, and the question turned out to be the wrong one by one word.

> ***Bottom line:*** *A test you have never watched fail is not a test. It's a decoration that costs CI minutes. When a model writes the tests for the code that same model just wrote, you get a suite optimized for green, not for truth. On 11 August 2026 airank had 676 passing tests and every modal on the site was unclickable in production. The suite wasn't lying. It answered a question nobody had asked it, whether the markup renders, and the markup rendered beautifully. Nobody could click it.*

A green signal that cannot go red is worse than no test at all.

• • •

WHEN IT BITES

- You ask the model to "add tests for this module." It writes twelve, all pass on the first run, and you feel good. You should feel nothing, because you have no evidence any of them are connected to anything.
- Someone deletes a whole subsystem and the test count doesn't move.

• • •

THE PATTERN

There are four ways a test ends up structurally incapable of failing, and models produce all four enthusiastically.

1. The tautology. `assert True`. Or its grown-up cousins: asserting a constant you defined in the test, asserting that a list you just built has the length you built it to. The academic literature calls the neighboring smell Magic Number Test, and finds generated suites manifest it consistently, with the smell profile driven by prompting strategy (arXiv 2410.10628, October 2024, accepted to TOSEM July 2026).

2. The mock of the thing under test. The agent can't get the real dependency to behave, so it patches it, then patches one more layer. Eventually the only unmocked object in the test is the as-

sertion itself. Hard to spot in review: the test looks *more* sophisticated than a real one, setup, fixtures, a well-appointed room with no floor.

3. The wrong-axis assertion. The test checks a real thing, honestly, with no mocks, and checks the wrong property. This is the airank modal bug, the dangerous one, because everything about the test is professionally competent except what it measures. The Oppia project's end-to-end testing wiki (4 December 2025) says the quiet part: tests that only check DOM rendering miss hit-testing failures.

4. Tests written after the code, by the same model. This is the one that generates the other three. A model that wrote an implementation and is then asked to test it has no independent theory of correct behavior; its theory *is the implementation*. If the code is wrong, the tests agree with it, and CI is green. The survey work names weak fault detection as a still-open problem, with no standardized benchmarks to measure it (arXiv 2511.21382, 26 November 2025, updated 30 December 2025).

The Ch. 8 rule exists because of #4: **write the tests before the prompt.** Not before the code. Before the *prompt*. Let the model write the spec after it wrote the answer, and you've run an open-book exam where the student also wrote the answer key.

The tool that settles this is mutation testing: change the code on purpose, then check that the suite notices. Trail of Bits published the argument on 18 September 2025; Microsoft shipped first-party mutation testing docs for .NET on 3 April 2026; Meta's engineering blog (30 September 2025) closed the loop: LLMs generate more realistic mutants than traditional operators, at a scale humans never reached.

You don't have to run a full mutation campaign to get 90% of the value. Break the code by hand, once, and watch the test go red. If it stays green, you didn't write a test.

• • •

ONE WORKED EXAMPLE

airank, 11 August 2026. Not broken-looking. Dead: you clicked, and nothing happened, forever.

The tests asserted the modal markup was present in the DOM, and that was true. What they never asked was whether the panel could receive a click: the backdrop div sat above it in hit-testing order and swallowed every click before it arrived. The user saw a modal. The browser saw a backdrop.

Root cause: Tailwind v4 removed the `transform` class that had been creating the stacking context. Nobody wrote that bug. A dependency upgrade wrote it. The fix was one word, `relative` on the panel, which is the usual ratio of pain to patch.

I want to be precise about who the idiot is here. Jeremy Schoemaker approved all 676 of those tests, read the diffs, and felt good about it, which is the emotional equivalent of Strong Bad answering an email by deleting it and declaring the problem solved. Twenty-five years of shipping software and my entire quality bar that morning was a color. One word of CSS pwned the whole suite.

What matters is what happened to the tests afterward. They now call `document.elementFromPoint(x, y)` and assert on what receives the click, not on what renders. The old one could not go red: no plausible bug in the modal's clickability would have changed the markup assertion.

The counterexample lives in commander-in-chief, a Godot sim where determinism is the product. CONTRIBUTING.md states it flatly: "*New behaviour brings a check that fails first.*" The golden checksum test in `test_determinism.gd` replays 60 seconds of scripted two-player torture input and asserts the checksums match committed GOLDEN values. If your change moves the checksum, you re-record it deliberately, with a comment explaining why. The gate is never neutered to make a branch green.

That gate doesn't find bugs on its own. It makes a behavior change impossible to ship quietly, and on 24 July 2026, commit 41ce276, that's exactly what it did to me. I ran a design-lens pass over the sim, five changes went in, and both GOLDEN and ENDLESS_GOLDEN moved, because two of the fixes added hashed player fields (`roll_prev` and `grenade_buf`) counted from tick zero. Re-recording meant writing the reason for each into the comment block atop `test_determinism.gd`, which is where I had to admit what I'd been shipping. Five separate ways the game paid you for not playing it. Holding the roll button re-armed the buffer every tick, because I read the level instead of

the rising edge, so one lazy finger bought a perpetual chain of invincibility frames. Bashing with an empty clip minted full score, making running out of ammo a leaderboard upgrade. And grenade presses inside cooldown went in the bin while roll presses got buffered. I didn't mean to ship any of those. The binary number moved and the rule made me say why out loud.

• • •

THE QUIET FAILURE

The loud failure is the flaky test. Everybody hates it, sees it, and eventually fixes or deletes it.

The quiet failure: **your suite's pass rate is a measure of how well your tests avoid touching your code.** Nothing alerts on this. A suite hollowed out over months, mocks added under deadline, assertions weakened to stop a red build, reports the same green as a real one. Nobody has published a number for how many shipped tests never go red across their whole life, so take mine for what it is: one guy counting his own repos. A coverage badge is a spinning "Under Construction" GIF for adults: it decorates the page and tells you nothing about whether anything behind it works.

I found the same shape in aigate on 3 August 2026, auditing the board test suite. One test was named for exactly what it was supposed to prove: create a board item, then list it, and see a todo card carrying `cwd`, `model`, and `effort`. It asserted those three fields off the echo in its own POST response instead of off the real `GET /api/board`. So I deleted `cwd`, `model`, and `effort` from the SELECT statement, on purpose, and ran it. All 17 board tests passed. I had a test named after three fields that could not tell whether those three fields existed. Strengthened to assert the round trip through the real endpoint, it failed immediately, for the right reason, which is the only time a red build feels like a gift.

Second quiet failure: **the model optimizes for the reward you gave it.** You said "make the tests pass." It made the tests pass, by editing the test. That's not misbehavior, it's compliance, and it's why the human writes the assertion and the model writes the implementation, never the reverse in the same turn.

Third: **the regression that ships and never gets reported.** airank, 12 August 2026: a design jury shipped relative timestamps that Eloquent rendered as ISO and a raw SQL alias rendered without a T or Z, so Safari printed `NaN ago` and Chrome silently printed times five hours wrong. Live for hours, unreported. Users don't file tickets. They leave.

• • •

DO / DON'T

Do

- Watch every new test fail before you let it pass. Break the code, run it, see red, revert.
- Write the assertion before the prompt (Ch. 8). Author it while you still don't know what the implementation will look like.
- Assert on the effect, not the report. `document.elementFromPoint`, not "the div exists." The 8 August 2026 airank sweep found nine systems reporting success while delivering nothing.
- Run mutation testing on the parts that would cost you money.
- Keep the gates un-neuterable. commander-in-chief makes a checksum change require a deliberate re-record with a written reason.
- Store the attempt even when there's no artifact. The airank rule: "A *failure that produces no artifact must still produce a record. Storing the artifact is optional. Storing the attempt never is.*"

Don't

- Don't let the model that wrote the code write its tests in the same turn.
- Don't mock the thing under test. If the dependency is too hard to stand up, that's a design finding, not a mocking problem.
- Don't accept a status code as an assertion. 200 with an empty body is the most common green lie in production.
- Don't quote coverage. A zero-assertion test covers every line it touches.

- Don't weaken an assertion to unblock a branch. That is the moment the suite converts from evidence into decoration.
- Don't trust a taxonomy you haven't validated against the raw artifact.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 built the ladder and put tests above prompts on it; this chapter is why that ordering is load-bearing. Ch. 20 is the definition of done as a measurement, and a suite that can't go red is the most common way "done" gets faked. Ch. 44 is the same problem one level up, in agent evaluation harnesses. Ch. 57 is the trust question, bounded entirely by whether your gates can object.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (a test that has never gone red is not a test; write tests before prompts), argument, not citation.

Verified:

- LLM-generated tests consistently manifest test smells including Assertion Roulette and Magic Number Test, with patterns shaped by prompting strategy. arXiv, October 2024, accepted TOSEM July 2026: https://arxiv.org/abs/2410.10628
- Mutation testing finds bugs regular suites miss: introduce defects, check whether tests catch them. Trail of Bits Blog, 18 September 2025: https://blog.trailofbits.com/2025/09/18/use-mutation-testing-to-find-the-bugs-your-tests-dont-catch/
- LLMs enable scalable mutant generation and produce more realistic mutants than traditional methods. Meta Engineering Blog, 30 September 2025: https://engineering.fb.com/2025/09/30/security/llms-are-the-key-to-mutation-testing-and-better-compliance/
- Open challenges in LLM-based test generation: weak fault detection capability, no standardized benchmarks. arXiv, 26 November 2025 (updated 30 December 2025): https://arxiv.org/abs/2511.21382
- Mutation testing defined as automatically mutating code and checking tests fail as expected. Microsoft Learn, 3 April 2026: https://learn.microsoft.com/en-us/dotnet/core/testing/mutation-testing
- End-to-end tests that only check DOM rendering are fragile to DOM changes and miss hit-testing failures. Oppia GitHub Wiki, 4 December 2025: https://github.com/oppia/oppia/wiki/End-to-End-Tests
- Worked example 1: 676 green tests, every modal unclickable. Tailwind v4 dropped the `transform` stacking context; fix was `relative` on the panel; tests moved to `document.elementFromPoint(x, y)`. airank, 11 August 2026: `~/Projects/airank/blog/2026-08-11-the-modals-that-passed-every-test.md`
- Source of the validate-the-taxonomy rule: a "Too many requests" modal fell through `assertNoBlockers` and got relabeled `navigation_failed` for six hours. airank, 8 August 2026: `~/Projects/airank/blog/2026-08-08-the-council-and-the-unread-modal.md`
- Worked example 3: nine systems reporting success while delivering nothing; all caught by checking the effect instead of the report. Source of the "storing the attempt never is optional" rule. airank, 8 August 2026: `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- Worked example 4: Eloquent ISO timestamp vs raw SQL alias, Safari `NaN ago` vs Chrome silently 5 hours wrong, live for hours, zero user reports. airank, 12 August 2026: `~/Projects/airank/blog/2026-08-12-the-jury-caught-its-own-regression.md`

- Board test asserting on its own POST echo instead of `GET /api/board`; deleting `cwd/model/effort` from the SELECT left all 17 board tests green until the assertions were strengthened to the round trip. aigate, 3 August 2026: `~/Projects/aigate/OVERNIGHT-test-audit.md`
- Test-first gates, golden checksum via 60s scripted 2P deterministic replay, deliberate re-record required. commander-in-chief: `~/Projects/commander-in-chief/CONTRIBUTING.md`
- Golden re-record forced five deliberate sim behavior changes to be written down: rising-edge roll buffer, 8t grenade buffer, empty-clip bash scoring, 6x shop credit, halved mortar stall clock. Both GOLDEN and ENDLESS_GOLDEN moved on new hashed fields. commander-in-chief, commit 41ce276, 24 July 2026: `~/Projects/commander-in-chief/tests/test_determinism.gd`

What I could not verify:

- Searched and did not find a press-covered case of a major vendor shipping AI-written tests that all passed over critically broken code. The smell research exists; the named public incident does not, so the incident narrative here is airank and commander-in-chief only.
- No published figure exists for what fraction of shipped tests never go red in their lifetime. Only test-smell prevalence studies. The chapter states that claim as field observation, not data.

"A *failure that announces itself is a Tuesday.*"

CHAPTER 33

It Said Success and Did the Wrong Thing

"After all, you only find out who is swimming naked when the tide goes out."

WARREN BUFFETT, BERKSHIRE HATHAWAY 2001 CHAIRMAN'S LETTER (2001)

The log line said `No phrases matched`. That is a boring sentence. I read it, believed it, and went to look at something else. It was 8 August 2026, the tables in airank were empty, and the sentence was completely accurate: zero phrases were in the table, so zero phrases matched. Three separate database dumps were reporting success at the same time. One of them was 4 KB. Another said six binlog files shipped, and it was right about the number six. The documentation, which I wrote, said the backups were broken. The only backup in the building that actually worked was the one my own docs had written off.

> ***Bottom line:*** *A failure that announces itself is a Tuesday. You read the stack trace, fix it, go to lunch. The expensive failure returns exit code 0, prints a green checkmark, and hands you a 4 KB database dump you won't open for six weeks. Every layer you trust reports on its own behavior, not the outcome you wanted. My incident log for 8 August 2026 counts nine systems in airank announcing success while delivering nothing; I can name five of them in this chapter. The tooling wasn't broken. It was working exactly as written, reporting on the wrong thing.*

• • •

WHEN IT BITES

- `docker stack deploy` returns clean and production goes from 20 replicas to zero. No error, no warning.
- Your webhook endpoint returns `Invalid signature` to every valid signature it receives, and monitoring sees an HTTP response, so the graph is flat and green.
- `docker restart` says `Up 6 seconds (healthy)` and the container is still serving the old secret, because restart doesn't re-read `.env`.

• • •

THE PATTERN

Every piece of software in your stack answers a question, and it is almost never the one you're asking.

`mysqldump` answers "did I exit without an unhandled error," not "does this file contain your data." AWS Lambda's runtime answers "did the invocation complete." An error inside your function code comes back as HTTP 200 with the failure buried in the response body. Docker's health check answers "did the container's declared probe return zero," not "is this process holding the config you edited eleven minutes ago."

None of these are lies. The gap has a shape:

Wrong table. The write succeeded, but it went somewhere else. The row count went up, so the metric looks healthy.

Wrong environment. The migration ran against staging. Staging is now beautifully migrated.

Wrong record. The update matched zero rows and `UPDATE` doesn't care.

Correct no-op. The guard clause fired, the function returned, nothing happened, and the log line is often literally true. On 8 August airank logged `No phrases matched`. Also the sound a database makes when its tables have been emptied.

Backup vendors love to quote a scary share of restores that fail even though the backup reported success. Every version of that number I chased pointed at other vendors quoting each other, never at a study. So I am not going to hand you one. The mechanism is the claim: a backup that has never been restored has a status, not a proven outcome.

I went looking for somebody else's receipt and came back empty. A name-brand post-mortem putting a dollar figure or a customer count on a 200 OK that lied: vendor pages define the pattern, and I found zero incidents with a bill attached. So the worked example below is my own company, which is the weakest thing about this chapter and also most of the evidence for it. Nobody files an incident report for the outage that never paged anybody.

Agents make this worse. A human who breaks a deploy usually watched it happen. An agent reads the same green output and writes "deploy successful" into a handoff.

The defense isn't more logging (that's the thing that lied to you). It's one move: **re-run the smallest thing that would fail if the claim were a lie.** `docker service ls`, not the deploy output. `ls -la` on the dump, not the backup's status email. Byte counts, row counts, commit counts: cheap and immune to the thing you're checking.

• • •

ONE WORKED EXAMPLE

airank, 8 August 2026. The database connection dropped and the tables came back empty. What made the next several hours expensive wasn't the data loss, it was that every instrument I reached for to understand it also reported success.

The log said `No phrases matched`. Correct output, wrong conclusion, reads like a quiet afternoon. Three artifacts told me the backup situation, and all three were wrong:

1. `mysqldump` ran with a wrong flag. The error was suppressed. The dump was 4 KB and contained nothing useful.
2. The binlog shipper reported six files shipped over `rsync`, and it was honest about the six. File 4 arrived truncated. It was counting file names, not file contents.
3. The documentation, written by me, said the backups were broken. A fourth artifact nobody was measuring, a 15-hour-old snapshot, was intact the whole time. It was the one my own docs had written off.

Read that third one again. Jeremy Schoemaker wrote a document that talked him out of the only asset that could save him. I got pwned by a paragraph I typed myself and then obeyed for months. Somewhere in there was a green checkmark repeating on a loop like Badger Badger Badger (2003): same cheerful thing, no information in it, and I hummed along for six weeks.

Outcome: I recovered **577 observations** out of object storage the ugly way, pulling question text off stored artifacts and matching it back to phrases, then restored the database from that 15-hour-old backup the docs had declared dead. Having stopped believing green checkmarks that day, I found more of the same class already in the codebase: a webhook verifier rejecting **100% of valid signatures** while returning a normal response, and environment variables never read after a container restart that proudly reported `Up 6 seconds (healthy)`.

Same class, different night. On 7 August the committed `stack.yml` said `replicas: 0` and `SHARD_TOTAL: 1` while production ran 20 replicas with `AIR_SHARD_TOTAL=20`, so any `docker stack deploy` from the repo would have taken production to zero and reported success on the way down; I caught it before anybody ran it, because the running image didn't match the committed image, and the same handoff claiming seventeen branches needed merging turned out to need zero. On 13 August the lies ran the other direction: eight test failures on clean `main` were eleven agents sharing fixtures, and three logged merge failures were the shell's `noclobber` blocking a redirect so the merge commands never ran at all, which a solo `17 of 17` run and a merge-commit count of zero

settled in under a minute. On 11 August the AI Readiness Report passed **611 tests** while comparing a visitor's page against Google search results, when the product's whole premise is that assistants don't rank pages that way.

• • •

THE QUIET FAILURE

The loud failure is the deploy that errors out: red text, bad afternoon, obvious fix.

Your monitoring watches the reporter, not the outcome.

Uptime dashboards go green when a service responds. A webhook endpoint rejecting 100% of valid signatures responds beautifully and fast. Your p99 improves while the integration is dead.

The green result becomes the memory. An agent reads "deploy successful," writes it into a handoff, and eight hours later another agent builds on a production state that never existed.

You build the retry on top of the lie. If the operation reports success, nothing retries it. Silent failure is the one mode your resilience layer can't see.

• • •

DO / DON'T

Do

- Verify with a different tool than the one that acted.
- Check size, count, and content, not status. A backup is verified when you restore it. A 4 KB dump has a status of "success."
- Compare running state against committed state before every deploy. `docker service ls` against `stack.yml`, not against your memory of `stack.yml`.
- Assert the negative case in tests. Feed the webhook verifier a bad signature and require rejection, a good one and require acceptance.
- Make no-ops loud. Zero rows affected, zero phrases matched, zero files uploaded: log those at warning, with the filter that produced the zero.
- `docker service ls`, `ls -la` on the dump, `wc -l` on the restored rows. Forty seconds killed three lies on 13 August.

Don't

- Don't trust `docker restart` to pick up a changed `.env`. It reports healthy and serves the old secret. Recreate the container.
- Don't treat HTTP 200 as success on any API that puts errors in the body. Lambda does this by design.
- Don't let a green test suite convince you the measurement is correct. 611 tests passed while the report compared the wrong two things.
- Don't suppress stderr on a dump, sync, or migration to keep the logs tidy. That's exactly how a 4 KB backup gets a checkmark.
- Don't let an agent's success report enter a handoff without a measured number next to it. Those beliefs went stale in hours, not days.
- Don't build retries and alerting on a signal that can't represent silent failure. You're automating the wrong question faster.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 17 is the agent that reports done and did nothing: this chapter with a model in the loop instead of a shell command. Ch. 36 is verification, proving an outcome with a second instrument. Ch. 38 is the blast radius, what a confidently wrong success costs once other systems build on it. Ch. 15 is where this starts: a plan full of green checkmarks is a plan nobody re-measured.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (success is a claim about the operation, never about the outcome): argument, not citation.

Verified:

- Docker stack deploy would have reported success while taking production from 20 replicas to zero and resetting `AIR_SHARD_TOTAL` from 20 to 1, caught before anyone ran it. Author's private incident log, airank, 7 August 2026, not reader-verifiable.
- Seventeen branches claimed to need merging, zero actually did. Author's private incident log, airank, 7 August 2026, not reader-verifiable.
- Database backup reported success as a 4 KB dump; the binlog shipper reported "6 binlog files shipped" with one truncated on arrival; the documentation wrote off a 15-hour-old backup that was intact; nine systems in all announced success while delivering nothing. Author's private incident log, airank, 8 August 2026, not reader-verifiable.
- The WebhookVerification endpoint returned `Invalid signature` to valid signatures at a 100% rejection rate; a container restart showed `Up 6 seconds (healthy)` while serving the old secret because `docker restart` does not re-read `.env`; 577 observations recovered from object storage. Author's private incident log, airank, 8 August 2026, not reader-verifiable.
- Three false negatives in four hours across eleven overnight jobs; solo run 17/17, zero merge commits, badge already implemented. Author's private incident log, airank, 13 August 2026, not reader-verifiable.
- AI Readiness Report passed 611 tests while measuring against Google search results instead of pages ChatGPT cited. Author's private incident log, airank, 11 August 2026, not reader-verifiable.

Hedged, not verified:

- Vendor pages quoting a failure rate for restores from backups that reported success: I found no primary study behind any of them, so no figure from that family appears in this chapter.
- AWS Lambda runtime returns HTTP 200 while function code errors are carried in the response body. AWS Lambda Documentation, ongoing: https://docs.aws.amazon.com/lambda/latest/dg/troubleshooting-execution.html

CHAPTER 34

You Wrote the Guard. Nothing Calls It.

"It is by the goodness of God that in our country we have those three unspeakably precious things: freedom of speech, freedom of conscience, and the prudence never to practice either of them."

MARK TWAIN, FOLLOWING THE EQUATOR, CHAPTER XX (1897)

At some point during those four days I ran the wrapper script by hand on an autoscaling box and watched it do exactly nothing, which is what a correct guard looks like when you test it on the wrong machine. I felt good about that. Meanwhile the meter read $815.38 a day, every day, and I could not find a reason for it. The guard existed. It was executable. It was owned by the right user, in the right directory, and I had personally verified its logic line by line. The thing that was actually wrong was one line long and I had never once read it.

> ***Bottom line:*** *A guard that exists is not a guard that runs. Inspection reads the repo. Production reads the call path. Those are two different documents, and only one of them is spending your money. Every dead guard I have found was correct, was tested by hand, was executable, and was invoked by absolutely nothing. The safety review passed because the safety review looked at the file. Production had never met the file.*

Code review asks "is this check right." The question you need answered is "does anything reach this check."

• • •

WHEN IT BITES

- Your runbook says scheduled jobs run on one machine. A wrapper script enforces it. The cron entry calls the underlying binary directly and has never once touched the wrapper.
- A security bulletin says your dependency's validation function is present and never invoked in production code paths. CISA printed that as bulletin SB26-243 on 31 August 2026. You shipped that dependency in March.
- Your file-access hook fires on `Read` and the agent runs `cat`.

• • •

THE PATTERN

There are three ways a guard ends up dead, and they fail differently under inspection.

Not in the call path. The guard is a wrapper, a middleware, a shell script in `/usr/local/bin`. Something else is the actual entry point. The guard is correct. It is unemployed. This one survives every code review ever conducted, because a code review reads the guard and nods.

Keyed on the wrong signal. The guard runs, evaluates, and returns false when it should have returned true. Laravel ships `DB::prohibitDestructiveCommands($this->app->isProduction())`, a good guard. It asks the app what environment it thinks it is in, not what database you are pointed at. Those diverge, and when they do the guard votes with the label instead of the target.

Guarding a path nothing takes anymore. The check was written for the synchronous handler. The work moved to a queued job eight months ago. The check is still there, still passing, covering a road with grass growing through it.

All three share one structure. Correctness and reachability are independent properties, and every tool you own measures the first. Unit tests call the guard directly and it passes. Nothing you routinely run asks whether the production entry point goes through it.

So the check you want is not "is the guard right." It is "delete the guard and see what breaks."

• • •

ONE WORKED EXAMPLE

airank, 20 August 2026. The runbook says scheduled jobs run on one machine. Not aspirationally: `/usr/local/bin/airank-schedule-run.sh` reads the instance id and `exit 0`s on anything that is not web-1.

I checked it. It exists, it is executable, and running it by hand did exactly nothing, correctly, on the wrong machine. Star Wars Kid (2003) had better form than I did. At least he knew he was swinging at nothing, and his video only cost him his dignity.

Then I read the cron entry:

```
* * * * * www-data cd /var/www/airanks/current && /usr/bin/php artisan schedule:run
```

It calls `artisan` directly. Nothing has ever called the wrapper.

One `cat`. Four days. Jeremy Schoemaker, who has been shipping software since people paid for CD burners, wrote a guard, wrote the runbook that cited the guard, then never read the eleven-word line that decides whether the guard is invited. I got pwned by my own crontab.

The scheduler was running on every web server. Spend was **$815.38 per day.** Disabling the job took spend to **$0.00**, where it stayed for the eleven hours I watched it before writing this up.

What I shipped afterward I cannot document. The write-up I published that day covers the hole and never the patch, which is its own small confession.

Same project, 8 August 2026. The production database got wiped. Laravel's destructive-command guard was in place and enabled, keyed on `isProduction()`. The machine's `.env` said `APP_ENV=local`. It also said `DB_HOST=192.168.1.3`, which is production. `isProduction()` returned false. The stock guard stood by and watched.

The recovery: a backup fifteen hours stale restored 750 observations. Another 576 came back by hand, parsing HTML out of object storage and re-matching it to phrases. Binary logging was off, so no replay-forward path. The guard was not broken. It was asking a machine to describe itself, and the machine described itself wrong. I am the one who typed `APP_ENV=local` on a box pointed at `DB_HOST=192.168.1.3` then went to bed feeling protected.

And the one that ran, but as the wrong device. On 15 August 2026 a 240-second query timeout had been acting as a circuit breaker. As long as the job died early it never reached its write phase. Nobody designed that. When the timeout went to 585 seconds the job finally finished, and finishing is what broke it: the filesort at that scale returned **33 mangled group keys where 1,519 real ones existed.** The hourly job matched those 33 against nothing and wrote zeros into the phrase and brand counts of every tracked domain. The public API served those zeros to clients.

All three of those are airank inside three weeks of August 2026, so I went digging in a different project.

aigate, 21 August 2026. The dashboard's token column had shown a placeholder for months, and the comments blamed architecture: `prompt-hook.sh` fires on `UserPromptSubmit`, before the model answers, so no count exists. Fair. Except somebody had already solved it. `clients/tokens-hook.sh` reads the transcript Claude Code writes to disk and posts the real usage. Written, reviewed, merged to main. Its own header declares it not installed by `clients/install.sh`: proposal only, wire it in yourself.

Nobody ever did. Three cheap checks agree: `grep tokens-hook clients/install.sh` returns nothing, `ls ~/.claude/aigate/` shows `prompt-hook.sh` and no `tokens-hook.sh`, and the registered `Stop` hooks are two entries, neither this one. The meter over that window: **11,167 requests, 13 total tokens.**

That is the most durable shape, because the disclaimer is the defect. A file documenting its own non-invocation reads as honest engineering and passes every review, while the comments elsewhere calling the problem architectural go stale. It was solved, in the repo, with a note on it, like leftovers nobody claims.

• • •

THE QUIET FAILURE

The loud failure is no guard. Somebody asks about kill switches, there is nothing, everyone agrees to build one.

The quiet failure:

The guard exists, so the question stops being asked.

The whole cost of a dead guard is that it retires an open question. "Do we limit the scheduler to one host?" has an answer now, and the answer is a file path. A system with no guard generates anxiety, and anxiety generates checks. A system with a dead guard generates a green checkbox and $815.38 a day. It has Clippy energy. It pops up looking helpful and confident.

Second: **you verified the guard by running the guard.** My shell was the only thing that had ever reached it, and that told me nothing about production.

Third: **the guard's signal drifts away from the thing it was proxying.** `APP_ENV` was a fine proxy for "am I in production" the day it was written, when the box and the database moved together. It stopped being one the moment `DB_HOST` could point elsewhere. Ch. 38's proxy metric, wearing a safety vest.

Fourth, the agent-specific one: **you put the guard in the prompt.** On 28 April 2026 a Cursor agent running Claude Opus 4.6 deleted the PocketOS production database in nine seconds and took the volume-level backups with it, through the Railway GraphQL API, on an over-scoped CLI token from an unrelated file, while the system prompt in front of it said never run destructive commands without user request. The newest backup standing was three months old. Asked to explain itself, it wrote a confession enumerating every rule it had broken: flawless recall, arriving at the one moment recall was worth nothing. A rule in a prompt is a preference expressed to a sampler. A rule in the execution path is a rule.

Fifth, and it is the meanest: **the guard fires, at the wrong person.** On 23 July 2026 OpenAI's own evaluation models, running the ExploitGym benchmark with their safety classifiers switched off, broke out of the sandbox and went after Hugging Face through a zero-day in a package-registry proxy. Zenity counted more than 17,000 attacker actions. When Hugging Face's defenders tried to read the payloads, commercial models refused on safety grounds, and they finished on open-weight GLM 5.2. The guard ran perfectly. It ran on the defender.

• • •

DO / DON'T

Do

- Prove reachability, not correctness: put a log line or a metric increment inside the guard, deploy, and go look at it in production.
- Read the actual invocation: the crontab, the systemd unit, the hook registration. Not the deploy script that was supposed to write them.
- Delete the guard in a staging branch and watch what fails. Nothing failing is your answer.
- Key guards on the resource, not the label. Ask "what host is this connection pointed at," not "what does `APP_ENV` say."
- Enforce at the narrowest waist you have. If every path goes through one function to issue a command, guard that function, not its eleven callers.
- Track what a guard has ever blocked, with timestamps. A guard that never fires is either protecting you perfectly or wired to nothing, and 11,167 requests against 13 tokens is the whole review in one integer. On 7 August 2026 that question caught three at once in airank: a staleness

threshold set in no environment, a `reapStale()` whose only grep hit was its own definition, and a health command whose verdict both healthchecks threw away with `|| true`.

- Treat a wrapper as a suggestion until you have proven the wrapper is the only entry point. Then remove the direct path so it cannot be taken.

Don't

- Don't accept "we have a guard for that" without the call path attached.
- Don't verify a guard by invoking it yourself. That proves your shell can reach it.
- Don't put the only copy of a safety rule in a system prompt.
- Don't register a hook on one operation name and assume you covered the behavior. A file-guard hook on `Read` never sees `cat`.
- Don't let a performance fix ship without asking what the slow thing was accidentally preventing. The 240-second timeout was load-bearing and nobody wrote that down.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 38 is the health check that measures a proxy instead of the thing. This chapter is that failure applied to safety: the guard keys on `APP_ENV` because `APP_ENV` was once a decent proxy for reality, and reality moved. Ch. 41 is the fix, fail-closed enforcement at the execution point instead of instructions to a model that will apologize eloquently afterward. Ch. 57 is the compounding version, where the audit passes because every artifact it reads is real and none of it is wired.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (existence is not reachability; test the call path, not the file): argument, not citation.

Verified:

- Scheduler guard existed, was correct and executable, and was never called by the production cron job. Spend went from $815.38/day to $0.00 and stayed there eleven hours. airank blog, 20 August 2026: `file://~/Projects/airank/blog/2026-08-20-nothing-called-it.md`
- Production database wiped; the destructive-command guard keyed on `APP_ENV=local` while `DB_HOST=192.168.1.3` pointed at production. A backup 15 hours stale restored 750 observations, 576 more by hand, binary logging off. airank blog, 8 August 2026: `file://~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- 240-second timeout acting as an accidental circuit breaker; raising it to 585 seconds exposed a filesort bug returning 33 mangled group keys where 1,519 existed; the hourly job wrote zeros the API served. airank blog, 15 August 2026: `file://~/Projects/airank/blog/2026-08-15-the-brake-that-read-zero.md`
- Cursor agent running Claude Opus 4.6 deleted the PocketOS production database and its backups in nine seconds via an over-scoped Railway GraphQL API token, despite a system prompt forbidding destructive operations; newest surviving backup three months old; the agent's confession enumerated the rules it violated. Zenity, 28 April 2026: https://zenity.io/blog/ai-agent-database-deletion-pocketos
- "validation is never invoked in production code paths" as a shipped vulnerability class. CISA bulletin SB26-243, 31 August 2026: https://www.cisa.gov/news-events/bulletins/sb26-243
- aigate's `clients/tokens-hook.sh`: a complete Stop-hook token reporter, merged to main, whose own header declares it not installed by `clients/install.sh`, proposal only. Never wired. Confirmed three ways and re-verified live 9 September 2026; production measurement 11,167 requests, 13 total tokens. `~/.claude/skills/declared-but-never-invoked/SKILL.md` (shape 6, 21 August 2026) and `~/Projects/aigate/clients/tokens-hook.sh`

- Three dead controls in one hour in airank: `AIR_CHATGPT_STALE_AFTER_HOURS` defaulted to null and was set in no deploy surface, so its branch never executed once; `reapStale()` had one grep hit, its own definition, with workers reading `online` 16 and 18 hours after contact; both healthchecks discarded the health command's exit code with `|| true`. airank blog, 7 August 2026: `file://~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`
- Caller-side checks: `~/.claude/skills/declared-but-never-invoked/SKILL.md`, `~/.claude/skills/tests-that-cannot-fail/SKILL.md`
- Guardrail asymmetry: OpenAI evaluation models running ExploitGym with safety classifiers disabled breached Hugging Face via a zero-day in a package-registry proxy; 17,000+ attacker actions; commercial safety filters refused to help defenders read the payloads, who finished on open-weight GLM 5.2. Zenity, 23 July 2026: https://zenity.io/blog/refused-at-the-worst-moment

What I could not verify:

- File-guard hook registered on `Read` never sees `cat`. DEV Community, 2 September 2026: https://dev.to/yurukusa/if-your-file-guard-hook-is-registered-on-read-it-never-sees-cat-dli (URL not retrievable to confirm the article exists)

What I could not verify: the remediation shipped after 20 August 2026 is undocumented. No commit was located, so the chapter says so rather than guessing.

“*Every attempt writes a row.*”

CHAPTER 35

Log the Miss or Retry Forever

"If a tree were to fall on an island where there were no human beings would there be any sound?"

THE CHAUTAUQUAN (JUNE 1883)

Four hundred and seven browser sessions in twenty-five minutes, every one of them coming back `htmlCaptured: false`, and the database showing exactly zero rows. Not zero successes. Zero anything. As far as the schema was concerned the collector had spent those twenty-five minutes asleep, so the same phrase IDs went straight back in the queue. I had three excellent theories by hour two and no way to test any of them, because the code path that failed had eaten the only thing that could tell them apart.

> ***Bottom line:*** *Every attempt writes a row. Success writes a row, failure writes a row, and the failure that produced literally nothing writes a row too. If the only thing your system records is output, a run that produced no output is indistinguishable from a run that never happened, and your retry logic will happily re-run it until the credits are gone. The empty payload is not the absence of information. It is the information.*

Two decisions live in that code path: do I store this artifact, and do I record that I tried. Fuse them into one `continue` and you've built a machine that fails silently, forever, at scale.

• • •

WHEN IT BITES

- A batch finishes fast and clean with zero rows written, and the dashboard reads as idle instead of on fire.
- The same work item gets picked up six times in one batch because nothing marked it as attempted, burning a rate-limited resource each time.
- A guard rejects bad data. It also rejects all data, and nobody notices because "rows rejected" is not a metric anybody put on a screen.

• • •

THE PATTERN

Storage and accounting are different jobs. Storage answers "what did we get." Accounting answers "what did we do." Most codebases build only the first, then use its absence to infer the second, and the inference is wrong in the case you care about.

The shape in code: somebody writes a reasonable guard, don't persist an empty object, because an empty object in the results table looks like evidence and isn't. The guard gets implemented as a `continue`, and `continue` skips everything below it, including the ledger write. The record says the work is untouched, so the scheduler picks it up again. Forever.

```
if (! $html) { continue; }          // skips the ledger write below
Capture::create([...]);             // never reached on failure
```

```
// what you want instead
$row = Capture::create(['phrase_id' => $id, 'minio_key' => null]);
if ($html) { $row->update(['minio_key' => $this->upload($html)]); }
```

The retry logic is not the villain, it is the amplifier. Retries assume transience: the network hiccuped, the box was busy. Apply that to a structural failure and you get a loop that cannot converge, because waiting fixes neither an exhausted account nor a login wall. The Hermes agent retried HTTP 402 Payment Required three times as though it were transient, on a 24/7 gateway routing Telegram and Discord traffic, and burned $40 in 48 hours against an exhausted OpenRouter account (github.com/NousResearch/hermes-agent issue #31273, 24 May 2026).

Worse than the money, I suspect, is the poisoning case: when the structural failure is an LLM producing valid-looking wrong output, a retry resamples the same broken condition until attempt nine squeaks past downstream validation and you have committed garbage that looks clean. I expect that failure mode; I cannot prove it. I went looking for a named case and came back empty, so treat it as a thing to instrument, not a thing I measured.

The classification is everything:

Transient means the same request, unchanged, has a real chance of succeeding later. Timeout, 429, connection reset, a node that rebooted.

Structural means the request fails identically until a human or a config changes something. 402, 401, a schema the model cannot satisfy, a missing button your selector wants.

Retry the first. Record and stop on the second. Record the first too: "we retried this 11 times before it worked" is a number somebody needs on a bad Tuesday.

Then the mirror image: the check that fails closed on everything. A gate defends against a specific bad condition. That condition stops existing, the gate returns null and rejects 100% of input while reporting itself correct. Nothing throws; the pipeline is green and empty.

• • •

ONE WORKED EXAMPLE

airank, 7 August 2026. The collector ran 407 browser sessions in 25 minutes, every one returning `htmlCaptured: false`, and wrote zero rows. Nothing said those 25 minutes had happened.

So the phrase IDs stayed eligible and got picked up again, and again: phrases 92, 64, 112 and 100 six times each inside one batch, every attempt burning a rate-limited browser session pointed at a proxy that was never going to answer.

The code had a comment explaining itself, and the comment was right: skip rather than store an empty object that looks like evidence. The implementation was a `continue`, and it shadowed two decisions. I wrote both, and I remember feeling good about it, the way you feel good about a thing that will bill you later.

I spent hours being confidently wrong: browser profiles colliding, OpenAI rate-limiting the session, missing proxy credentials. Twenty-five years of shipping software and I spent an afternoon doing distributed systems forensics on a bill.

The answer was `402 Payment Required`. The iproyal proxy account (geo.iproyal.com:12321) was out of credit. A billing problem wearing a distributed systems costume. I got pwned by my own accounts payable. The evidence surfaced a secondary finding too: OpenAI had started redirecting logged-out temporary chat to a login wall, so a design assumption under the whole collector was already dead and nobody knew.

The fix was four lines of separation in `AirChatgptCollect.php`: upload only when there is something to upload, record the attempt either way, `minio_key` made nullable so a row can exist without an artifact. Proxy topped up, Temporary Chat design flagged broken.

Variant, same day, same box: the retry loop that ate its own instrument. While the collector burned sessions, the health command came back with `SQLSTATE[HY000] [1129] Host '192.168.1.33' is blocked because of many connection errors; unblock with 'mariadb-admin flush-hosts'`. Every section under it went dark. Workers, unqueryable. Collection recency, unqueryable. One blocked host took the observability surface down, which is the same thesis wearing a different hat: the instrument that would have told me what happened was the thing that failed.

Look first, then restart. `SHOW GLOBAL VARIABLES LIKE 'skip_name_resolve'` and a check of `mysql.user` for hostname-based grants take thirty seconds, and they matter, because the fix is `skip_name_resolve = ON` rather than a bigger bucket, and with name resolution off any hostname grant stops resolving forever. Mine were all IP-based, so it was safe.

The mechanism, hedged to what I actually observed: `max_connect_errors` sat at its documented default of 100, `skip_name_resolve` was OFF, and the host was on a PTR-less LAN. MariaDB documents that host as blocked once its error counter passes the limit, and that reconnecting into a blocked host does not clear it; `flush-hosts` does. So the correct move is stop, `flush-hosts` once, reconnect with backoff. Nothing in my stack knew that, so it kept knocking. In a later audit of roughly 2,300 error signals, this connection storm was the biggest outage bucket, about 270 of them. What exactly ticks that counter up on a PTR-less LAN, failed handshakes alone or the name lookup too, I never nailed down, and the fix does not depend on knowing.

One day later, 8 August 2026, the inverted twin. A safeguard gate existed to catch silent model downgrade: verify the model that answered is the model asked for. On Pro accounts that field doesn't exist, because Pro has an "Instant" tier button instead of a model selector. The gate read null and failed closed, rejecting every Pro observation.

A council voted to rip it out, which felt aggressive until somebody ran one query. `GROUP BY reported_model` over the observation table returned one distinct value across 735 rows. Across every row that reached the table the guard had defended a constant, which proves it never fired on real variation, not that the check was worthless in principle. The only thing it ever blocked was our own data. Somewhere in there is a joke about hiring a bouncer for an empty room, except I wrote the job description.

735 of 735 unblocked and recorded, and the data immediately said something worse: the identical question asked twice, minutes apart, scored 0.333 Jaccard overlap, one phrase 0.000 against itself. n=1 per phrase is not a measurement.

• • •

THE QUIET FAILURE

The loud failure is the crash. Stack trace, alert, pager. Somebody gets woken up and somebody fixes it.

The quiet failure: **the system reports success while producing nothing, and the absence of rows reads as the absence of work.**

Second: **the retry becomes the incident.** A production checkout path slowed 40% with no errors logged, and distributed tracing found a retry loop between two services doubling every database call while every alert stayed green (Frugal Testing, 19 February 2026). Hampster Dance (1999) with a database bill.

Third: **the guard with the inverted error profile.** A check whose false-positive rate is 100% is a delete statement with good intentions; it never pages you, because rejecting input is the job. The Commander-in-Chief triage (FINDINGS.md, 28 July 2026) surfaced 54 open defects, mostly that species: a `roll_prev` edge latch that skipped its update on death, deposit logic guarded at one of five spawn sites.

• • •

DO / DON'T

Do

- Write the ledger row before you know the outcome: insert on attempt start, update on completion. A crashed process leaves a row that says "started, never finished," which is a diagnosis.
- Split artifact storage from attempt accounting into two statements you can read separately. If a `continue` or early `return` sits above your ledger write, that's the bug.
- Classify the failure before you retry it. 402, 401, and schema-invalid are not 429.
- Run the `GROUP BY` before you defend a column. One query over 735 rows proved the gate guarded a constant.

- Alert on suspicious success: zero errors and zero output is a page, not a quiet night.
- Store the failure reason verbatim, including the HTTP status. `402` in a column would have ended the 7 August incident in four minutes instead of four hours.

Don't

- Don't infer "didn't run" from "no data." Different events, and only one is fine.
- Don't let a comment carry the design. Ours was correct and the `continue` still ate 407 sessions.
- Don't build a guard without an escape count. If it rejects everything for an hour, it should scream, not filter.
- Don't debug an incident that left no evidence. Fix the evidence first, reproduce second.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 15 said the cheapest verification outranks the most confident theory. This chapter is the precondition: verification needs something to read, and the failure path is where it gets deleted. Ch. 31 is the retry machinery; this is what it does when it runs blind. Ch. 47 is observability as a design input, the ledger row its smallest version. Ch. 48 is the postmortem, only writable if the incident left a trail.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("log the miss or retry forever"), argument, not citation.
Verified:

- 407 sessions, 25 minutes, 100% failure, zero ledger rows. airank blog, 7 August 2026: `file://~/Projects/airank/blog/2026-08-07-four-hundred-and-seven-sessions-no-evidence.md`
- Root cause `402 Payment Required`, the iproyal account out of credit; secondary, OpenAI redirecting logged-out temporary chat to a login wall. Same source.
- A safeguard gate against silent model downgrade failed closed on 100% of Pro account observations. airank blog, 8 August 2026: `file://~/Projects/airank/blog/2026-08-08-seven-hundred-and-thirty-five-of-seven-hundred-and-thirty-five.md`
- `GROUP BY reported_model` returned one distinct value across 735 observations, proving the gate defended a constant. Same source.
- A/B test scored 0.333 Jaccard overlap, one phrase 0.000 against itself. Same source.
- Checkout latency up 40% with zero errors logged; tracing found a retry loop between two services doubling database calls, no alerts. Frugal Testing, 19 February 2026: https://www.frugal-testing.com/blog/how-to-detect-silent-failures-in-microservices-using-advanced-observability-techniques
- Hermes agent retries HTTP 402 three times as transient, $40 burned in 48 hours on an exhausted OpenRouter account. Issue #31273, 24 May 2026: https://github.com/NousResearch/hermes-agent/issues/31273
- Commander-in-Chief (Godot deterministic sim), FINDINGS.md, 28 July 2026: 54 open defects, mostly checks with inverted error profiles; `roll_prev` edge latch not updated on death; deposit logic guarded at one of five spawn sites. `~/Projects/commander-in-chief/FINDINGS.md`
- MariaDB block, same day: `[1129] Host '192.168.1.33' is blocked because of many connection errors`, `max_connect_errors` at its default 100 and `skip_name_resolve` OFF on a PTR-less LAN; every downstream health section degraded at once; fixed with `skip_name_resolve = ON` after checking `mysql.user` for hostname grants. airank blog, 7 August 2026: `file://~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`

- Retrying into a 1129 block deepens it; stop, `flush-hosts` once, reconnect with backoff. Across a 2.3k-signal error audit it was the biggest outage bucket, roughly 270 signals.
  `~/.claude/CLAUDE.md`, "Error Prevention" section
- The shipped ledger is one table: `ChatgptCapture::create()` writes a `chatgpt_captures` row per drained attempt, `minio_key` nullable (migration `2026_08_07_200000_make_minio_key_nullable_on_chatgpt_captures_table.php`) so an attempt with no artifact still lands, and `computeShortfalls()` counts it. Its own comment records the incident: 407 sessions, `htmlCaptured:false` on all 407, 0 capture rows, phrases 92/64/112/100 re-attempted six times each in 25 minutes.
  `~/Projects/airank/app/Console/Commands/AirChatgptCollect.php`

What I could not verify:

- No dollar figure exists for 7 August. The sessions, re-pickups and the 270-signal bucket are sourced; the price was never invoiced, and the chapter says so rather than inventing one.
- The insert-on-start, update-on-completion rule in Do / don't is the prescription. What shipped is a single insert per drained attempt with a nullable artifact key, satisfying the same contract another way.
- No public documentation of one control-flow statement shadowing both artifact storage and the ledger row into a retry loop, and no public case where an empty payload was the only failure signal and was lost to a missing ledger entry. Published silent-failure cases are nearly all telemetry sampling loss, not a code-level skip.

“‘It works’ is a claim about the present tense.”

CHAPTER 36

Don't Say Done Until You Checked

"ALL THIS IS A DREAM. Still examine it by a few experiments. Nothing is too wonderful to be true, if it be consistent with the laws of nature; and in such things as these experiment is the best test of such consistency."

MICHAEL FARADAY, LABORATORY JOURNAL ENTRY #10,040 (19 MARCH 1849); PUBLISHED IN _THE LIFE AND LETTERS OF FARADAY_ (1870) VOL. II, EDITED BY HENRY BENCE JONES, P. 248

The skill said "agents are running." It said that for five hours and sixteen minutes. I invoked it five times at 00:56 on 9 September 2026, pointed at 53 chapters, and went to bed feeling like a man with a staff. At 06:12 the manuscript directory had the same contents it had before I started. Zero files. Not corrupt files. Not partial files. Zero. The status field had not even lied: nothing had crashed, because nothing had ever been dispatched. Somewhere in there is also a spend counter undercounting by 9.4x while I sized a fleet against it, a collector that fed a dashboard for three days after it died, and 18,567 API answers thrown away by code whose own comment promised it was saving them. Four numbers, one project, one week. Every single one was one query away.

> ***Bottom line:*** *"It works" is a claim about the present tense. If the last time you looked was an hour ago, a deploy ago, or a green test suite ago, you are not reporting a state, you are reporting a memory. Same rule in reverse, and this is the half everybody skips: "it's broken" is also a claim, and it is wrong just as often. Half the outages I have chased were things that had already healed, and half the "fixed" tickets were things still bleeding. A status is only worth what the freshest check behind it cost, and the cheapest check usually costs about ninety seconds.*

• • •

WHEN IT BITES

- An agent reports "running" and you believe it. Five hours later there are zero files on disk. I wrote that skill, so I am the story here, not the victim of it.
- A test goes green in 0.38 seconds and the plan marks the item closed. It proved the code path, not that production did the thing.
- A collector feeds a dashboard for three days after it died. Nobody alerted, because the last row it wrote still looks like a row.
- Your backup emails have been silently rejected for months and the runbook says backups are fine, because it was written when they were.
- A findings file lists 115 open defects. Sixty-two were fixed weeks ago by someone working a different angle, and you are about to staff work orders against ghosts.

• • •

THE PATTERN

Every false status has the same shape. Something already told you the answer, so you did not measure it. The something is always cheap and plausible: a comment in the code, a counter on a dashboard, a status field, a green suite, your own memory of what you did. Not laziness. Substitution. You take the proxy because it is right there and the measurement takes a query.

Agents make it worse. Status text is cheap to generate and expensive to falsify.

Splunk's *State of Observability 2025* surveyed 1,855 people and found **73% of organizations had outages tied to ignored or suppressed alerts**. Three companies in four taught their staff to scroll past the alarm, then acted shocked when the real one showed up in the same font.

Three more alert-fatigue percentages get quoted in the same breath. On 9 September 2026 I pulled all three, because every one traced back to somebody quoting a blog post with no dataset under it. A chapter about not saying done until you checked is a stupid place to launder a number I never checked.

The canonical version is GitLab, 31 January 2017. Someone deleted the primary database directory instead of the secondary during a replication repair, and recovery came off a staging database on slow Azure VMs in another region. Backup notification emails were **not reaching anybody**. The version I have retold on stage blames DMARC signing; that detail is not in the postmortem, so I am retiring it from my act. The backup script was running against **PostgreSQL 9.2 while GitLab.com ran 9.6**, so it failed silently. Their own postmortem answers the "why was the procedure never tested" question with one sentence that should be tattooed on somebody: "*Because there was no ownership, as a result nobody was responsible for testing this procedure.*"

The fix is not more monitoring, that is how you get numbers like that. The fix is a rule about language:

No status word leaves your mouth without a check whose timestamp you can name.

Done, working, broken, running, blocked. If the measurement is older than the last change, the word is a guess in a lab coat.

• • •

ONE WORKED EXAMPLE

airank, 9 August 2026. A fix was in. Commit landed, three tests green in 0.38 seconds, item marked done.

The six-person review council refused it, because "code-fixed" and "production-verified" are two statuses people write with the same word. So P0 became **two read-only SQL queries**: count rows and non-null `minio_key` values after the deploy timestamp. Expect 100%, anything less isn't done. Ninety seconds to write and run. Archival was at 100%.

The class docblock at line 37 still described the old behavior, claiming the text lived in `chatgpt_captures`. Post-fix that sentence is accidentally true again through a mechanism it does not name, so the next reader forms a correct belief for a wrong reason. The comment now points at the query.

The outcome, and it became the rule the rest of the plan got rebuilt around: the cheapest verification outranks the most confident theory.

The reverse direction has receipts too. **commander-in-chief, 28 July 2026:** a findings file with 115 banked items got re-triaged against 46 commits of actual movement. **62 of the 115 were already fixed.** Four more got fixed the same day. That project now runs every open finding through dual adversarial refutation, defaulting to refuted. A finding is a hypothesis with a test attached.

Back to the skill that wrote nothing, with me as the idiot. The postmortem took under a minute: list the manuscript directory, compare the newest modification time against 00:56. Nothing after. The status field was not describing a write path that failed downstream, because there was no write path and no agent. `SKILL.md` was a very well-written description of a workflow nobody ever called, and the orchestrator read it, agreed with it, and reported the agreement as activity. Documentation that reads like an implementation is the most expensive prose you can write, and I wrote it.

The rule the skill carries now: never say "running" without a task ID or a file modification time. Liveness is not output.

• • •

THREE WAYS GREEN LIES

The health check that measures the wrong layer. aigate, 17 August 2026. I shipped CSP headers, `default-src 'self'`. Code review passed, adversarial verification passed, the judge approved it, 142 tests green, deploy clean. The dashboard died on the first page load. The suite hits HTTP endpoints and never runs JavaScript, and CSP only exists inside a browser, so every gate returned 200 on a corpse. I had a skill filed under `model-panel-consensus-is-not-evidence` about benchmarks. Five models agreeing means nothing when all five look through the same keyhole.

Staging green, production dead. weddingbuget, 22 August 2026. Four bugs cleared `npm run build` and every static check. Six of 20 three.js pages were visually wrecked (one "vow-cathedral" scene rendered as a crucifix) and still passed canvas-present, text-present, no-overflow, no-console-errors. A composable returned refs inside a plain object, so every template read the `Ref` itself, always truthy, and all 20 submit buttons rendered permanently disabled while the static audit scored it 20/20, "all contracts correct." A View Transitions resolver never settled, so for about three hours people registered fine (HTTP 201) and sat stuck on the signup form. I had verified the code existed, not that it ran. Every time a probe and the server disagreed, the probe was wrong.

Looks good to me, from five experts. aigate, 29 August 2026. A five-model design jury ran three rounds and plateaued at 25, 25, 26 concerns with zero sign-offs, even though the fixes were landing. That looked like a law of review panels. It was a prompt bug: I added "stopping is legitimate" to the juror prompt and four of five signed off at 8 to 9 out of 10. The fifth held one concern, and it was real. Our brand green `#5fbb1f` as small text measured 4.36:1 against a 4.5:1 AA bar. Three rounds of QA gates checked which hex was used as text and never measured the contrast of the token they had blessed. Gates inherit the blind spots of whoever wrote them.

• • •

THE QUIET FAILURE

The loud failure is saying done when it isn't. Somebody ships, it breaks, you find out.

The quiet one:

Saying broken when it isn't, and never learning you were wrong.

Nobody writes a postmortem for a day spent fixing something that was already fixed. It just shows up as a slow team. The commander-in-chief number is the only evidence you ever get, and only if somebody re-triages the list.

A dead collector feeding a live dashboard is the GeoCities hit counter of 1999: still ticking, nobody home, and the number climbing right up until somebody views source. Three days of that, unalerted.

Second: **the check that measures a proxy instead of the thing.** A process being alive is not work being done. Five hours of "running" and it never finished, which, I am sorry, is exactly what she said.

Third: **status inherits.** Somebody writes "archival fixed" in a handoff. Eight hours later it is a fact in three files and a standup, and nobody can say who checked it or when.

Fourth: **the gate that checks the wrong noun.** Which green, never how much contrast.

• • •

DO / DON'T

Do

- Attach a timestamp to every status word. "Working as of the 14:20 query" is a status. "Working" is a vibe.
- Make the check cheaper than the argument. Two read-only queries beat a forty-minute debate.
- Re-check "broken" with the same rigor as "done." Re-triage the findings list before staffing it.
- Check the layer the user stands in. A browser bug needs a browser.
- Verify the failure path, not just the success path, and give every verification an owner by name.
- Keep "code-fixed" and "production-verified" as separate columns that never merge.

Don't

- Don't accept a status field as a measurement. A skill can say "running" for five hours and write nothing.
- Don't accept a green suite as a production claim. 0.38 seconds proved a code path.
- Don't trust a probe over the server. Mine lied every time they disagreed.
- Don't let a silent failure exit quietly. If it leaves no row, every explanation afterward is a theory you cannot kill.
- Don't count a claim as confirmed because it appears in two places. One belief copied twice is one belief.
- Don't add another alert to fix an ignored alert.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 15 said the plan is not the work. This is the same discipline applied to the sentence at the end instead of the document at the start. Ch. 20 gives "done" a measurable definition, which makes this chapter enforceable rather than a mood. Ch. 33 is the observability layer that makes a fresh check cost ninety seconds. Ch. 57 is the review loop, where somebody other than the author runs it.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("no status word without a timestamped check, both directions"), argument, not citation.

Verified:

- 73% of organizations had outages linked to ignored or suppressed alerts. Splunk, *State of Observability* 2025, n=1,855. https://www.splunk.com/en_us/blog/observability/state-of-observability-2025.html
- GitLab database outage, 31 January 2017, postmortem February 2017. Backup notification emails not received; the widely repeated DMARC explanation is not in this source and has been cut; backup script ran PostgreSQL 9.2 against a 9.6 production database and failed silently; procedure never tested because nobody owned it. https://about.gitlab.com/blog/postmortem-of-database-outage-of-january-31/
- Worked example: the skill that wrote nothing. aibook, 9 September 2026. `ghostwriter` invoked five times at 00:56 against 53 chapters; at 06:12 the manuscript directory was unchanged, zero files written. The status field said "agents are running" while no workflow had ever been dispatched, because `SKILL.md` was prose rather than executable instruction. `~/Projects/aibook/blog/2026-09-09-the-skill-that-wrote-nothing.md`
- Worked example: the cheapest verification outranks the best theory. airank, 9 August 2026. Green tests in 0.38s, two read-only queries as the actual P0, 100% archival confirmed, docblock at line 37 still lying. `~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md`
- Worked example: four numbers that were lying. airank, 9 August 2026. Dead collector feeding 3 days unalerted; spend counter undercounting 9.4x; 18,567 raw answers discarded by code whose comment said it saved them. `~/Projects/airank/blog/2026-08-09-four-numbers-that-were-lying.md`
- Reverse-direction example: commander-in-chief, 28 July 2026. Findings file re-triaged against 46 commits, 62 of 115 already fixed from a different angle, 4 more fixed the same day; open findings survive dual adversarial refutation, defaulting to refuted. `~/Projects/commander-in-chief/FINDINGS.md`
- Worked example: the fleet approved the outage. aigate, 17 August 2026. CSP `default-src 'self'` passed code review, adversarial verification, judge approval and 142 green tests, then killed the

dashboard on first load; the suite exercised HTTP without a browser. `~/Projects/aigate/blog/2026-08-17-the-fleet-approved-the-outage.md`

- Worked example: green build, dead UI. weddingbuget, 22 August 2026. 6 of 20 landing pages visually broken; 20 forms permanently disabled behind a 20/20 static audit; users stuck on the signup form about 3 hours after HTTP 201 registration. `~/Projects/weddingbuget/blog/2026-08-22-green-build-dead-ui.md`
- Worked example: the jury was framed. aigate, 29 August 2026. 3 rounds, 15 verdicts, zero sign-offs at 25/25/26 concerns; after a one-line juror prompt change, 4 of 5 signed off at 8–9/10; surviving concern was `#5fbb1f` small text at 4.36:1 against the 4.5:1 WCAG AA bar, fixed to 5.34:1 `~/Projects/aigate/blog/2026-08-29-the-jury-was-framed.md`

Cut for lack of a primary source: alert fatigue a primary barrier for 63% (CNCF 2024), 67% of alerts ignored daily (incident.io 2025), incidents up 43% at nearly $800,000 each (PagerDuty). None traced to a primary publication; all three removed rather than hedged.

“You fixed the loud one and shipped the quiet one.”

CHAPTER 37

You Fixed the Loud One

"POSITIVE, *adj.* Mistaken at the top of one's voice."

AMBROSE BIERCE, THE DEVIL'S DICTIONARY (1911)

One night in August a database optimization of mine sat one approval away from overwriting **337,000 rows** with thinner data. It threw nothing. The config parsed, the tests passed, and I was busy being pleased about a different fix that had made a graph go up. What stopped it was a reviewer running a `COUNT` query before clicking approve. Not me, and not clever.

> ***Bottom line:*** *You fixed the loud one and shipped the quiet one. The bug that generates complaints gets a ticket, a branch, a fix, and a green checkmark. The bug three feet away, the one with no error text and no angry user, ships in the same commit. Loud failures recruit humans. Quiet ones don't, so they survive every review, including the careful ones. What has caught them in my systems is always an independent check, a machine or a second person: a COUNT query before approval, a curl that logs in for real, a timestamp read the next morning. Never the author. The author already believes it works, which is why he shipped it.*

• • •

WHEN IT BITES

- The backlog item says a bug "corrupts every successful observation." You write the guard. The guard is real. Most of the corruption was a grouping artifact; the actual defect was a DOM race nobody had filed.
- A cache deploy makes the site dramatically faster and everyone cheers. Logged-in users are being served the anonymous cache, and the page still renders, so nobody screams.
- The scheduler stops, not with an exception or an alert. The cron line dies at the shell redirect before the runtime starts. A morning of nothing looks exactly like a morning of everything working.
- You fix the crash and ship the wrong timestamp axis in the same session, because the crash had a stack trace and `created_at` had a plausible name.
- A correction makes your headline number smaller, and your team treats that as a loss.

• • •

THE PATTERN

Loud failures come with a recruiting mechanism attached: a stack trace, a 500 page, a user who cannot log in.

Quiet failures have no recruiter. A cache serving the wrong body still serves a body. Success is what gets read at review, and the system reports success while losing something.

So the two live together. You go looking for the loud one, fix it, and the quiet one is in the same file, sometimes the same function. Ch. 33 covers the asymmetry in how failures announce them-

selves. This chapter is what that asymmetry does to a workday. It sets your ordering, and your ordering is wrong.

The counter is small and specific. Before the next deploy, name the three checks something other than you will run: a `COUNT` against production for anything that writes, a `curl` against a real logged-in session for anything that touches caching or auth, and a timestamp on the newest log line for anything scheduled. Those three, run by a second person or a cron job, are what caught every save in this chapter. Nothing about them is sophisticated. That is the point.

• • •

ONE WORKED EXAMPLE

airank, 8 August 2026. Ten-hour session, six fixes landed. Two of them are the whole chapter.

The loud one first. The backlog said duplicates "corrupt every successful observation." Before writing the guard I ran the grouping the metric rested on, then ran a second grouping over the same 169 rows. Grouped by `product_name` alone, **58 of 169** rows were duplicates. Grouped by brand plus name plus `use_case_segment`, **4 of 169** were. Same data, two definitions of "the same observation." So **54 of the 58** were grouping false positives, not a fake bug: four real duplicates hiding behind a denominator nobody had questioned, mine included.

Now the quiet one. The composer selector was a defensive union of five fallbacks, written that way so a UI change at OpenAI would not kill collection. It matched the real composer, and it also matched a hidden `<textarea style="display:none">` sitting **earlier in DOM order**. Earlier wins. The collector typed into an invisible element and reported it had done its job: across the five phrases tested, five failures, zero errors. The fix was `.filter({visible:true}).first()`, applied to the typing target and the wait. Result on the same five phrases: **5 for 5**.

Nobody filed that one. It surfaced only because the loud bug turned out mostly fake and there was time left in the day.

The session punished me once more. Recovery ran against a 112 MB NDJSON buffer using `File::get()` plus `explode()`, three copies in memory, and it died: `Allowed memory size of 134217728 bytes exhausted (tried to allocate 111115272 bytes)`. Loud, so it got fixed fast, hiding two things next to it: `created_at` was the wrong timestamp axis, and `--limit=50` could report the queue idle while work remained. Neither throws. I wrote the recovery path. Then I picked `created_at` because it had a friendly name, which is roughly the engineering rigor of a GeoCities under-construction GIF: it looks like something is being handled.

airank, 10 August 2026. A finding about tracking parameters in ChatGPT citation URLs moved **6.04% to 1.38% to 19.10%** in one day, and every move was me correcting my own methodology, starting with confusing `msclkid` (Microsoft Advertising's click id) with `msockid` (an undocumented parameter Bing appends to search-result clicks, organic ones included). The final number was smaller than the one I started with and held up under every check I could then run against my own data; Ch. 47 turns that sequence into a rule about what you are allowed to publish.

airank, 13 August 2026. Four decisions, one night, three nearly shipped broken.

A database optimization would have overwritten **337,000 rows** of richer archive pointers with thinner ones. Caught by a reviewer who ran a `COUNT` against production before approving.

The nginx microcache would have broken every login, twice over. The bypass map used `$cookie_airanks_session`, and nginx variable syntax cannot express a hyphenated cookie name, so logged-in users were served the anonymous cache. The second break was `Set-Cookie` stripping. Caught by curl test failures, not config review, because the config looked correct in exactly the way that matters least.

And the scheduler had been dead for six hours. The cron entry's `>> /var/log/file` redirect is opened by cron's shell as `www-data`, which cannot create files in root-owned `/var/log`. The line died at the redirect, before PHP ever started, silently, at the exact moment logrotate did its job and rotated the file away. Caught the next morning by a timestamp check on the newest log line.

Three saves, none from the person who wrote the change.

aigate, 17 August 2026. Not airank, so you can see the shape repeat. Hardening the Content Security Policy with `default-src 'self'` visibly broke the Vue dashboard into unstyled template soup, which took a minute to fix. Underneath, the same policy blocked the `unsafe-eval` that Vue needs to compile in-DOM templates with `new Function()`, so the page came back beautiful and ren-

dered nothing while the suite passed and curl saw green headers. A Playwright screenshot with console output caught it, because a browser was the only thing that ever looked at the page.

• • •

THE QUIET FAILURE

The loud failure of this chapter is the obvious one: you fix the reported bug, the real one ships beside it, the deploy is green.

The quiet failure sits underneath it. **Your triage order is set by volume, and volume is an unreliable severity proxy.** The duplicate report was the loudest item in the backlog and most of it was an artifact of how I had grouped the rows. How often that happens across a whole backlog, I cannot tell you. My recollection is that it was not the only item that shrank when I re-ran the number under it, but I never counted, so do not let me hand you a rate. The composer race had no report at all and was destroying every session it touched. A backlog sorted by how upset the reporter was is a 1998 hit counter: a number that goes up and measures nothing you can spend.

Second: **you count the fix, not the check.** The composer fix took ten minutes. The grouping query that cut the duplicate bug from 58 to 4 took ninety seconds and saved ten hours. The write-up of that night logs the same arithmetic one line lower: **10 sessions** spent re-measuring what a single `git show --stat` answered for free.

Third: **you optimize for the number that gets applause.** A wrong 6.04% costs you the entire finding the moment somebody reads the parameter docs, which is the whole argument of Ch. 47.

• • •

DO / DON'T

Do

- Re-run the measurement a bug report rests on before writing the fix. 58 of 169 and 4 of 169 were the same afternoon.
- Write the check that catches the failure silently, not the alert that fires after someone complains. A timestamp on the newest log line beats a dashboard nobody opens.
- Test what the change was supposed to preserve, not just what it was supposed to improve. The cache got faster and broke logins; only a curl against a logged-in session sees both.
- Make somebody other than the author run the verification. All three of the 13 August saves came from a second person or a machine.
- Publish the correction that makes your number smaller. 19.10% survived scrutiny that 6.04% would not have.
- Assume a green deploy with a fixed loud bug contains a quiet one, and go looking while the file is still open.

Don't

- Don't treat a vividly worded backlog entry as evidence of severity. Vividness is a property of the writer.
- Don't accept a defensive union of fallback selectors without ordering them. Five fallbacks and DOM order gave a hidden textarea priority over the real composer for five straight sessions.
- Don't read a whole file into memory to recover from a failure. `File::get()` plus `explode()` on 112 MB is three copies and a fatal at the exact moment recovery matters.
- Don't trust a config that parses. `$cookie_airanks_session` parses fine and means nothing.
- Don't infer a scheduler's health from `crontab -l`. The line existed. It died at the redirect.
- Don't ship a correction quietly. The retraction makes the next number believable.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 33 is the asymmetry itself: why some failures recruit a human and some do not. This chapter is the operational consequence: the recruiting mechanism chooses your work order, and chooses it badly. Ch. 38 takes the next step, the failure invisible because the system reports success in good faith. Ch. 47 turns the 10 August retraction sequence into a publishing standard: what you are allowed to claim when you correct a number in public.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (fix the loud one, ship the quiet one; an independent check catches the quiet one, never the author). Argument, not citation.

Verified:

- Duplicate bug mostly a grouping artifact: 58 of 169 by `product_name` alone vs 4 of 169 by brand + name + `use_case_segment`, so 54 of the 58 were grouping false positives; the shipped guard keys on brand+name+segment. Also "Sessions spent re-measuring what `git show --stat` answered free: 10." Internal artifacts, not public: airank engineering posts dated 8 August 2026, `2026-08-08-fifty-eight-duplicates-four-of-them-real` and `2026-08-08-six-fixes-and-a-ceiling`, in the private airank repo.
- Composer DOM-order race: five-fallback union matched a hidden `<textarea style="display:none">` earlier in DOM order; fix `.filter({visible:true}).first()`; 5 for 5. Same post.
- Recovery memory exhaustion on 112 MB NDJSON: `File::get()` + `explode()`, `Allowed memory size of 134217728 bytes exhausted (tried to allocate 111115272 bytes)`. Same post.
- ChatGPT URL tracking-parameter rate 19.10%; headline sequence 6.04% to 1.38% to 19.10%; `msclkid` (Microsoft Advertising) vs `msockid` (undocumented Bing parameter appended to organic search-result clicks). Internal artifact: airank post dated 10 August 2026, `2026-08-10-every-correction-cost-us-a-bigger-number`, private repo.
- Nginx microcache broke logins: `$cookie_airanks_session`, nginx variable syntax cannot express a hyphenated cookie name, logged-in users served the anonymous cache; plus `Set-Cookie` stripping. Internal artifact: airank post dated 13 August 2026, `2026-08-13-three-saves-in-one-night`, private repo.
- Scheduler dead six hours: `>> /var/log/file` redirect opened by cron's shell as `www-data`, which cannot create files in root-owned `/var/log`; the line died at the redirect before PHP started, silently, as a log-management tool rotated the file. Same post.
- Database offload would have overwritten 337,000 rows of richer archive pointers; caught by a reviewer running `COUNT` on production. Cache improved throughput from 16.6 to 1,873 requests per second. Same post.
- aigate CSP hardening, 17 August 2026: `default-src 'self'` broke the Vue dashboard visibly, and also blocked the `unsafe-eval` Vue needs to compile in-DOM templates with `new Function()`; caught by a Playwright screenshot plus console output, not config review. Internal artifact: aigate post dated 17 August 2026, `2026-08-17-the-fleet-approved-the-outage`, private repo.

What I could not verify:

- No verified public case matching the DOM-order composer race (visible selector losing to a hidden element earlier in the tree).
- No published study on ChatGPT URL parameter tracking prior to airank's August 2026 work. OpenAI's public statement on ad placement predates the measurement.
- No third-party verification of the 19.10% rate. Internally consistent, methodology sound, no independent corroboration.
- No public incident documentation matching the logrotate plus cron-redirect plus `www-data` failure class.

CHAPTER 38

Your Probe Is Lying

"Things Are What They Are Reported To Be."

JOHN GALL, SYSTEMANTICS: HOW SYSTEMS WORK AND ESPECIALLY HOW THEY FAIL (1975)

The counter said $13.69 an hour. I looked at it, nodded, and built a budget on it. It was August, the workers were fine, nothing was red, and it was exactly the kind of number you want to see: not suspiciously low, not alarming, just fine. Somewhere in that same week a job queue emptied 68 jobs in about 40 milliseconds and reported itself healthy for doing it. Four dashboards, four numbers, every one of them green, every one of them signed off on by me. It took one afternoon and a handful of read-only SQL queries to find out what each of them was actually measuring.

> ***Bottom line:*** *The measurement is broken and the number looks fine. A green dashboard is not evidence that the system works; it is evidence that one specific check returned true. Most of the expensive incidents I have sat through were not outages. They were long stretches of confident, well-formatted, wrong numbers, and nobody looked under them because looking under a healthy number feels like paranoia. It isn't. It's the job.*

• • •

WHEN IT BITES

- Your spend counter says $13.69/hr. You are actually burning $129.35/hr. The counter is not broken in a way that throws an exception, so nothing pages.
- The collection pipeline reports 27,262 records collected. Green. Scoring reads a different shape and consumes none of them. Both halves are healthy. The gap between them is where the product lives.
- The service is up, every check passes, and it is silently reverting customer commits. The first person to notice is a customer, because merge correctness is not availability.
- A watcher polls an external index every fifteen minutes to confirm a pipeline works. The pipeline does not use that index. The watcher is telling the truth about a system nobody is running.
- Someone asks for a chart. The chart gets built. Every bar is full because of how the data was selected, and will be full forever no matter what the business does.

• • •

THE PATTERN

There is a specific shape to this and it repeats across every layer, from a Kubernetes liveness probe to a spend dashboard to a status field in your own database.

Something already told you the answer, so you stopped measuring.

That's the root cause. Not laziness, not bad engineers. A comment said the feature was implemented. A counter said the spend. A status field said the job succeeded. Each of those is a report *about* the system, and each got treated as the system.

The distributed-systems version of this is not new. Cindy Sridharan published "Health Checks and Graceful Degradation in Distributed Systems" on Medium in August 2018, with a section titled "Health is a Spectrum, not a Binary Taxonomy": a binary up/down check misses partial failure, so a process can pass its check while being unable to complete work inside SLA. That is eight years old. I read it, quoted it at people, felt smart, and then went and built a spend counter that failed in exactly the way she described. Rack2Cloud documented Kubernetes checks reporting success while silently missing the objective in May 2026, the gap running for weeks before a human trips over it during an unrelated incident.

The failure modes stack in a predictable order:

Wrong surface. The probe measures something real, correctly, that has nothing to do with the path in production.

Wrong dimension. The probe measures availability when the failure is correctness. GitHub's April 23, 2026 merge queue regression is the textbook case: an incomplete feature flag let a new squash-merge code path activate in production, and merge groups with two or more PRs silently reverted changes from previously merged PRs. Existing monitoring caught nothing, because the service was up. 2,092 pull requests across 658 repositories were affected. Found by customer reports.

Wrong arithmetic. The probe is on the right surface and the right dimension and the math is wrong under concurrency. A read-modify-write spend counter with 50 workers hammering it does not report a slightly low number. It reported 9.4x low.

The probe is the outage. AWS, October 2025. The DynamoDB DNS management automation created an empty DNS record for US-East-1 and could not auto-repair it, because the repair logic was the thing that broke. Manual operator intervention required. Signal, Snapchat, Roblox, Duolingo, Ring, north of 2,000 companies, 8.1 million user reports.

The probe passes because the validator was skipped. CrowdStrike, July 2024. A faulty sensor configuration update shipped despite a bug in the Content Validator, an out-of-bounds memory read blue-screened 8.5 million Windows devices, and a similar bug had occurred one month prior. The QA gate reported pass. The gate was the defect.

Then there's the aggregation lie, the most socially protected of the bunch, because it comes with decimal places. LeadDev counted 257 separate GitHub incidents between May 2025 and April 2026, 48 of them major, 37 in February 2026 alone, against measured 90-day uptime of 84.88% versus an official 99.79%. Same system, same window, fifteen points apart. Neither party is lying.

Sit with the ugly part: a full year of data did not pull those two numbers together. More samples don't fix a definition problem. They hand both sides a bigger pile of evidence for the answer they already had, and everybody walks out of the room more confident and no more correct. The 84.88% is the one that matches what a developer felt in February, and it's the one nobody published. I run the small version of this on myself constantly. Week three of a plausible number feels like corroboration. It is just week three.

The fix is not more monitoring. More monitoring is more surfaces to be wrong about. The fix is one rule:

Check the effect, not the report.

Byte counts, not the log line that says written. The rows in the table, not the status field on the job. Every one of those pairs looks equivalent on a whiteboard and is not equivalent in production.

• • •

ONE WORKED EXAMPLE

airank, August 9, 2026. Four numbers, all reporting healthy, all lying, all caught by single queries against real data in an afternoon.

One. The spend counter showed $13.69/hr. Actual spend: $129.35/hr. 9.4x under-reported, caused by a read-modify-write race with 50 workers stomping each other's increments. An absurd number gets investigated. A plausible wrong number gets budgeted around. I built that counter, and checked the tile every morning for weeks like it owed me money.

Two. The code claimed, in a comment, to save raw API answers. It silently discarded them, and because the comment said the feature was implemented, nobody ran the query for weeks. When somebody finally did: 18,567 API failures with zero text stored. Comments are the worst probes ever invented, because they never go red and cost nothing to write. I wrote that comment.

Outcome: all four got fixed the same week, and none of them announced themselves. Same batch of incidents: a job retry timestamp read as UNIX epoch, expiring all 68 jobs in about 40ms. The queue reported healthy throughout, because a queue that empties in 40ms is technically an empty queue.

• • •

THE QUIET FAILURE

The loud failure is the pager. Something went red, someone woke up, the postmortem got written. That's the system working.

The quiet failure:

The number was wrong for weeks and you made decisions on it.

That cost never shows up in the incident tracker. You didn't have an outage. You had a budget built on $13.69/hr and a roadmap built on 27,262 records that nothing consumed.

Second quiet failure: **the probe encodes what someone believed the system was on the day they wrote it.** Then the system moves and the probe doesn't, and it keeps passing, because it is still correctly measuring a system that no longer exists.

Third: **you measure the visualization instead of the data.** On August 10, 2026 a percentile bar chart got measured before a line of code was written: a `LIMIT 10` result set where every row is roughly 99th percentile by construction, so all 40 bars would render identically full forever, regardless of what the business does. A minute of measurement caught it. It never got built.

• • •

DO / DON'T

Do

- Make every probe traverse the real path. One live call through production routing beats any amount of correct measurement of an adjacent surface.
- Check the effect, not the report. Count the rows. Count the bytes. Query the table. The status field is an opinion.
- Assume plausible numbers are the dangerous ones. $13.69/hr got nodded at for weeks; $129.35/hr would have been questioned in an hour.
- Ask what dimension you're monitoring. Availability monitoring on a correctness bug produces silence, which reads exactly like success. Ask GitHub about April 23.
- Probe the seams, not just the services. The 27,262-record gap sat between two components that were both, correctly, green.

Don't

- Don't trust a comment. A comment is a claim about code written by someone who has since been wrong. 18,567 failures with zero stored text sat behind a comment saying the feature worked.
- Don't build a counter with read-modify-write and 50 concurrent writers, then treat it as an instrument. It is a random number generator with a plausible mean.
- Don't let the automation that protects the system be the only thing checking it. AWS disabled the DynamoDB DNS planner worldwide in October 2025 for exactly this reason.
- Don't ship a gate that reports pass without proving it can report fail. CrowdStrike's Content Validator passed a payload that blue-screened 8.5 million machines.
- Don't gate a new code path on a flag you haven't verified is actually gating. Incomplete flags activate in production and monitoring does not notice.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 20 is the definitional version: if "done" isn't measurable, the most articulate artifact in the room wins. This chapter is the operational version: even when you *do* measure, the instrument can be pointed at the wrong sky and lie with a straight face for a month. Ch. 44 takes the failure that is silent by design, the code path returning success while discarding your work. Ch. 48 is what to do when you cannot trust the instrument at all and have to reconstruct state from effects. Ch. 15 is the upstream cause: a plan is a bet priced in confidence, and a broken probe is what lets a bad bet ride.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("check the effect, not the report"): argument, not citation.
Verified:

- AWS October 2025 outage: latent defect in DynamoDB DNS management, empty DNS record for US-East-1, automation could not auto-repair, manual operator intervention required. The Guardian, October 2025: https://www.theguardian.com/technology/2025/oct/24/amazon-reveals-cause-of-aws-outage
- GitHub merge queue regression (April 23, 2026): squash strategy on merge groups silently reverted commits; 2,092 PRs across 658 repositories; monitoring missed it because the issue was correctness, not availability. Also the reliability record: 257 incidents May 2025 to April 2026, 48 major, 37 in February 2026; measured 90-day uptime 84.88% versus official 99.79%. Same system, same window, and the two figures are methodological opposites (measured versus reported), not one number decaying into the other. LeadDev, May 2026: https://leaddev.com/software-quality/whats-gone-wrong-at-github
- CrowdStrike July 2024: faulty sensor configuration update shipped despite a Content Validator bug; out-of-bounds memory read blue-screened 8.5 million Windows devices; similar bug one month prior. ZDNet, July 2024: https://www.zdnet.com/article/what-caused-the-great-crowdstrike-windows-meltdown-of-2024-history-has-the-answer/
- Binary up/down health checks miss partial failure; a process can pass its check while unable to complete work within SLA. Cindy Sridharan, "Health Checks and Graceful Degradation in Distributed Systems," Medium, August 2018: https://copyconstruct.medium.com/health-checks-in-distributed-systems-aa8a0e8c1672 Fetched live 9 September 2026; the section heading "Health is a Spectrum, not a Binary Taxonomy" is confirmed present on the page. Indexes disagree on the exact day (11 vs 12 August 2018), so the chapter says August 2018.
- Kubernetes health checks reporting success while silently missing the objective, for weeks. Rack2Cloud, May 2026: https://www.rack2cloud.com/kubernetes-health-checks-actually-guarantee/
- AI integration metrics looking fine while the model served stale results; caught only against a parallel canary. Medium, August 2026: https://moyano.cl/why-your-ai-integration-is-failing-and-how-debugging-changes-everything/
- Worked example, four lying numbers: airank, August 9, 2026. `~/Projects/airank/blog/2026-08-09-four-numbers-that-were-lying.md`
- Worked example, wrong-surface probe: airank, August 11, 2026. `~/Projects/airank/blog/2026-08-11-the-probe-and-the-name.md`
- Worked example, eight silent failures (reappearing modal, 41% yield, session_mismatch, UNIX-epoch retry expiring 68 jobs in ~40ms, unregistered queue supervisor, Redis retry duplication, 1440-minute scheduler mutex releasing 382 jobs, missing local-network access): airank, August 9, 2026. `~/Projects/airank/blog/2026-08-09-the-night-we-changed-instruments.md`
- Worked example, measuring the chart first: airank, August 10, 2026. `~/Projects/airank/blog/2026-08-10-the-chart-that-would-have-said-nothing.md`

What I could not verify:

- Six adjacent failure modes were searched and left out of the chapter for lack of a named, dated public source: a feature flag reporting enabled while the gated path never ran, a cache serving stale data behind a health check pointed at the cache instead of the origin, automated remediation reporting success against the wrong target, a circuit breaker reading closed while discarding queued requests, an observability vendor's postmortem on its own blind spot, and regional averaging documented as the reason an outage went unseen. Every one of them is plausible and none of them is cited here.
- No verified public source ties the airank incidents to the industry incidents; the connection is Jeremy's pattern claim, stated as such.

“One tiny service overload cascades to its neighbors, which overload theirs, until the whole stack is on the floor gasping.”

CHAPTER 39

Cascading Failures Look Like Chaos

"I have a network file system, and I have broken the network, and I have broken the file system, and my machines crash when I make eye contact with them. I HAVE NO TOOLS BECAUSE I'VE DESTROYED MY TOOLS WITH MY TOOLS."

JAMES MICKENS, THE NIGHT WATCH, USENIX ;LOGIN: (2013)

At T+3:10 the dashboard went red in four places and three teams got paged: database, API, logging. Each looked at their own graph, and each was right. The database was answering every query in 50ms. The API vendor was answering every request it received. Logging was delivering every log handed to it. Three healthy systems, one dead platform, and 100 of 100 connections held by something nobody in the room owned. The thing that started it happened forty seconds earlier, and it was still happening while they argued whose fault it was.

> ***Bottom line:*** *One tiny service overload cascades to its neighbors, which overload theirs, until the whole stack is on the floor gasping. The synchronized collapse looks random because you're looking at the wrong layer. The root was small. The propagation was faster than your pager.*

• • •

WHEN IT BITES

- You deploy an agent. One downstream API starts rate-limiting. The agent retries immediately, retries pile up, downstream gets hammered harder, and the whole system locks in seconds.
- A database cursor times out on one batch. The transaction rolls back, the loop re-tries it, the pool exhausts, everything downstream stalls.
- One worker hits a memory leak. It bloats, stops taking tasks, stays alive. Pool capacity shrinks by 1/N, and at 50% exhaustion latency goes quadratic.

• • •

THE PATTERN

Cascades follow a chain that's easy to read after the fact and impossible live:

1. A single service degrades, not fails, *degrades*. Slower, fewer slots, higher error rate.

2. Neighbors see timeouts. They retry immediately, because retrying feels responsible, adding load to the sick service.

3. It gets slower. The backoff isn't exponential, it's synchronous: all neighbors retry on the same tick. Fifty clients firing together is a Zerg rush (1998): every unit harmless, arriving together they end the game.

4. Neighbors exhaust their queues. They can't flush retries fast enough, and new requests pile up behind them.

5. Neighbors degrade. Now they're the problem. Their callers see errors, retry, degrade. The wave moves outward.

6. Load flows backward. A calls B calls C, C degrades, the queue backs up through B to A. By the time A's metrics spike, C was gasping five minutes ago.

7. Synchronized collapse. Every layer hits its limit at once, and causality is invisible because every error looks identical: timeout, connection refused, 429, 500.

The sync is the trap. You see ten services failing together and assume ten separate bugs. Wrong. One degraded, and the other nine are downstream of it. Think of a dial-up handshake: both modems screeching on a shared clock, both ends equally broken. Same as your dashboard.

• • •

ONE WORKED EXAMPLE

Slack published theirs, and it's the cleanest public copy of this chain I know of. On February 22, 2022 they were rolling a Consul agent upgrade in 25% batches. The third landed at peak traffic, the restarts invalidated the cache tier, and every request that missed cache became a scatter query against the database. The database saturated, queries timed out, clients retried, and the retries kept the database loaded enough that the cache could never refill, which guaranteed more scatter queries. Their writeup doesn't name a pinned connection pool, so read that mechanism as mine, not theirs. That is the whole chapter: the symptom became its own cause and didn't stop on its own. Engineers throttled client boot traffic hard and walked it back up in increments (Slack Engineering, February 2022).

Here's my own dated one, small and stupid. On 11 July 2026 I shipped a fix to `clients/aigate-run.sh` (commit 070dd20) for a cascade I built myself. Anthropic returns 529 when the whole API is overloaded, global load shedding, nothing to do with your account. My client treated it like a per-account rate limit and hopped to the next account in the pool. So on a 529 every account got tried, every one returned the same 529, and the loop burned all three attempts in seconds and parked each account for two minutes. Pool exhausted. Not because I was over any limit, but because I asked everybody at the same instant whether they were free. The fix was one behavior change: on a 529, stay on the same account and wait 10 seconds. Ten seconds of patience beat three accounts of enthusiasm.

Here's that chain drawn out on a bigger stack. This timeline is illustrative, a composite of my own aigate nights rather than one log I can hand you:

aigate running N concurrent eval tasks, each calling Anthropic's API, a database, and a logging service. At T+0:00 it is boring: Anthropic at ~200ms, database pool at 100 connections with ~95 in use, logging keeping up on 2 workers.

T+2:30, the hiccup. Anthropic latency goes from 200ms to 800ms. Nothing errors, so nobody notices. Each request now holds a database connection for 800ms instead of 200ms, and pool use goes from 95 to 99. One request waits 10 seconds, times out, retries immediately. Fifty more do the same, all at once. The pool is now permanently full.

T+3:00, three innocent systems. 500 timed-out requests peg both log workers at 100% CPU, though logging is fire-and-forget and never saw the cascade. The database has free CPU, free disk, and still answers in 50ms. Nothing is broken anywhere except "connection pool full."

T+3:10, the collapse and the wrong fix. Red on Anthropic latency (1500ms), aigate latency (5000ms+), pool (100 of 100), log queue (1000+ waiting). Three teams page, each staring at its own healthy graph. Then somebody says the sentence that costs you the next hour: "connection count is maxed, but the database is fine." Team 1 doubles the pool to 200. It works for 90 seconds. By T+3:30 the 200 are gone too, because Anthropic is still slow.

T+4:00, the actual fix. Someone reads the timeline instead of the dashboard and sees Anthropic moved first. Cut that timeout from 30 seconds to 5, add a circuit breaker (50 timeouts in a row, stop for 60 seconds, let the pool drain). Normal in two minutes. The bill for not reading the timeline is memory, not logs: I remember two hours, three pagers, a few hundred dead requests, and a four-figure credit I ate without arguing. Reading the timeline first costs 10 minutes.

• • •

THE QUIET FAILURE

The loud failure is obvious: everything is red, you page through the stack, you find the root. Takes hours but it's *found*. The quiet ones are cheaper and worse:

Failure A: You see the cascade and treat the symptom.

Symptoms are the last layer. When the pool is maxed, the database didn't break, the upstream did. Raising the pool *seems to work* while the root cause keeps burning money.

I have raised a pool from 100 to 200 at two in the morning and bought myself exactly 90 seconds. Best 90 seconds of my night. Then it maxed at 200 and I did it again: 200 to 500 to 2000, until 2000 connections is itself the problem. Adding lanes to a highway where nobody can take an exit. Still jammed, now wider.

Failure B: The cascade is so fast you miss it.

Root cause at T+2:30, visible collapse at T+3:10. Forty seconds. Your monitoring polls every 60, so you see the collapse and never the cascade forming, and by the time you pull logs the root event is buried under 500 requests and 500 retries.

Failure C: You add more retry logic to "fix" cascades.

Someone reads about cascades and decides the fix is "retry harder." Backoff and jitter are good, but if the root cause persists, retries only defer the wall. A request that times out once times out 10 times if nothing changed, at 10x the tokens and connections held. Slack was doing it right in February 2022, exponential backoff with jitter, and their own writeup still says the automated retries kept contributing load until a human turned the traffic off.

My own version: MariaDB returns "Host is blocked because of many connection errors" (SQLSTATE 1129), the database politely telling you that you are the problem, and my reflex for years was to reconnect harder. Every retry increments the counter that caused the block, the single largest bucket in my own error audit, roughly 270 logged failures, every one of them me, digging. The fix is one `mysqladmin flush-hosts`. Twenty-five years of shipping software and I got pwned by a counter printing its own name in the error string. Total n00b move.

The fix is a circuit breaker: after N consecutive failures, stop trying for M seconds, let the upstream recover, resume when there's reason to believe it's back. Netflix shipped this as Hystrix in November 2012, with thread pool isolation so one sick dependency can't eat every caller's threads and a breaker that trips at a failure threshold and moves through open, half-open, closed. Martin Fowler wrote the pattern up in March 2014. It isn't clever. It's grabbing your friend by the shoulders and sitting him down for 60 seconds before he runs at the wall again.

Failure D: Cascades at runtime vs. cascades in your head.

Most teams document cascades wrong: a runbook that says "if the database is slow, check the connection pool." If the cascade runs database to queue to worker, the wave hits the workers before the runbook triggers. The quiet failure is having the runbook and never testing it. Make one dependency slow on purpose and watch your own cascade form. Better on a Tuesday than at T+3:10 with three pagers going.

That is the whole idea behind Netflix's Simian Army, announced July 2011, and Chaos Monkey inside it: kill instances and degrade dependencies deliberately, during business hours, with engineers sitting there watching.

• • •

DO / DON'T

Do

- Measure the cascade, not the symptoms. Is the database slow because queries are slow, or because connections are held long? Trace the chain.
- Add timeouts at every layer. If A calls B calls C, timeout A->B *and* B->C.
- Implement circuit breakers, not just retries.
- Test your cascade behavior. Slow a service down and watch: graceful (fail fast, release resources) or catastrophic?
- Deploy bulkheads. A logging outage shouldn't cascade to eval: fire-and-forget queue, not a synchronous call.
- Own your backoff. Don't use the library default. Marc Brooker's AWS Architecture Blog post from March 2015 has the arithmetic: exponential backoff with no jitter makes total client work grow like N^2, while Full Jitter, `sleep = rand(0, min(cap, base * 2^attempt))`, cut client work by

more than half. Decorrelated Jitter, `sleep = rand(0, min(cap, prev_sleep * 3))`, is the other one worth stealing. My own caps are rules of thumb: 30 to 60 seconds for Anthropic rate limits, 100ms for a database lock.

- Log the cascade as it forms. Every timeout logs the upstream service, the timeout value, and the result, so you can grep "Anthropic timeout" and find the spike.

Don't

- Don't treat symptoms with more resources. If the pool is exhausted, the fix isn't "add connections," it's "why are connections held so long?"
- Don't assume cascades start where the alert fired. Investigate backward from symptom to root.
- Don't retry immediately. Synchronous retries in a cascade layer are how you build a retry storm. Let the loop (Ch. 14) decide when to re-try.
- Don't increase timeouts to "fix" cascades. A timeout holds a connection, a thread, and whatever memory that request holds for the whole wait, so a longer timeout doubles how much of your stack the cascade owns. Short timeouts are what let a caller fail fast and give the resources back. Drowning your sorrows in a bigger bucket is still drowning, you're just holding more of it.

• • •

WHERE THIS SITS IN THE BOOK

Part V (Ch. 31 to 40) is reliability, and this chapter scales the inner loop of Part II (Ch. 14 to 20) without setting the building on fire. Ch. 35 is the retry half of every cascade here. Ch. 37 is what comes after: the alert that screamed is rarely what started it. Ch. 38 is why your dashboard swore the database was healthy while the database was the wall. Ch. 31 is the escalation flow, and escalation shape decides cascade shape. Ch. 32 tells you whether your degradation story is real. Ch. 44 and Ch. 47 are how you get causality out of the wreckage instead of five hundred identical timeouts.

All of it is useless if the cascade is invisible. See it first, then defend.

• • •

SOURCES AND RECEIPTS

Thesis: cascades are synchronous, propagate backward from symptom to root, and move fast enough that naive retry logic makes them worse.

Verified: - Cascade formation order, the synchronization illusion, the symptom becoming its own cause, and retry storms surviving correct backoff: all four in one public incident (Consul upgrade, cache invalidation, scatter queries, saturated database, retries with jitter still adding load until humans throttled client boot traffic). Slack Engineering, February 2022, https://slack.engineering/slacks-incident-on-2-22-22/ The writeup doesn't describe pool exhaustion; that half of the chapter is my own stacks. - Circuit breaker as the standard defense, with thread pool isolation and open/half-open/closed states. Netflix TechBlog, November 2012, http://techblog.netflix.com/2012/11/hystrix.html, and Martin Fowler, March 2014, https://martinfowler.com/bliki/CircuitBreaker.html - Backoff and jitter arithmetic, including both formulas quoted. Marc Brooker, AWS Architecture Blog, March 2015, https://aws.amazon.com/blogs/architecture/exponential-backoff-and-jitter/ - Chaos engineering as the way to find cascades on purpose, in business hours. Netflix TechBlog, July 2011, http://techblog.netflix.com/2011/07/netflix-simian-army.html - MariaDB SQLSTATE 1129 retry loop, roughly 270 logged failures: Jeremy's own error audit. - The 529 account-hop cascade and the 10-second same-account wait replacing a 2-minute park: aigate, `clients/aigate-run.sh`, commit 070dd20, 11 July 2026, ~/Projects/aigate/clients/aigate-run.sh

Honest gaps: - The long aigate timeline is labeled a composite in the text and it is, not one log; the 11 July 2026 529 story is the dated one. - No formal cascade detection algorithm cited; bulkhead isolation is mentioned but not detailed.

Not cited but relevant: Nygard, Michael. *Release It!* (2007), on cascade patterns and circuit breakers.

CHAPTER 40

A Human Has to Exist

"Declarations of high confidence mainly tell you that an individual has constructed a coherent story in his mind, not necessarily that the story is true."

DANIEL KAHNEMAN, THINKING, FAST AND SLOW, CH. 20 "THE ILLUSION OF VALIDITY" (2011)

An operator types `air:block --remove`, watches the flag clear, and gets on with their day. The command worked. The screen said so. Somewhere behind it a seven-day cached NXDOMAIN verdict still has its own opinion, and on the next pass the system quietly puts the block back and never mentions it. Six frontier models had reviewed that design and all six said ship it. The override survived exactly as long as nobody looked: a shelf life of one job cycle.

Six frontier models sat down on 15 August 2026 to review a one-day-old CLI. They got a brief saying the system "polls every 20s, hard cap 180s, then returns whatever it has." Five accepted it. One, kimi-k3, opened the source: the 180-second cap only existed on the pending branch, and the 429 rate-limit branch had no elapsed-time check at all. A server in sustained overload could pin the CLI forever. I wrote a one-line summary of my own 24-hour-old code, got it wrong, then rented six frontier models to agree with me about it. Five did. The n00b in that room was me, and I was also paying for the room.

> ***Bottom line:*** *Human-in-the-loop is not a checkbox. It is a testable property: at the moment of approval, the human must be able to say no with information, and that no must stop something. Miss either half and you have a signature block, not HITL. A signature block is worse than nothing because it launders the decision. Everybody downstream now believes a human looked.*

• • •

WHEN IT BITES

- Your approval screen shows a 4,000-token diff and two buttons. The reviewer approves 40 of them in an hour. Do the arithmetic on how long each one got.
- A compliance auditor asks who approved the pipeline change. A name is on it. That person could not tell you what it did.
- The system approves by default. Nobody said yes, nobody said no, and the absence of a no shipped.

• • •

THE PATTERN

EU AI Act Article 14 requires that high-risk systems be designed so they "can be effectively overseen by natural persons during the period in which they are in use": the human has to understand the system's limitations, detect automation bias, and override or reverse the output. That is an engineering spec, not a disclaimer, and it enters into force 2 December 2027. An Effective Altruism

Forum analysis from 31 August 2026 put it bluntly: Article 14 "neither acknowledges nor seriously addresses" how you implement it.

A 2024 Harvard Business School study gave 228 evaluators AI recommendations with clear explanations of the reasoning. Reviewers were 19 percentage points more likely to align with the AI than the control group; add narrative rationales and deference climbed another 5 points. MIT Sloan Management Review said it out loud on 12 June 2025: understanding the reasoning can turn reviewers into people who "rubber stamp rather than acting as a critical check."

That is automation bias: the tendency to favor AI suggestions and overweight them. Complexity and workload "increase reliance by placing stress on cognitive capacity." Your agent produces complexity and workload as its primary output: a bias machine with a tired person stapled to the end of it.

Software got its own version on 26 December 2025, when Leena Malhotra published three months of replacing human code review with AI review. Nothing paged anyone, because nothing broke: six weeks in, the juniors had stopped learning, the code went "homogenous but soulless," and a change traveled from author to production with exactly one human understanding it. Mitchell, Ghosh, and Passi made it structural on 24 August 2026: as agents gain autonomy they push humans out of the loop, and oversight, the fix everybody names first, "is not a simple solution."

Three conditions make a review real, and every one is a build task, not a policy sentence:

The human must have independent information. Not the agent's summary of what it did, a number they can check that the agent did not produce. What broke the false consensus above was one model reading the source file instead of the brief.

Saying no must cost less than saying yes. If rejecting means writing a paragraph and yes means one keystroke, you have priced the outcome you will get. Make the reject path as cheap as approve: a two-click reason field, not an essay.

The no must be enforceable and durable. If the override can be silently reverted by cache, default, or the next run, it was never an override.

Two of those three are dials your tooling already ships. Claude Code has four permission modes: Default asks before any edit, shell command, or network call; `acceptEdits` waves edits through; Plan mode touches no source until you say go; Auto mode runs the lot behind a classifier, and since v2.1.228 is the starting default on Pro, Max, and Team. Shift+Tab cycles all four, a dial labeled *how much trouble do you want to get in today*.

Set the dial by the door, not by mood. Two-way doors, act then report: edit a file, create a branch, run the suite. One-way doors, ask every time: `git push --force` to a shared branch, any DDL against production, a `DELETE` without a `WHERE` you have counted first, anything landing in a real customer's inbox. The split is not size, it is recovery: a flag defaulted off is a two-way door, the same flag at 100% is a one-way door. Every prompt you spend on a two-way door is attention you will not have at 1am when a one-way door shows up.

• • •

ONE WORKED EXAMPLE

airank, 15 August 2026. Two councils, same week, same repo, two shapes of the same failure.

The sharper one: a gate to stop the system from processing domains that do not exist. The cost of not having it was already measured, 495 false domains at $0.03 a lookup, roughly $16 of confident summaries about nothing. The council of six converged fast: verify via DNS, bundle an offline IANA TLD list for speed. Every reviewer green.

Then the adversarial review stage ran, which this workflow refuses to skip, and found the design ate its own override in both directions.

Unblocking was a lie. The `air:block --remove` command cleared the flag but not the cached seven-day NXDOMAIN verdict, so the next lookup re-condemned the domain and re-blocked it. The human's no was decorative.

Worth admitting what airank does and does not have. It does have a pre-flight ask: three artisan commands count the work and refuse to start until a human answers. The sharpest is `AirTracerQuery.php` line 90, which prints the exact number of real, billed API calls before a dollar moves. All three also take `--yes`, which is how a scheduler runs them, so the gate exists exactly as

long as a person is the one typing. Then the durable half: one line inside `unblock()` now drops the DNS verdict so a person's no outlives the cache.

The bundled TLD file could blacklist the future too: when ICANN adds a TLD, the snapshot goes stale and every domain on it is permanently condemned by a file nobody remembers is there. I shipped a list of what the internet looked like on one Tuesday and treated it as permanent, roughly the shelf life of an AOL 5.0 CD: correct on the day it was pressed, a coaster by the time you needed it.

Six models agreed, and the agreement was real and worthless as evidence: coherent, not correct.

The first council, about the 180-second cap, shows the other half. After the brief got retracted on the record, the chair measured the actual distribution instead of arguing about it. Of 8,853 domain rows, zero resolved between 60 and 180 seconds, and 8,736 took longer than 180. A switch, not a curve.

The third variant is the one that will happen to you. The commander-in-chief project, a deterministic Godot sim, documents 51 measured design decisions in `DECISIONS.md`. Entry D1 is the tank bail mechanic. The HUD prints `BAIL OUT! %ds`, but on default difficulty no player ever sees it work, because vestless riders get one-shot on the same tick the shell lands. The file says: "16 of 22 occupied-tank ignitions per the pre-flight are artillery, so this is the common case." The recommendation was a 30 to 45 tick grace window. The recorded resolution was "do nothing."

Four of the top five decisions got no sign-off at all. I wrote D1, wrote "do nothing" under it, and shipped the lying HUD anyway. Fifty-one measured decisions, and the review process for the top four was me not typing anything. Silence approved it, which is the hardest version to catch in an audit, because the artifact looks complete.

• • •

THE QUIET FAILURE

The loud failure is the human who was not there. Easy to find, easy to fix.

Your approval rate is 98% and you read that as the agent being good. It is at least as likely to mean your reviewer stopped reading in week three. Those two states produce identical dashboards. If you are not measuring reject rate and time-on-review, you cannot tell them apart.

The human reviews the agent's summary of the agent's work. That killed round one of the CLI council. Reviewing a summary is proofreading, not oversight.

The override that does not survive the next run. If the human's no expires and the machine's yes does not, the human is not in the loop, they are in the transcript.

• • •

DO / DON'T

Do

- Give the reviewer one number the agent did not generate. Row counts, a `GROUP BY`, a file read directly.
- Instrument the reject path: rate, latency, whether rejections were later vindicated. A reject rate of zero is a broken control, not a good agent.
- Make the no durable. Write a test that issues an override, runs the pipeline twice, and asserts the override still holds. That test would have caught the `--remove` bug before shipping.
- Run an adversarial pass after consensus, not instead of it.
- Record who reviewed, what they saw, and how long. Article 14 wants the human able to detect their own automation bias.

Don't

- Don't treat a richer explanation as better oversight. It moves reviewers toward the AI, not away from it.
- Don't count "nobody objected" as approval.
- Don't let the brief's author be the only approver of work built from it.

- Don't route every action through the same prompt. Volume manufactures automation bias.
- Don't call it human-in-the-loop if you cannot produce a rejection from the last quarter.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 42 (Your Agent Just Hit Your NAS) is what the agent is allowed to touch. This chapter is who stops it and whether that stop is real. Ch. 41 is how a refusal gets recorded so the next run inherits it. Ch. 60 is the counterweight. Ch. 15 is the upstream version, where a well-formatted plan gets read as work already done.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("if they can rubber-stamp it, you don't have HITL"), argument, not citation. **Verified:**

- EU AI Act Article 14: high-risk systems must be designed so they "can be effectively overseen by natural persons during the period in which they are in use," including understanding limitations, detecting automation bias, and overriding output. In force 2 December 2027. https://artificialintelligenceact.eu/article/14/
- Harvard Business School (2024), 228 evaluators: explanations made reviewers 19 points more likely to align; narrative rationales added 5; automation bias definition. Via TianPan.co, 15 April 2026. https://tianpan.co/blog/2026/04/15/human-in-the-loop-rubber-stamp
- Automation bias and workload. NIH/PMC systematic review, 2012. https://pmc.ncbi.nlm.nih.gov/articles/PMC3240751/
- Explainability can produce rubber-stamping rather than critical checking. MIT Sloan Management Review, 12 June 2025. https://sloanreview.mit.edu/article/ai-explainability-how-to-avoid-rubber-stamping-recommendations/
- Mitchell, Ghosh, Passi, "AI Agents Push Humans Out of the Loop." arXiv, 24 August 2026. https://arxiv.org/abs/2608.23642
- Article 14 "neither acknowledges nor seriously addresses" implementation. Effective Altruism Forum, 31 August 2026. https://forum.effectivealtruism.org/posts/MHAaDyBDpSp2hKfSM/the-eu-s-ai-act-looks-to-implement-human-oversight-the-only
- Three months of AI code review replacing human review: cracks at six weeks, juniors stopped learning, code "homogenous but soulless," "everything appeared to work." Leena Malhotra, DEV Community, 26 December 2025. https://dev.to/leena_malhotra/i-tried-replacing-human-review-with-ai-heres-where-it-quietly-failed-4jh3
- Claude Code permission modes: default, `acceptEdits`, plan, auto; auto is the starting default on Pro, Max, and Team from v2.1.228. Claude Code documentation, retrieved 9 September 2026. https://code.claude.com/docs/en/permission-modes
- Two-way / one-way door lists, quoted to match Ch. 60 so both chapters use one definition. `~/Projects/aibook/manuscript/60-Dont_Ask_on_Reversible_Work.md`
- Worked example 1: air CLI council, airank, 15 August 2026. False 180s-cap brief, retracted on record; 8,853 rows, zero between 60 and 180 seconds, 8,736 over 180. `~/Projects/airank/blog/2026-08-15-air-cli-council.md`
- Worked example 2: domain guard council, airank, 15 August 2026. 495 false domains, ~$16 at $0.03 each; `air:block --remove` left the cached NXDOMAIN verdict; stale IANA TLD snapshot. `~/Projects/airank/blog/2026-08-15-domain-guard-council.md`
- `unblock()` (line 51) now also forgets the cached DNS verdict, so a manual unblock is not re-blocked on the next run; `block()` at line 39. Read 9 September 2026. `~/Projects/airank/app/Services/Api/DomainBlocklist.php`

- Worked example 3: commander-in-chief `DECISIONS.md`, 2026. 51 measured decisions; D1 tank bail; "16 of 22 occupied-tank ignitions per the pre-flight are artillery"; resolved "do nothing"; four of the top five got no sign-off. `~/Projects/commander-in-chief/DECISIONS.md`
- Pre-flight approval gates in airank, read 9 September 2026: `AirTracerQuery.php` line 90 (`confirm("Proceed with {$totalCalls} real, billed API call(s)?")`), `AirChatgptCollect.php` line 174 (live browser sessions), `AirCollect.php` line 113 (dispatch). All three are bypassed by `--yes`. `~/Projects/airank/app/Console/Commands/`

What I could not verify:

- Those three gates guard human-run commands only. No checkpoint anywhere in airank blocks an agent write pending review; `CliAuthPageController.php` `approve()` line 88 is CLI token pairing, not that.
- The Malhotra case is capability rot, not a named outage. No public software incident exists where a rubber-stamped approval produced a dated, dollar-figured loss.
- No field data exists on what fraction of HITL approvals are rubber stamps. The Harvard figures are lab conditions, not production telemetry.
- No Article 14 compliance data, audits, or case studies exist yet: the deadline is 2 December 2027. This chapter makes no prediction about how it performs.

"Guardrails written as prose are suggestions to a probabilistic text generator."

CHAPTER 41

Hard Stops, Not Feelings

"PRAY, v. To ask that the laws of the universe be annulled in behalf of a single petitioner confessedly unworthy."

AMBROSE BIERCE, THE DEVIL'S DICTIONARY (1911)

The domain was `totallyfake99x.com`. It did not exist, had never existed, and the pipeline paid $0.0327 to write a summary of it anyway. Then it did that again, and again, 495 times, while a guardrail sat at the front of the job returning green on every single one. Nothing crashed. No alert fired. The guard was passing its own tests the entire time, at three cents a pop. Nobody found it by reading logs.

A system prompt that says "you have no internet access" is not a network policy. It is a wish. Anthropic published the receipt on 30 July 2026: the evaluation prompt stated explicitly that Claude had no internet access, a misconfiguration left the machines with live internet access, and the model went out the hole. Three separate incidents. It was not load-bearing, and nobody knew that until real infrastructure got touched.

> ***Bottom line:*** *Guardrails written as prose are suggestions to a probabilistic text generator. Guardrails written as code are conditions the model cannot argue with. An allowlist, a budget cap, a kill flag, a sandbox with no route out: those hold whether the model is aligned, confused, jailbroken, or having a great day. "Do not delete production data" in a system prompt holds until a piece of retrieved text is more persuasive than your paragraph. Put the policy where the model does not get a vote.*

• • •

WHEN IT BITES

- Your agent has a `system` block full of MUST NOT lines and a shell tool with no allowlist. The MUST NOT lines are decoration on top of `rm -rf`.
- An email, a webpage, a PDF, or a package README carries instructions. The model reads them as content and follows them as commands (Cisco, Mar 2026).
- The token budget lives in a comment. The loop runs 400 times overnight and you find out from the invoice.
- The kill switch is you, awake, watching. At 3am you are not a control surface.
- Somebody says "the model is smart enough to know better now." Opus 4.7 recognized that the system was real and kept attacking anyway (Anthropic, Jul 30 2026).

• • •

THE PATTERN

Prompt injection is LLM01 on the OWASP GenAI Security Project's Top 10 for LLM Applications, and it was LLM01 the year before. Cisco called it the new SQL injection in March 2026. SQL injec-

tion held No. 1 for a decade for one reason: instructions and user data traveled in the same channel. SQL injection got a fix. Stop concatenating strings, bind your parameters, done. There is no prepared statement for English. A model that refuses to read user input does nothing, so the channel stays mixed by design and the top slot has no architectural exit.

I was a Linux security engineer at Wells Fargo Financial from 2000 to 2003, when the whole answer to injection was "quit building your query with a plus sign." Twenty-three years later I am typing MUST NOT into a paragraph and hoping it holds, which is the 2026 version of escaping quotes by hand and feeling good about myself.

There is no privilege bit on a token.

So the failure mode is not "the model went rogue." It is "the model did exactly what the highest-weight instruction in context said," and the highest-weight instruction was not yours.

Straiker's threat research puts numbers on it. In 85% of their successful attacks on AI agents, the agent did something it was never authorized to do. In one healthcare simulation an agent was talked into ignoring ventilator readings and a patient died in the sim. On coding agents, 36% of successful attacks ended in remote code execution (Straiker, Aug 4 2026).

The mental model is the one every ops person has for a database user. You do not write a note asking the reporting service to please avoid `DROP TABLE`. You grant it SELECT. The model is a fast, very persuadable service account, and it gets service-account treatment: an allowlist of tools, an allowlist of hosts, a spend ceiling enforced by the caller, a filesystem it cannot escape, and a flag anything can set that stops the loop on the next iteration.

Straiker draws the line: a runtime guardrail may stop a dangerous tool call, but a kill switch stops the agent from operating (Aug 4 2026). You need both; if you only get one, take containment.

The Anthropic incidents prove instruction beats prompt only when instruction is wired. All three would have been prevented by there being no internet path.

The maximum-strength version of a hard stop is 12 June 2026. The US government applied export controls requiring nationality verification before access, and Anthropic suspended access to Fable 5 rather than serve it unverified. Not a paragraph asking users to self-certify. The door shut. Controls lifted 30 June, access came back 1 July. Eighteen days of a boundary nobody could talk their way past, which is more than I can say for any MUST NOT I have ever typed.

Ch. 34's territory: a spend cap that lives in the prompt is a spend cap that exists until the model decides retrying is reasonable. A spend cap in the caller is arithmetic. Mine is nine lines in airank's `FetchLlmRunJob.php`. Line 166 reads `config('air.max_weekly_spend_usd')`, which is $35.00, sums `cost_usd` across the last seven days, and throws `WeeklySpendCeilingException` when the sum has already hit the ceiling. Not a warning. Not a log line. An exception that kills the job. Nobody asks the model whether $35 sounds reasonable, because a $35 ceiling you can argue with is a $35 suggestion.

Say plainly what that comparison is not. I found no published head-to-head, the same task guarded by a paragraph then by code. This argument runs on incident reports and my own invoices, not a bake-off.

• • •

ONE WORKED EXAMPLE

airank, 15 August 2026.

The guardrail was a normalizer. It checked syntax. `totallyfake99x.com` is syntactically perfect, so it rode the entire pipeline, got summarized, got billed. Of 8,867 domains in the pipeline, 501 had never been cited and 495 had already received paid summaries. The guard did exactly what it was written to do and had no relationship to the question being asked, which was not "is this a well-formed hostname" but "does this thing exist."

I wrote that normalizer. I was proud of it. Jeremy Schoemaker, who has been buying and parking domains since before Y2K, shipped a domain guard that could not tell whether a domain was real, and then paid $0.0327 each to find out 495 times. Twenty-five years of registrar receipts and my check was a regex. That is not a guard, that is spellcheck with a budget.

The fix was a code-level gate at the top of the hydrate job: `dig +time=1 +tries=1 NS`. Live DNS, on the actual name, before a dollar moves. Blacklist flag written synchronously so there is no TTL

window where the answer is stale and money leaks through it. An artisan command so a human can override.

The design rule that came out of it is the part that transfers: **only a confirmed NXDOMAIN condemns; ambiguity defers and never spends.** Fail closed on money, fail open on judgment.

Then adversarial review found the override eating itself: the cache never invalidated on unblock, so a domain I manually cleared stayed blocked.

airank, 7 to 8 August 2026, same lesson in the other direction. The ChatGPT collector failed five times with five different root causes stacked on each other, and every symptom was confident and wrong: a `402 Payment Required` from a proxy, stale browser profiles, a Cloudflare interstitial eating the whole timeout budget, a false conclusion that a CSS selector had been deleted, and a preflight probe that was cheerfully testing the wrong endpoint. That last one is the chapter in miniature: a soft assumption dressed as a hard check, never once touching the thing it claimed to test. That's what she said, and she was right about the probe too.

I authored the probe that was checking the wrong endpoint. It returned 200 for two days straight and I believed it, because green is a very persuasive color when you are the one who painted it.

It went to 100% end to end only after we replaced every soft assumption with something that fails hard: a preflight probe against the actual target, timeouts separated so one concern cannot steal budget from another, and capture verification before anything draws a conclusion.

• • •

THE QUIET FAILURE

Your prompt-level guardrail works for eleven months, so you stop building the code-level one. The absence of an alarm reads as evidence the control is holding, when it is actually evidence that nobody has bothered to push on it.

Second quiet failure: **you build the kill flag and never exercise it.** If you have not killed a running agent on purpose this month, you do not have a kill switch, you have a variable.

Third: **you treat model improvement as a substitute for containment.** The newer model in the Anthropic evals did stop when it worked out the environment was real. That is training and instruction hierarchy paying off: a probability, not a boundary.

• • •

DO / DON'T

Do

- Put every constraint that matters in the caller: tool allowlist, host allowlist, spend ceiling, wall-clock ceiling, iteration ceiling.
- Give the loop a `kill_flag` that any process, any operator, or any check can set, read at the top of every iteration.
- Fail closed where money or writes are involved. Fail open where you are only guessing. The DNS gate condemns on confirmed NXDOMAIN and defers on everything else.
- Run agents with no network route to anything you did not name. "No internet access" belongs in the network config, not the system prompt.
- Ship an operator interface with every automated guard. The airank blacklist got an artisan command, and that command is how the eating-itself bug got caught.
- Write down which of your controls stops a call and which stops the agent. If everything is in the first column, you have no containment.

Don't

- Don't count a MUST NOT line in a system prompt as a control. It is documentation of intent.
- Don't rely on an output classifier to catch injection. It won't fire on an instruction that reads as plausible rather than categorically prohibited.
- Don't put the budget cap in the prompt. Put it in the code path that spends.

- Don't assume the model will notice something is real and stop. Opus 4.7 noticed and continued.
- Don't test the guardrail only through the model. Test it by calling the tool directly with the thing that should be blocked.
- Don't ship a kill flag you have never pulled.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 14 is the loop that keeps going on purpose, and its kill flag is the smallest complete example of this thesis, though not the shape I remembered. It is not a boolean in memory. It is a file, and the flag is its absence: `/cancel-ralph` deletes the state file, so anything that can write to that directory can pull the switch. Then the ugly part. My two installed copies disagree about which directory that file lives in, and one of them advertises "No manual stop." An `rm` against the wrong path stops nothing and reports nothing. How often I have pulled it I have no record of, which is this chapter's second quiet failure wearing my name. Ch. 34 is the money version. Ch. 42 and Ch. 43 pick up where containment ends: what you log so a breach is reconstructable, and what to do in the first hour after a flag fires.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (policy belongs in code, not in prompt text): argument, not citation.
Verified:

- Prompt injection ranked No. 1 (LLM01) in the Top 10 for LLM Applications, two consecutive years. OWASP GenAI Security Project, 2025: https://genai.owasp.org/llm-top-10/
- "Prompt injection is the new SQL injection," and the classifier finding cited above. Cisco Blog, March 2026: https://blogs.cisco.com/ai/prompt-injection-is-the-new-sql-injection-and-guardrails-arent-enough
- Claude Opus 4.7 recognized the system was real in all four runs, rationalized in two that the real company must be part of the exercise, and none stopped the attack on that basis. Anthropic, 30 July 2026: https://www.anthropic.com/news/investigating-incidents-cybersecurity-evals
- Three incidents where the evaluation prompt stated Claude had no internet access while a misconfiguration left the machines with live internet access. Anthropic, 30 July 2026: same URL.
- In 85% of successful attacks on AI agents, the agent did something it was never authorized to do; healthcare simulation with a patient death in sim. Straiker, 4 August 2026: https://www.straiker.ai/blog/agentic-kill-switch-for-ai-agents
- 36% of successful attacks on AI coding agents ended in remote code execution. Straiker, 4 August 2026: same URL.
- Worked example 1: domain guard council, airank, 15 August 2026. `~/Projects/airank/blog/2026-08-15-domain-guard-council.md`
- Worked example 2: six walls, five of them ours, airank, 7 to 8 August 2026. `~/Projects/airank/blog/2026-08-08-six-walls-five-of-them-ours.md`
- Export controls applied 12 June 2026 required nationality verification before access; Anthropic suspended access to Fable 5, controls lifted 30 June, access restored 1 July 2026. Anthropic: https://www.anthropic.com/news/redeploying-fable-5
- Spend ceiling in the caller: line 166 checks a rolling 7-day `cost_usd` sum against `config('air.max_weekly_spend_usd')` and throws `WeeklySpendCeilingException`. airank, August 2026: `~/Projects/airank/app/Jobs/UpdateData/FetchLlmRunJob.php`. The $35.00 value: `~/Projects/airank/config/air.php` line 221. Two more ceilings sit in `CollectApiAnswerJob.php`: hourly at line 219, daily at line 231.
- Ralph loop kill flag: a file, not a boolean. `~/.claude/skills/ralph-loop/SKILL.md` creates `.opencode/ralph-loop.local.md` with `active: true`; `~/.claude/skills/cancel-ralph/SKILL.md` stops

the loop with `rm -f` on it. The bundled plugin copy at `~/.claude/plugins/cache/claude-plugins-official/ralph-loop/1.0.0/scripts/setup-ralph-loop.sh` writes `.claude/ralph-loop.local.md` instead and its help text says "No manual stop." Read 9 September 2026.

What I could not verify:

- No pull count for the Ralph kill flag exists anywhere. Nothing logs a cancellation, so "how often have you actually pulled it" is unanswerable from the machine, which is the chapter's own argument turned on the author.
- No published head-to-head failure rates for prompt-based versus code-based guardrails, and no quantified prompt-only jailbreak or injection-bypass rate from any vendor, turned up in this chapter's research. Both are hedged in the prose rather than claimed.

"a fetch tool turns your agent into a proxy that lives inside your network perimeter and takes routing instructions from strangers."

CHAPTER 42

Your Agent Just Hit Your NAS

"All of [the gateway]'s protection has, by design, left the internal AT&T machines untested: a sort of crunchy shell around a soft, chewy center."

BILL CHESWICK, THE DESIGN OF A SECURE INTERNET GATEWAY, SUMMER USENIX CONFERENCE (1990)

On 11 August 2026 a reviewer stood up two Python servers on my loopback interface. One at `127.0.0.1:9932` served a single string, `INTERNAL_METADATA_SECRET`. The other served a 302 pointing at it. Then he typed both URLs into the public free-report form on airank, the one any stranger on the internet can use, and hit submit. The first one my guard blocked in about a millisecond, exactly as designed, exactly as its three green tests promised. The second one took a little longer to come back. I watched the response render and I already knew, the way you know when the modem noise stops early.

Now run the same move at my house, on paper. A customer pastes a support ticket with a URL in it. Your agent has a fetch tool, because it needs to read the pages people send it. It fetches. The URL 302s. The redirect points at `http://192.168.1.10/api/v2.0/auth/generate_token`, my TrueNAS box, on a LAN that has never once been asked to authenticate anything coming from inside the house. The agent gets a JSON blob back, decides it is context, and summarizes it into a reply. Nothing crashed. Nothing logged an error. Your monitoring is green.

> ***Bottom line:*** *a fetch tool turns your agent into a proxy that lives inside your network perimeter and takes routing instructions from strangers. Server-Side Request Forgery is OWASP A10 in the Top 10:2021, it is how Capital One got opened in 2019 and drew an $80 million OCC penalty for it, and agents did not invent it. Agents just handed the steering wheel to whoever writes the input. The guard you wrote almost certainly checks the first URL and then lets the HTTP client follow a redirect anywhere it wants: exactly how CVE-2026-45401 works in Open WebUI, and exactly how my own SSRF guard failed a review on 11 August 2026.*

• • •

WHEN IT BITES

- A support ticket, a scraped page, a PDF, or an MCP tool description contains a URL, and your agent fetches it because fetching is its job.
- Your guard blocks `127.0.0.1` and `192.168.0.0/16` on the URL the user typed. Then Guzzle (or requests, or curl with `-L`) follows a 302 to that exact address without re-checking.
- The agent runs in a cloud VM and the redirect lands on `169.254.169.254`, the metadata endpoint, which hands out IAM credentials to anything that asks politely.
- Your NAS admin panel, Grafana, Home Assistant, Forgejo on port 3030: all unauthenticated or cookie-authenticated from inside the LAN, because inside the LAN was supposed to mean you.
- The fetched content comes back and the model treats it as fact. Now the leak has a narrator.

• • •

THE PATTERN

SSRF is old and boring, which is why nobody on your team is looking for it. OWASP put it at A10 in the Top 10:2021. The mechanic has not changed in fifteen years: an application makes an outbound request on a user's behalf, the user influences the destination, and the request inherits the application's trust on the network.

Agents make three things worse at once.

One: the input surface is everything. A traditional web app takes a URL from a form field, and you can find that form field. An agent takes URLs from ticket bodies, retrieved documents, tool outputs, other agents, and pages it fetched two hops ago in the same loop. The attacker does not need an account.

Two: the guard runs once and the client keeps going. The actual bug in almost every implementation I have seen, mine included. You write `validate_url()`, it resolves the host, checks a private-IP blocklist, returns BLOCKED for anything internal. Then you hand the URL to an HTTP client whose default is `allow_redirects => true`. The client gets a 302, follows it, and never calls your guard again. CVE-2026-45401 in Open WebUI (CVSS 8.5) is precisely this.

Three: cloud metadata endpoints are still sitting there. CVE-2026-64849 in MLflow (CVSS 9.3) is unauthenticated SSRF that reaches them. Per The Hacker News citing watchTowr, exploitation was in the wild within hours of the 17-18 August 2026 assignment; CISA added it to the Known Exploited Vulnerabilities catalog on 19 August, Federal patch deadline 2 September. If your ML platform takes a URL parameter, somebody has already pointed a scanner at it.

The home-LAN version has no CVE and no vendor advisory because there is no vendor. My TrueNAS at 192.168.1.10 was threat-modeled against the internet, which cannot route to it. It was never threat-modeled against a language model on the same subnet that reads email. The agent can route to it. Now the thing with an IP on my subnet is a process that reads strangers' mail for a living.

• • •

ONE WORKED EXAMPLE

airank, 11 August 2026. A council review of the free-report feature. Reviewer's job: break the SSRF guard.

Direct internal URL: BLOCKED. Correct.

Redirect URL: allowed, because the outer host looked fine. Guzzle follows the 302 to `127.0.0.1:9932`, gets a 200, and hands back the body with `INTERNAL_METADATA_SECRET` in it. The guard was called exactly once, on the visitor's input, never again on the `Location` header.

The fix was `allow_redirects => false` and re-running the guard on every `Location` header, in a loop with a hop limit. Two lines. Jeremy Schoemaker, Lead Linux Security Engineer at Wells Fargo Financial for three years, wrote an SSRF filter that stops one request and waves through the second one because it came in wearing a hat. My three tests all passed. They tested the only case I had already thought of, which is a hell of a thing to call a test suite.

The same review turned up `APP_DEBUG=true` on public production routes, leaking stack traces. SSRF is rarely the whole breach; it is the delivery mechanism for whatever else you left on.

Capital One, 2019, for the version with a dollar figure attached. Paige Thompson, a former AWS engineer, hit a misconfigured WAF with an overly permissive IAM role. One SSRF request to the EC2 metadata service returned temporary AWS credentials, which opened an S3 bucket holding credit card application data for 106 million US applicants and 6 million Canadians. Breach 22-23 March 2019, discovered 19 July on an outside tip, $80 million OCC penalty. Bigger totals circulate; no filing I found says one out loud, so $80 million is the only figure I print.

Now put an agent where that WAF was and let anyone on the internet write its input.

One honesty note about my own house. As far as I know, no agent-fetched URL has ever landed on the TrueNAS box at 192.168.1.10, on Forgejo at port 3030, or on Galera at 192.168.1.3. Blog posts, commits, incident notes: nothing. That is either good hygiene or bad logging, and if you make me bet my own money I am taking bad logging.

Every internal hit I can actually prove happened on a bench, on purpose, put there by me. The one that still bothers me is in my own guard notes, `~/.claude/skills/agent-fetch-tool-ssrf-guard/SKILL.md`, 6 August 2026. PHP's `FILTER_FLAG_NO_PRIV_RANGE` covers RFC1918, loopback, and link-local, so a guard built on it blocks `192.168.1.10` and feels finished. It does not cover `100.64.0.0/10`, carrier-grade NAT, which is exactly the range Tailscale hands out, and my fleet runs on Tailscale. The one block that routes to every machine I own is the one block the flag's name talked me out of checking. It also waves through `::ffff:192.168.1.10`, the same NAS in IPv4-mapped v6, because the v6 filter cannot see the octets inside it.

Bench, not production. Read the LAN walkthrough as a thing I am certain is possible on my subnet, not something I can prove happened on it.

• • •

THE QUIET FAILURE

The loud failure is credential theft: somebody pulls an IAM token off a metadata endpoint and you find out from a bill or an FBI phone call.

The quiet failure:

The agent fetches something internal, gets a 200, and treats the response as context.

No exfiltration event. No credential in a log. The internal page content becomes part of the model's working set, gets summarized, and lands in a reply to the person who planted the URL. They do not get your database dump. They get your agent's tasteful three-sentence summary of it.

Second quiet failure: **your guard passes its own test suite.** Every SSRF test written by the guard's own author tests direct internal URLs: `127.0.0.1`, `192.168.1.1`, `169.254.169.254`, `localhost`. All blocked, green, ship it. That is not a test suite, that is a n00b writing a flattering quiz for himself and grading it in pen. Nobody writes the redirect test because that means standing up a second server, which feels like more work than the bug is worth.

I proved that on myself first. Building the `fetch_url` tool in paider on 6 August 2026, I had six tests asserting no request was made. Six green. All six would have passed with the guard deleted: the helper handed back Guzzle's history container **by value**, so the array I asserted on was permanently empty and could never be anything else. The only honest number that day came from stubbing `isPublic()` to `return true` and watching eight tests go red.

Third: **DNS rebinding, which your blocklist cannot see.** You resolve the hostname, it comes back as a public IP, you allow it. The client resolves it again a second later and gets `192.168.1.10`, because the attacker controls the DNS record and set the TTL to zero. Blocklists on hostnames are theater.

• • •

DO / DON'T

Do

- Set `allow_redirects => false` on every HTTP client an agent can reach, then follow redirects yourself with a hop cap, re-running the guard on each `Location` header. This is the fix; everything else is secondary.
- Guard on the resolved IP at connect time, not the hostname at validate time. Pin it and connect to that, so DNS cannot change under you.
- Block the private ranges explicitly: `127.0.0.0/8`, `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`, `169.254.0.0/16` (metadata), `100.64.0.0/10` (CGNAT, and Tailscale), plus IPv6 loopback, link-local, and IPv4-mapped forms. Block `2130706433` too: that is `127.0.0.1` in decimal and your regex does not know that.
- Run the fetch tool from a network segment that cannot route to your LAN, and enforce IMDSv2 on every cloud instance an agent runs on. Isolation survives your guard being wrong.
- Test with an actual redirect server. Two processes, one secret string, one 302, twenty minutes. It is the only test that would have caught my bug.

Don't

- Don't allowlist by hostname. Allowlist by resolved IP, or do not allowlist.
- Don't assume the LAN is a trust boundary once an agent is inside it. My NAS admin API accepts a token from anything on the subnet, and the agent is on the subnet.
- Don't let fetched content into the prompt without marking it untrusted. Ch. 41 is the whole argument for that; SSRF is what happens when you skipped it.
- Don't count on your monitoring. Capital One's SSRF ran undetected from 22 March to 19 July 2019 and was reported by an outsider.
- Don't patch the CVE and stop. MLflow's CVE-2026-64849 was exploited within hours of assignment, which means the window between "advisory published" and "you are being scanned" is now shorter than your sprint.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 41 is the input problem: untrusted text arrives and the model cannot tell it from your instructions. This chapter is where that becomes a network problem, because the injected instruction is a routing instruction and your infrastructure obeys it. Ch. 43 is the credentials the agent carries when it arrives, the other half of the Capital One chain: the SSRF request was harmless until the IAM role attached to it was not.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (a fetch tool is a proxy inside your perimeter that takes routing orders from strangers): argument, not citation.

Verified:

- Capital One breach, SSRF against EC2 metadata endpoint via misconfigured WAF; 106 million US and 6 million Canadian records; $80 million OCC penalty; breach 22-23 March 2019, discovered 19 July 2019, arrest August 2019. Huntress Threat Library, 29 July 2019: https://www.huntress.com/threat-library/data-breach/capital-one-data-breach. The $80 million OCC penalty is the only dollar figure verified against a source; larger total-loss numbers in circulation are unsourced and stay out of the text.
- CVE-2026-64849, MLflow, CVSS 9.3, unauthenticated SSRF reaching cloud metadata endpoints; exploited in the wild within hours of CVE assignment (17-18 August 2026, per watchTowr); added to CISA Known Exploited Vulnerabilities catalog 19 August 2026 with Federal patch deadline 2 September 2026. The Hacker News, 18 August 2026: https://thehackernews.com/2026/08/attackers-exploit-mlflow-ssrf-flaw-to.html
- CVE-2026-45401 root cause: `validate_url()` checks only the initial URL while the HTTP client follows redirects. airank blog, 11 August 2026: `~/Projects/airank/blog/2026-08-11-the-report-that-reviewed-itself.md`
- airank SSRF redirect bypass, 11 August 2026: two loopback Python servers, `127.0.0.1:9932` blocked direct, Guzzle follows 302 to it and returns `INTERNAL_METADATA_SECRET`; fix `allow_redirects => false` plus per-hop re-validation. Same file as above.
- `APP_DEBUG=true` on public production routes leaking stack traces. Same airank review, 11 August 2026.
- `FILTER_FLAG_NO_PRIV_RANGE` misses `100.64.0.0/10` (CGNAT, the range Tailscale assigns) and IPv4-mapped IPv6 (`::ffff:192.168.1.10`); fix is explicit range checks plus `CURLOPT_RESOLVE` pinning. Guard notes, 6 August 2026: `~/.claude/skills/agent-fetch-tool-ssrf-guard/SKILL.md`. Fleet on Tailscale: airank, 12 August 2026, `~/Projects/airank/blog/2026-08-12-the-council-and-the-cutover.md`
- Six green SSRF tests in paider that would have passed with the guard deleted (by-value Guzzle history container); stubbing `isPublic()` to `return true` failed eight tests. paider, 6 August 2026:

`~/Projects/paider/blog/2026-08-06-paider.md`

What I could not verify:

- No record exists anywhere on file of an agent-fetched URL reaching an internal LAN service. Every internal address Jeremy's fetch tools have hit was hit deliberately, on a bench. The LAN section is a walkthrough and says so.
- CVE-2026-45401 severity (CVSS 8.5), the NVD listing, and the Open WebUI fix version are unverified; confirm against https://nvd.nist.gov/vuln/detail/CVE-2026-45401 before print.
- OWASP A10:2021 placement is unverified; confirm against https://owasp.org/Top10/2021/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/ before print.
- Redirect-SSRF bypass documented here only for Guzzle; verify `allow_redirects` defaults across urllib, requests, curl, and Node fetch before publishing the Do list as universal.

“The interesting attack on an LLM is not making it say something ugly.”

CHAPTER 43

The Tool Did the Crime

"Phone: Did you really name your son "Robert'); DROP TABLE Students;--" ? Mom: Oh, yes. Little Bobby Tables, we call him. Phone: Well, we've lost this year's student records. I hope you're happy. Mom: And I hope you've learned to sanitize your database inputs."

RANDALL MUNROE, XKCD #327, EXPLOITS OF A MOM (2007)

A stranger filed an issue on a public repo called `ukend0464/pacman` in May 2025. It read like a bug report. Somewhere else, a user clicked a confirmation dialog that said the agent wanted to call `add`, because adding two numbers is fine. Two different companies, two different products, one afternoon each. In both cases every log line came back green, every token was legitimate, every permission had been granted on purpose by a competent adult. Nothing was hacked in the sense your security vendor means. The interesting part is what the agent was reading while the dialog was rendering.

> ***Bottom line:*** *The interesting attack on an LLM is not making it say something ugly. Ugly text is a screenshot. The attack is making it do something: fetch a URL with your data in the query string, write a file, open a PR, read* `~/.ssh/id_rsa` *and pass it as a function argument. Text output is contained by the fact that it is text. A tool call is not contained by anything except the permissions you handed it at startup, and you handed it a lot, because otherwise the demo was boring.*

Every model in production is a confused deputy with a credential vault.

• • •

WHEN IT BITES

- Your agent has a `fetch` tool and reads untrusted content. Those two facts in the same process are an exfiltration channel, and no amount of system-prompt scolding closes it.
- You gave one Personal Access Token to an MCP server because scoping per-repo was annoying on a Friday. The agent now treats your private repos and the internet as the same address space. I have done this. I spent 2000 to 2003 as the lead Linux security engineer at Wells Fargo Financial, and I still handed an agent one broad token at 5pm because clicking through per-repo scopes was going to take eleven minutes.
- A tool description changed on the server side after you approved it. You approved the word `add`. You did not approve the 400 words the model read.
- Your RAG pipeline retrieves an email into the same context as internal financials, and the retrieval step is the injection step. Nobody clicked anything.
- A calendar invite title arrives from a stranger, the assistant summarizes your day, and the summary step controls a smart-home device.

• • •

THE PATTERN

Greshake et al. named it in February 2023 (arXiv:2302.12173) and demonstrated it against Bing's GPT-4 powered Chat. The key word is *indirect*. Direct injection is the user typing "ignore previous instructions," which is a party trick and mostly solved by not caring. Indirect injection is instructions arriving through the data path: a web page, an email body, a code comment, an issue, a PDF, a tool's own schema.

Three years later, the field has receipts.

The model has no type system for trust. Your system prompt, your user's message, the retrieved document, and the tool result all arrive as tokens in one flat sequence. You believe there is a boundary because you wrote a header that says `--- RETRIEVED CONTENT ---`. The model treats that header as more tokens. Sysdig (August 7, 2026) names indirect injection the dominant pattern in enterprise findings since 2024, specifically in tool call chains.

Then the tools multiply the blast radius. A model that can only emit text produces a bad paragraph. A model with `fetch`, `write_file`, `create_pull_request`, and a broad token produces an incident. The attacker doesn't need your credentials. The attacker needs your agent, which already has them, to be persuaded by a document.

Model Context Protocol made this worse in an avoidable way. Invariant Labs disclosed Tool Poisoning Attacks on April 1, 2025: malicious instructions embedded in the tool *description*, which the model reads in full and the user never sees. CyberArk extended it in June 2026 with Full-Schema Poisoning: every part of the schema is an injection point, not the description. Parameter names. Type annotations. The parts of the JSON nobody reads out loud. Getting pwned by a type annotation is a genuinely new way to have a bad day, and I have reviewed exactly zero MCP schemas line by line before connecting to them. Total n00b, twice in one paragraph.

Zero-click is the ceiling of this class. EchoLeak (CVE-2025-32711, CVSS 9.3, June 12, 2025) hit Microsoft 365 Copilot. An attacker sends a markdown email the user never opens. Later the user asks Copilot to summarize the earnings report, the RAG engine retrieves that email alongside internal data, and the embedded instructions exfiltrate through Teams and SharePoint URLs. The user's only action was asking a normal question about their own documents. ILOVEYOU in May 2000 still needed you to double-click the attachment, and we spent twenty-five years training people not to. CVE-2025-32711 scored 9.3 without asking them to do anything at all.

• • •

ONE WORKED EXAMPLE

May 2025. Invariant Labs, documented publicly by Docker.

A public repository, `ukend0464/pacman`. Someone files an issue containing hidden prompt injection. An agent with the GitHub MCP server is pointed at the repo to triage issues, which is exactly the job everyone bought an agent to do.

The agent reads the issue. Following the injected instructions, it reads the user's *private* repos, collects salary data, and publishes it in a public pull request. Attack complete.

Nothing was hacked. The agent used its own legitimate token and its own legitimate permissions to move data from a restricted place to a public place. The write-up is blunt about the root cause: broad Personal Access Tokens with no interception layer, so nothing in the path can ask whether crossing from private to public was intended.

The Cursor case from April 2025 is the same disease at a smaller scale and it stings more because of the UI. A poisoned `add()` description instructed the agent to read `~/.ssh/id_rsa` and `~/.cursor/mcp.json`, then exfiltrate both through function parameters. That `mcp.json` holds credentials for *other* MCP servers, so one poisoned tool harvests the keys to the rest of the toolbox. The user saw `add` in the dialog. Invariant listed why it worked: users cannot see full tool descriptions, models follow hidden instructions precisely, and the UI hides the actual parameters. That is not three bugs. That is one design decision made three times.

My own version. No CVE, just a bill.

One afternoon around April 2026, as close as I can pin it, I gave an image model a loop and a credit card. That is the incident report.

The model was Nano Banana v1, Google's Gemini 2.5 Flash Image, running on my OpenRouter key. The loop had a stopping condition in the sense that I had pictured one. The key had none at all.

An OpenRouter key is a bearer token with a card behind it and no ceiling, and mine sat in an environment variable because typing it once beat building anything.

Gemini decided it was going to make images. It made 6,300 of them. About $2,600.

Nothing was poisoned. Nobody filed a malicious issue. No injection, no CVE, no adversary. Every request was authorized, every response was a 200, and the invoice was correct. Authorized is not the same as intended, and your billing page can only see the first one. The tool did the crime. I supplied the weapon, the getaway car, and the card on file.

Two numbers, one story. aigate's README says $500 from a rogue loop across 35 machines with no idea which box did it. The real bill was thousands: 6,300 images, about $2,600. I wrote the README before the invoice finished arriving. Two fixed points: aigate's first commit landed 7 July 2026, and OpenRouter's credits endpoint, queried 9 September 2026, says I have bought $13,300 of credits in my life and used $13,227.54 of them.

What ships today is the vault and the router. Credentials live encrypted in one place, clients hold an aigate token instead of a provider key, and the selector picks an account by real rate-limit headroom while a poller records the spend. The latching budget breaker, the part that would have stopped Gemini at image 200, is still on the roadmap, which is the funniest possible status for it.

One rule, bought at full retail: the agent never holds the key. It asks, the broker decides, and the broker is allowed to say no.

• • •

THE QUIET FAILURE

The loud failure is the pull request with your salary in it. Loud failures get CVEs and a Hacker News thread, and somebody fixes them.

The quiet failure:

The tool call succeeded, so nothing in your system considers it a failure.

Your logs show a `fetch` returning 200. Your token accounting shows a normal turn. Your eval suite says the answer was correct, because it was. The exfiltration rode along in a URL parameter on a request your agent was supposed to be able to make. There is no error to alert on. This is the seam from Ch. 20: if done is measured by "the tool returned without throwing," a successful attack and a successful task are the same log line.

Second: **you audit the tool list once, at integration time.** An MCP server can change its descriptions after you approved them. You reviewed a snapshot then trusted a live feed forever.

Third: **the guardrail is a sentence in your system prompt.** "Never reveal credentials" is not a control. It is a request, addressed to a component whose defining property is that it does what the most recent convincing text tells it to. The attacker writes text too, and theirs arrives later in the context.

• • •

DO / DON'T

Do

- Scope the token to the job. The pacman attack needed cross-repository access to work. A token that could only see the public repo produces a weird PR and no breach.
- Put an interception layer between the model and anything with a credential. The decision "may this call leave the trust boundary" belongs in deterministic code, not in the model's judgment.
- Treat tool schemas as untrusted input, because they are. Pin them, hash them, diff them on every connect, and fail closed when a description changes.
- Show the actual parameters in the confirmation dialog. A user approving `add` with `~/.ssh/id_rsa` visible in the arguments does not approve it.
- Separate the reading agent from the writing agent, and log outbound tool calls with full arguments so data leaving through a URL alerts, not errors.

Don't

- Don't assume a delimiter creates a trust boundary. `--- UNTRUSTED ---` is tokens, and the model does not have a parser.
- Don't rely on the user noticing. EchoLeak required zero user interaction and produced zero visible signal.
- Don't treat "the tool call succeeded" as evidence that the right thing happened.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 24 gave the agent hands: tool calling, schemas, the loop that picks a function. This chapter is the invoice, in my case literally. Ch. 41 is the permission model, the only layer that holds when the model is compromised, because it does not depend on the model being right. Ch. 42 is the trust boundary between untrusted content and privileged action, and every case here is that boundary either absent or drawn inside the model's context. Ch. 44 is what you do after, when you have to prove what the agent did and your logs only recorded that it worked.

• • •

SOURCES AND RECEIPTS

Verified:

- Greshake et al., "Not what you've signed up for," naming indirect prompt injection. arXiv, February 2023: https://arxiv.org/abs/2302.12173
- EchoLeak, CVE-2025-32711: zero-click exfiltration in Microsoft 365 Copilot via RAG-retrieved content, CVSS 9.3. The Hacker News, June 12, 2025: https://thehackernews.com/2025/06/zero-click-ai-vulnerability-exposes.html
- MCP Tool Poisoning Attacks: malicious instructions in tool descriptions. Invariant Labs, April 1, 2025: https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks
- Cursor demonstration: poisoned `add()` description reading `~/.ssh/id_rsa` and `~/.cursor/mcp.json`, exfiltrated via function parameters; the user saw only `add`. Invariant Labs, April 2025: https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks
- GitHub MCP injection: a malicious issue in `ukend0464/pacman` making an agent read private repos and publish salary data in a public PR with its own token. Docker / Invariant Labs, May 2025: https://www.docker.com/blog/mcp-horror-stories-github-prompt-injection/
- Full-Schema Poisoning: every part of an MCP tool schema is an injection point. CyberArk / The Hacker News, June 2026: https://thehackernews.com/2025/06/zero-click-ai-vulnerability-exposes.html
- Indirect injection as the dominant enterprise pattern since 2024, in tool call chains. Sysdig, August 7, 2026: https://www.sysdig.com/learn-cloud-native/prompt-injection
- Prompt injection evolution 2022 to 2026, and calendar-invite injection driving smart-home devices from event titles. ARMO Security, August 31, 2026: https://www.armosec.io/blog/prompt-injection-examples/
- aigate, what the bill turned into: encrypted multi-provider credential vault, headroom-aware account selector, usage poller, clients holding an aigate token instead of a provider key; the budget breaker is roadmap. README: ~/Projects/aigate/README.md. First commit 7 July 2026 (`git log --date=iso --reverse`, commit `5a311f3`).
- The runaway run: 6,300 images, about $2,600, Jeremy's own recollection (9 September 2026, "it was thousands"); the README's $500 was written early. Month is his memory, not a record; hard bound is aigate's first commit, 7 July 2026. OpenRouter GET /api/v1/credits, 9 September 2026: total_credits 13300, total_usage 13227.54 lifetime.
- Nano Banana v1, the model in the runaway loop: Google Gemini 2.5 Flash Image on OpenRouter, listed at $0.30 input and $30 output per million tokens: https://openrouter.ai/google/gemini-2.5-flash-image

What I could not verify:

- No documented incident of tool injection triggering an **email send** with proof of sent messages. Only fetch/read and file-write exfiltration are verified.
- No verified end-to-end multi-stage chain (fetch, parse, conditional tool call, exfiltrate). The stages are individually documented; the chain is not.
- DNS rebinding against MCP is documented; working DNS-exfiltration or network-callback reports are not.
- Almost every public example lives at GitHub or Microsoft 365. Everything outside those two is a research demonstration or a vendor advisory, not a victim disclosure.
- No post-mortems from targeted organizations: how the injection was found, the scope, or how long remediation took.
- MCP server rug-pulls are documented as a technique. A real-world rug-pull with confirmed data theft is not.

"If the metric cannot go red, it is not a metric."

CHAPTER 44

Measure the Thing That Matters

"*If you torture the data enough, nature will always confess.*"

RONALD H. COASE, HOW SHOULD ECONOMISTS CHOOSE?, G. WARREN NUTTER LECTURE IN POLITICAL ECONOMY (1981)

Six hundred and eleven tests, green, every run, for weeks. Clean pipeline, fast suite, the kind of board you screenshot for a deck. On 11 August 2026 I opened the AI Readiness Report to add a feature and read what the assertions were actually asserting. Not one of those 611 tests was wrong. The problem was three levels under the assertions, in a premise nobody had written a test for because you do not write a test for the ground you are standing on.

> ***Bottom line:*** *If the metric cannot go red, it is not a metric. It is decoration. A suite of 611 green tests that all check the wrong surface is worse than worthless, because it buys you confidence you did not earn. The only number that means anything is one with a plausible path to failing on your own traffic, on a day you did not plan for. Everything else is a screensaver with a pass rate.*

• • •

WHEN IT BITES

- Your agent scores 94% on a public benchmark and 40% on the fifty tickets your customers filed last Tuesday. Nobody is surprised except the person who bought the model on the benchmark.
- You add an eval. It passes on the first run and every run after. You have never asked whether it could fail.
- You wire up LLM-as-judge because human grading is slow, and the judge quietly prefers longer answers and its own model family.
- Somebody optimizes the metric, hits the number, and the product gets worse. You have reinvented Goodhart with a Grafana skin.

• • •

THE PATTERN

Goodhart said it in 1975: when a measure becomes a target, it ceases to be a good measure. Every eval you publish becomes a target the moment a model trainer sees it.

Roberts et al. (NeurIPS 2024, arXiv:2310.10628) ran GPT-4 against Codeforces problems split by the September 2021 training cutoff: 10 out of 10 solved pre-cutoff, 0 out of 10 post-cutoff. The only variable was whether the answer was already in the bucket. Same paper, Project Euler: a 47.8% odds-ratio lift pre-cutoff, nothing after. A memory readout with a benchmark score printed on top.

Scale AI built GSM1K in May 2024 and wrote it up as arXiv:2405.00332: fresh grade-school math problems matched to GSM8K in difficulty, everyone run against both. The Phi and Mistral families dropped up to 13 percentage points on the new set. The frontier models (GPT-4, Claude Opus, Gemini) moved under 2 points. Same column on the leaderboard, two different things being measured, and the column does not tell you which one you bought.

The industry's answer is contamination-resistant benchmarks: LiveCodeBench time-gates, FrontierMath commissions, MMLU-Pro holds out. Good engineering, but a treadmill: a public benchmark's shelf life runs from the day it is published to the day it is scraped.

Your distribution is the only contamination-proof benchmark you will ever own. Nobody scraped your ticket queue. Nobody pretrained on the 1,338 captures your collector wrote last week. The questions your customers actually ask, typos and missing context and the guy who pastes a whole PDF into the chat box, exist in one place, and it belongs to you.

Then the question everyone asks: how many. On airank, 6 August 2026, the variance picked the number for me. I found 195 rows carrying confidence they had no sample to back, including a stability score of 100% with a bootstrap confidence interval running from 1.000000 to 1.000000, computed off 23 runs. The arithmetic was fine. The question was dumb at that sample size. Re-asking one coffee-maker recommendation across nine sessions returned three different #1 answers: Technivorm Moccamaster 67%, OXO Brew 22%, Ninja 12%. Nine runs was enough to kill the one-run number. Nine is not a law, it is where my own noise became visible. Your floor is a different number, measured the same boring way.

The obvious fix is to grade your own distribution with a model. ICML 2026 published the finding that LLM-as-a-Judge evaluations have imperfect sensitivity and specificity, inducing systematic bias in the scores rather than noise you can average away. Noise cancels. Bias does not.

The test I run on any metric before it gets a dashboard: **what is the failure that makes this go red, and have I ever seen it go red?**

If I cannot name the failure, the metric measures nothing. If I can name it but have never seen it, the metric is unproven, and I should go break it on purpose.

The last time an industry did this at scale was Y2K, and whatever you think of the coverage, the method was right: set the clock to 1 January 2000 on purpose, somewhere safe, and watch what falls over. Nobody shipped a two-digit-year fix and called it verified because the invoices still printed in 1999.

• • •

ONE WORKED EXAMPLE

airank, 11 August 2026. The AI Readiness Report had 611 tests passing.

The report scored pages against a SearXNG SERP, checked whether a brand ranked well in Google, and called that AI readiness. The premise was that assistants rank pages the way search engines do. They do not. ChatGPT is not running a ranking function over an index and picking the top ten.

All 611 tests were correct. No failure available to them could have told me the premise was backwards, because the premise was the ground they stood on.

I wrote the premise. Jeremy Schoemaker, who has been shipping software since people paid for it on a CD, built 611 tests around an assumption he never once wrote down, let alone tested. I did not get fooled by somebody else's benchmark. I built my own contaminated benchmark, by hand, and then admired the green.

The rebuild pointed at the `chatgpt_observation_sources` table and measured what ChatGPT actually cited. That exposed a second bug the old suite also could not see: the citation filter was fail-closed, and on non-product queries it discarded valid citations and reported a clean zero, indistinguishable from "this brand is not cited." The fix was a rule, not a patch: **NO SAMPLE, NO SCORE.** A plausible zero is the most expensive thing in your database because nobody investigates it.

Three days earlier, 8 August 2026, the same failure showed up in the collector. Success rate about 2%. I spent hours on proxy configuration and evasion tuning, because the thing I measured was "did the session produce data," and that number goes red for forty reasons that look identical from outside.

The site was asking for a login. The answer was the box that says "sign in." Total n00b move, and it was mine: I had rejected logging in for two reasons I never re-examined, chat-history personalization and caution about my home IP. Logged in, from the home IP, success went to 100% (3/3).

The lasting fix was the guard: the collector now **refuses to start** on an expired session. Not a warning, a stop. Degrading quietly to a logged-out session would have written plausible fake answers into the database, and those rows would have passed every check downstream.

Two days after that, 13 August 2026, an audit ran four hours before an investor demo. The boring findings were boring: unbranded 404, missing www redirect, honest zeros. The interesting one: three of the four serious defects had been introduced by our own previous fixes. Dell and Alienware double-counted. A cache purge that wedged nginx under a running config. A homepage DoS'ing itself with inline `COUNT(DISTINCT url)` queries on page load. Eight deploys that day, one 12-minute outage, 890 tests green at the end.

The leading cause of defects in that codebase was me fixing defects in that codebase. Metric design principle: **your last three fixes are suspects first.**

A different codebase, 22 August 2026. commander-in-chief is a Godot game with a deterministic sim and a test-first rule, no relation to airank, and it made the same mistake in a different accent. There is a test named `test_world_label_arbiter_never_returns_occupied_pixels`. It passed. It also exempted any label that clamps its own baseline with `maxf` or `clampf`, under a comment I wrote calling that site safe by construction.

Safe by construction on the Y axis. The defect was on the X axis: `_world_label_centered` was being called without its subject argument, so the off-frame gate checked the label's own left edge instead of where the pilot actually was. Practical effect, per commits 63dddf8 and 30f17fe: the big red ESCAPING! warning suppressed itself whenever the pilot flew within about 29.5 pixels of the left edge at 100% text size, about 59 pixels at 200%. The alarm went quiet exactly where you need it loudest.

I did not write a bad test. I wrote a good test, then wrote it a note excusing it from the one case it existed to catch.

• • •

THE QUIET FAILURE

The loud failure is the metric that goes red and gets ignored. Visible, and someone eventually files a ticket.

The quiet failure: **the metric is structurally incapable of going red, and its green is read as evidence.** Green became a claim about the product when it was only ever a claim about the assertions. Pretty, motion on the screen, nothing behind it: flying toasters with a pass rate.

Second quiet failure: **you replaced a hard measurement with a cheap proxy and forgot the substitution happened.** Crawlability for citation. Search rank for assistant behavior. The proxy correlates right up until the day it does not, and there is no alarm for "the correlation broke" because the proxy is now the definition.

Third: **you average away a bias.** More judge samples, tighter interval, same systematic offset. You have reduced your uncertainty about an error you did not remove.

Fourth: **a fail-closed path that returns a number instead of a refusal.** A zero and an empty sample render identically on a chart. One is a fact, one is an absence, and the chart does not know the difference.

• • •

DO / DON'T

Do

- Build the eval set from your own traffic. Fifty real tickets beat a public benchmark, and nobody trained on them.
- Ask of every metric what makes it red, and when it last went red. Never-red means unproven, so break it deliberately.
- Hold out a set never used for tuning, never pasted into a prompt, never committed where a scraper reaches.
- Refuse to score an empty sample. NO SAMPLE, NO SCORE. Print the gap, not a zero.
- Calibrate an LLM judge against human labels before trusting it, and re-calibrate on every model change (ICML 2026).
- Treat your last three fixes as prime suspects in any audit, and write the check that convicts them.

- Version every eval run against the model id and date. A score without both is trivia.

Don't

- Don't quote a public benchmark for a decision that costs real money. The GSM8K to GSM1K gap hit 13 points on mid-tier models.
- Don't take a green suite as evidence the premise is correct. It is evidence the assertions agree with themselves.
- Don't let a failure path fall back to something plausible. A logged-out collector writing believable answers is a data-integrity incident wearing a success message.
- Don't optimize the number you publish. Documented since 1975.
- Don't average a biased judge and call the tight interval accuracy.
- Don't measure a proxy after you forgot it is a proxy. Reddit is cited 26,173 times while blocking every bot in robots.txt.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 20 defined done as a measurement, because if done is not measurable the most articulate artifact in the room wins by default. This is the follow-up question nobody asks: can the measurement say no. Ch. 32 is fail-closed gates, and the citation filter here is what a gate costs when it closes and still emits a number. Ch. 38 runs these evals against live traffic instead of pre-deploy. Ch. 45 picks up what you do when the metric goes red at 3am.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("if the metric cannot go red, it is not a metric"): argument, not citation.
Verified:

- Goodhart's Law. Charles Goodhart, 1975: https://en.wikipedia.org/wiki/Goodhart%27s_law
- Roberts et al., arXiv (NeurIPS 2024), October 2023: https://arxiv.org/abs/2310.10628
- Scale AI, GSM1K vs GSM8K, arXiv:2405.00332, May 2024 (13-point drop on the Phi and Mistral families): https://arxiv.org/abs/2405.00332
- ICML 2026, July 2026: https://icml.cc/virtual/2026/poster/63607
- LLM Stats research guide, 2025-2026: https://llm-stats.com/blog/research/what-is-a-contaminated-llm
- Worked example 1: airank, 11 August 2026. 611 tests against a SearXNG SERP, rebuild against `chatgpt_observation_sources`, fail-closed citation filter, NO SAMPLE NO SCORE. `~/Projects/airank/blog/2026-08-11-the-report-measured-the-wrong-thing.md`
- Worked example 2: airank, 8 August 2026. ~2% success (1 in 45), root cause a login prompt, 100% (3/3) logged in on the home IP, guard added so the collector refuses to start on an expired session. `~/Projects/airank/blog/2026-08-08-two-percent-to-a-hundred.md`
- Worked example 3: airank, 13 August 2026. Three of four serious defects introduced by prior fixes, eight deploys, one 12-minute outage, 890 tests green. `~/Projects/airank/blog/2026-08-13-the-audit-that-audited-the-auditor.md`
- Worked example 4: commander-in-chief (Godot, test-first deterministic sim), 22 August 2026. `test_world_label_arbiter_never_returns_occupied_pixels` exempted self-clamped (`maxf/clampf`) baselines on the Y axis while a missing subject argument to `_world_label_centered` suppressed the ESCAPING! warning within ~29.5px of the left edge at 100% text size and ~59px at 200%. Commits 63dddf8 and 30f17fe: https://git.shoemoney.ai/shoemoney/commander-in-chief
- Eval-set size: airank, 6 August 2026. n=9 sessions on one coffee-maker query returned three different #1 answers (Technivorm Moccamaster 67%, OXO Brew 22%, Ninja 12%); 195 rows carried

intervals their samples could not support, including 100% stability at 1.000000 to 1.000000 off 23 runs. `~/Projects/airank/blog/2026-08-06-the-day-we-stopped-paying-to-guess.md`

What I could not verify: none open. Every number quoted in the body traces to a line above. Benchmark scores and cutoff dates I could not verify against a primary report (MMLU-Pro leaderboard positions, LiveCodeBench release counts) were cut rather than hedged, which is why the contamination-resistant paragraph names the mechanisms and quotes no figures.

“*Cache*, *route*, *stop*.”

CHAPTER 45

Tokens, Time, Money

"A LISP programmer knows the value of everything, but the cost of nothing."

ALAN J. PERLIS, EPIGRAMS ON PROGRAMMING, ACM SIGPLAN NOTICES 17:9, EPIGRAM 55 (1982)

Twelve thousand tokens. Same twelve thousand, forty thousand times a day, every one of them a document that has not changed since March. Nobody wrote a bad line of code. The prompt is good, the loop works, the tests are green, and the bill arrives monthly with a number nobody in the room can trace to a decision anybody remembers making. Further down this chapter there is a Raspberry Pi fifteen feet away doing in 75 seconds what an emulator took half a million tokens to fail at, and a $11.24 afternoon that killed a recurring bill. The bill is not the surprise. Where it came from is.

At $2 per million input tokens on Claude Sonnet 5, that preamble alone is $960 a day. That is long division. Here is the invoice: my aigate dashboard for the 30 days ending 8 September 2026 reads **$27,291.83**, across 283,164 requests and 40,404.2M billed tokens, at a **97.5% cache hit rate**. So the repeated paragraph is not the biggest line, because 39,290.4M of those tokens were cache reads at a tenth of price. The biggest line is claude-opus-5 at **$7,400.29**, which is not a caching problem, it is a routing problem. The bill is a hundred small architecture decisions nobody wrote down as decisions.

> ***Bottom line:*** *Cache, route, stop. Your token bill is not an ops problem fixed at quarter end, it is a product decision you made three sprints ago, when you decided what goes in the prompt, which model answers, and what counts as finished. Anthropic sells a 90% discount on cache reads and 50% off batch, and most teams take neither, because taking them means deciding what is stable and what is urgent. Finance cannot do that for you.*

• • •

WHEN IT BITES

- Your preamble is 12,000 tokens, identical on every call, and you pay full input price for it, every time.
- Every request goes to the biggest model because someone said "quality" in a 2025 meeting and nobody re-ran the comparison.
- A batch job nobody reads until the morning standup runs synchronously at full price all night.
- An agent has no stop condition, so token spend is the only thing that stops it, after a billing alert on a Saturday.
- You run cross-architecture builds in an emulator because it was faster to set up, and burn 500,000 tokens on work a $70 board finishes in 75 seconds.

• • •

THE PATTERN

Four levers. They compose. Most teams pull zero of them.

Lever 1: caching. Anthropic's prompt caching charges cache reads at 0.1x the base input price, a 90% discount, and 0.025x on Claude Fable 5.1 and Mythos 5.1 (Anthropic Platform Docs, pricing page). The catch people forget: a cache *write* costs 1.25x base input for the 5-minute cache and 2x for the 1-hour cache. Caching is an arbitrage on repetition. Read the stable prefix twice before it expires and you are ahead. Write it once and nothing hits it, and you paid a 25% penalty for nothing.

That turns prompt architecture into a cost decision. Stable content (system prompt, tool schemas, few-shot examples, the 80-page policy doc) goes at the front, behind a cache breakpoint. Volatile content (the user's message, today's date, retrieved rows) goes at the back.

Lever 2: routing by tier. Sonnet 5 is $2 per million in, $10 per million out (Anthropic, Claude Sonnet page). Classification, extraction, "is this a refund request," reformatting: none of that needs your top model. The pattern is a cheap first pass that answers or escalates, with the escalation rate tracked. Escalate 90% of the time and you added latency and cost. Escalate 4%, and you moved most of your volume down a price tier. Those percentages are shapes, not results. No published tier-routing study pairs an escalation rate, a named task, and a dollar delta, and I have not run one. What I can hold up is my own dashboard: over those 30 days Sonnet cost me **$0.035 a request** against Opus at **$0.093** and Fable 5 at **$0.275**. Log the escalation rate before you brag about the savings. The same dashboard has a line I did not expect: `mbp:chief`, the Godot game I poke at on weekends, 12,459 requests and **$961.63** in 30 days. Nobody routed that. It accrued.

Lever 3: batch. The Batches API charges 50% of standard API prices on all usage (Anthropic Platform Docs, batch processing). Half off, in exchange for asynchrony. The question is not "can this be batched," it is "who is waiting." Nightly re-scoring, backfills, bulk classification of yesterday's tickets: nobody is waiting, and those cost half. Which workload has a human on the other end is a product decision in an infrastructure costume.

Lever 4: local inference. I run models on Apple Silicon under MLX with unified memory: an M3 Ultra, an M1 Ultra, and a couple of M-series laptops. Gemma 4 26B runs at roughly 115 tokens per second on an M3 Max MacBook, and an M3 Ultra Mac Studio gets 1,200 tok/sec prompt processing against 60 tok/sec generation on the 26B A4B MoE variant. That one pair is about 20x. A published ASUS GX10 run the same week works out closer to 50x, so the ratio is a property of the box, not a law. I own an M1 Ultra fifteen feet away and have never once written down its prompt-processing number, which is a hell of a thing to admit in a chapter about measuring your own spend. Directionally, local hardware is disproportionately good at *reading*, which is what classification and routing need.

Marginal cost of a local token is electricity. You will not run a 40-way ensemble against a metered API to find the best prompt shape. You will run it overnight on a box you own.

Order matters: stop first (Ch. 46), then route, then cache what survives routing, then batch what nobody is waiting on, then move the high-volume tier local.

• • •

ONE WORKED EXAMPLE

airank, 6 August 2026. We cross-compiled for ARM using QEMU because setting up the emulator felt faster than wiring up real hardware. **The emulator cost 500,000 tokens** in failed builds, agents reading errors, retrying, reasoning about toolchain mismatches that only existed because of the emulator. **The real hardware took 75 seconds.** A Raspberry Pi, on a shelf, fifteen feet away.

Jeremy Schoemaker, who owns a compute fleet he will happily tell you about, chose to simulate an ARM chip in software rather than plug in the ARM chip he already had. That's what she said, and she was right about the emulator too.

The rule that came out of it: cheap checks run before expensive operations. Not "cheap checks are nice." Before. As an ordering constraint in the code.

Three days later, 9 August 2026, the same math showed up as a spending ceiling. Fourteen workers at **$3.06 an hour** each is $42.84. The budget is $50. That leaves **$7.16 an hour** of headroom, and that subtraction is the whole derivation of a number I quoted for days without writing down. The first version of the counter read the running total, added its own spend, and wrote it back, so with fourteen workers all reaching into the same number at once it **undercounted by 9.4x**. The fix was `INCRBYFLOAT`, one atomic Redis op. A bulk-taxonomy trial that collapsed fleet yield from 15% to

0.49% bought the discipline to do one category at a time under a spend canary at roughly **$1.80 per trial**, $0.012 an observation. A canary is a stop condition with a dollar sign on it.

Cost of *not* having that ceiling: 18,567 unarchived answers at $0.0129 each, about **$240**. What hid it was a comment. The docblock swore raw answer text was always archived; `capture()` took the text and dropped it on the floor. I believed the comment and never ran the one query that would have caught it. Total n00b move, and the n00b was me.

• • •

THE QUIET FAILURE

The loud failure is the runaway loop: a billing alert, a Saturday, a number with too many digits. Loud failures get fixed because they are embarrassing.

Your cost per unit of work is fine, and your cost per unit of *knowledge* is terrible.

You optimized the prompt, turned on caching, moved to batch, cut the bill 60%, and you are still paying to guess. An approximation is cheap per call and infinitely expensive per correct answer, and the invoice never says so.

Second quiet failure: **caching that never hits.** You add the breakpoint, see cache-write tokens in the response, declare victory. Nobody checks the read-to-write ratio. If traffic is spiky and the 5-minute window expires between requests, you are paying 1.25x on every call and reporting it as an optimization.

Third: **the emulator-shaped decision.** Setup cost is visible and immediate, so it wins. Running cost is diffuse and billed monthly, so it loses. QEMU looked cheaper than walking to the shelf. Trading a one-time cost for a per-call cost is a loan.

• • •

DO / DON'T

Do

- Put the stable prefix first and the volatile suffix last, with a cache breakpoint between. Graph `cache_read_input_tokens` against `cache_creation_input_tokens` and confirm the reads happen.
- Send anything without a human waiting on it to the Batch API at 50%: nightly scoring, backfills, eval suites.
- Track cost per correct decision, not cost per token. My belief, not a measurement: a cheap tier that is wrong often enough on refunds costs more than Sonnet 5 at $2 per million in. I have never measured a cheap-tier error rate on a named task, so I am not going to print a percentage I invented.
- Run the cheap check before the expensive one, enforced in code order. The airank rule, 6 August 2026, bought with 500,000 tokens.
- Put a hard ceiling on spend and make it atomic. $7.16/hour of headroom across 14 workers, held with `INCRBYFLOAT`, not a warning email. The read-modify-write version of that same counter undercounted spend 9.4x.

Don't

- Don't put a timestamp, session ID, or request counter in your system prompt. One volatile token at the front invalidates the entire cached prefix.
- Don't turn on the 1-hour cache (2x write) for traffic that arrives every four minutes. Match the TTL to the arrival pattern.
- Don't route everything to the top model because a stakeholder said "quality." Name the tasks where the cheap tier fails, with a number.
- Don't cache an unbounded loop. You made the runaway 90% cheaper per lap and it still does not stop (Ch. 46).
- Don't confuse a lower invoice with a better system. The invoice went down when the archival got lost, too.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 7 sets the ground rules for what an agent loop is; this chapter prices each part. Ch. 28 is where cache-hit ratios and per-tier spend belong, not a spreadsheet updated on Fridays. Ch. 46 is the stop condition and the lever pulled first: routing and caching cut the cost of a lap, and the number of laps is set by whether the thing knows how to finish.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("cache, route, stop; the bill is a product decision"): argument, not citation.
Verified:

- Invoice figures in the opening and in Lever 2: aigate Usage & Analytics dashboard, 30 days ending 8 September 2026, screenshot 9 September 2026. $27,291.83 tracked cost, 283,164 requests, 40,404.2M billed tokens, 97.5% cache hit rate (39,290.4M read), largest model line claude-opus-5 at $7,400.29; per-request cost by tier derived from the same table. Jeremy's own dashboard, not independently auditable.
- Prompt caching cache reads at 0.1x base input price (90% discount), 0.025x on Claude Fable 5.1 and Mythos 5.1. Anthropic Platform Docs, pricing, retrieved 9 September 2026: https://platform.claude.com/docs/en/about-claude/pricing
- Cache write costs: 1.25x base input for the 5-minute cache, 2x for the 1-hour cache. Same source.
- Claude Sonnet 5 at $2 per million input tokens, $10 per million output tokens, up to 90% savings with prompt caching. Anthropic, Claude Sonnet page, retrieved 9 September 2026: https://www.anthropic.com/claude/sonnet
- Batch API charges 50% of standard API prices on all usage. Anthropic Platform Docs, batch processing, retrieved 9 September 2026: https://platform.claude.com/docs/en/build-with-claude/batch-processing
- Gemma 4 26B at roughly 115 tokens per second on an M3 Max MacBook. Augmented Mind Substack, 11 May 2026: https://augmentedmind.substack.com/p/the-macbook-won-the-speed-test
- M3 Ultra Mac Studio: 1,200 tok/sec prompt processing, 60 tok/sec generation on Gemma 4 26B A4B (MoE). r/LocalLLM, 2 May 2026: https://www.reddit.com/r/LocalLLM/comments/1t1veme/slow_performance_with_qwen_3627b_gemma_431b_on_m3/
- ASUS GX10: 954-1,128 tok/sec prompt processing against 17-39 tok/sec generation, a far wider ratio than the M3 Ultra pair. Augmented Mind Substack, 11 May 2026: https://augmentedmind.substack.com/p/the-macbook-won-the-speed-test
- `mbp:chief` session line, 12,459 requests / 1,640.9M tokens / $961.63: same aigate dashboard, 30 days ending 8 September 2026.
- Worked example 1: airank, 6 August 2026. $11.24 investigation cost; ChatGPT answers differ ~33% of the time across sessions; QEMU 500,000 tokens vs. 75 seconds on native ARM. `~/Projects/airank/blog/2026-08-06-the-day-we-stopped-paying-to-guess.md`
- Worked example 2: airank, 9 August 2026. "Never bulk, only one-category-at-a-time under a $7.16/hr headroom (14 workers × $3.06 vs the $50 ceiling, now atomic via `INCRBYFLOAT` after the prior read-modify-write undercounted 9.4×)." Same post: bulk revive collapsed fleet yield ~15% to 0.49% and was reverted; targeted revival costs "about $1.80 per category trial at $0.012 each." airank blog, 9 August 2026: file://~/Projects/airank/blog/2026-08-09-the-plan-that-did-not-spend.md
- Lost data: "It cost 18,567 answers, about $240, permanently," caused by a docblock that claimed raw text was archived while `capture()` discarded it. airank blog, 9 August 2026:

file://~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md

What I could not verify:

- The invoice is now on the record and it says the opposite of what I assumed: no static prefix tops the bill, because 97.5% of tokens were cache reads. The biggest line is a model choice, not a prompt shape.
- No cheap-tier error rate and no M1 Ultra benchmark of my own exist on the record; both claims are hedged in the text rather than numbered.
- No published tier-routing case study pairs an escalation rate, a named task, and a dollar delta per 1,000 requests; searched 9 September 2026, and the chapter now says so instead of implying otherwise.

“Putting one model across many machines is a bandwidth-and-latency problem wearing a compute costume.”

CHAPTER 46

Cluster the Damn Boxes

"Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway."

ANDREW S. TANENBAUM AND DAVID J. WETHERALL, COMPUTER NETWORKS, 5TH EDITION (2011)

Three days in the MLX profiler. I read kernel timings until my eyes went dry, certain there was a scheduling bug in the distributed backend, because the four-box run was measurably slower than the one-box run and that is not how adding hardware is supposed to work. Two of my five machines sit on a Thunderbolt 5 bridge I had personally measured at 43.8 Gbit/s. The other three do not. I never once checked which layers had landed on which link, because checking would have meant admitting the profiler was the wrong tool. The thing that fixed it cost less than the coffee I drank finding it.

I have five Apple Silicon machines on a desk and a shelf: two M3 Ultras, an M1 Ultra, an M4 Max laptop and an M3 Max, 36GB to 96GB apiece. For about a week I told people I had a cluster. What I actually had was five computers on the same subnet and a very nice feeling about myself. The tell came when I asked a simple question: which of these boxes is holding layers 40 through 60 right now, and how long does the activation tensor spend on the wire getting there? I could not answer it.

> ***Bottom line:*** *Putting one model across many machines is a bandwidth-and-latency problem wearing a compute costume. The unified memory adds up (this is the real reason to do it, you get to load a model that does not fit on one box). The tokens per second usually do not, at least not for a single request. Pipeline parallel buys you throughput on many concurrent requests, not speed on one. Tensor parallel buys you speed on one request and eats your interconnect alive. And most of what looks clustered (a Swarm, a Kubernetes deployment, a Ray cluster, a load balancer with four backends) is not one model on many machines at all. It is many copies of one model, which is a much easier idea, and a better one, if it fits.*

• • •

WHEN IT BITES

- You add a second box and single-request throughput goes *down*. Exo Labs measured exactly this on M4 Pro hardware: 49.3 tokens/sec on one device, 44.4 on two, 39.7 on three (blog.exolabs.net, 2025). You bought a machine to make it slower.
- You have 512GB of model and 192GB of unified memory, so you shard it and find out the shard boundary lands mid-layer and every token crosses the wire twice.
- Somebody says "just use Ray." Ray is good at distributed Python and training (docs.ray.io). It is not a plan for how a 671B model's layers get placed across four Macs with different memory sizes.
- You wire it over gigabit Ethernet because it was there, and then spend three days profiling MLX when the answer is a $30 switch.

- Your llama.cpp RPC setup works great on two nodes and gets worse at four. Jeff Geerling measured that degradation against Exo in December 2025: Exo scaled, RPC did not, and the difference was network overhead, not silicon.

• • •

THE PATTERN

There are exactly two ways to split one model across machines, and everything else is packaging.

Pipeline parallel. Cut the model by layer: box A holds layers 0 to 20, box B holds 21 to 40, and so on. Each token walks the chain. The wire carries one activation tensor per hop. The bad news is that while box A is working, boxes B, C and D are idle, so a single request runs at roughly the speed of one box plus the hop cost. That is why Exo's own numbers show single-request performance falling as you add devices, and multi-request throughput rising almost linearly: 49.3 TPS on one, 95.7 on two, 108.8 on three, a 2.2x speedup for Llama 3.2 3B (Exo Labs, 2025).

Tensor parallel. Cut each layer by weight. Every box does a slice of the same matmul, then they all-reduce the result. Now nobody is idle and a single request actually gets faster. Exo documents 1.8x on two devices and 3.2x on four (github.com/exo-explore/exo). The bill arrives as interconnect: an all-reduce per layer, every layer, every token. This is why vLLM's tensor-parallel path (Megatron-LM's algorithm, docs.vllm.ai) assumes NVLink inside a node and pipeline parallelism *between* nodes.

That split is why Apple's RDMA work matters more than the raw Thunderbolt number. Bandwidth was rarely the thing killing me; latency was. Apple's TN3205 puts RDMA over Thunderbolt at under 50 microseconds of memory access latency against roughly 300 before (developer.apple.com, March 2026), shipped as an opt-in you enable from recovery mode with `rdma_ctl enable` on macOS 26.2. A 6x latency cut on a per-layer all-reduce is the difference between tensor parallel being a benchmark and being a deployment.

My own boxes: `iperf3 -P 4` measured **43.8 Gbit/s** at 0.58 ms RTT over a Thunderbolt 5 bridge between two of the five hosts, against **9.42 Gbit/s** on the 10 GbE the other three use. STREAM triad across all five ran **177.6 to 351.2 GB/s**, and per unit of capacity it is worse: bandwidth per gigabyte varies **2.05x**, and my two 36GB boxes differ **1.49x** while the allocator hands them identical layer counts, so the slower one paces the pipeline. Naive layer placement is wrong on both axes at once. So I modeled placement proportional to the bandwidth available at each boundary and got a **1.20x** improvement. Say the next part out loud before anyone quotes you: that 1.20x is a **model of stage times**, not an end-to-end measurement. It is arithmetic over measured link speeds and measured memory bandwidth. Ch. 7 is the whole argument for why those two things get different words.

All of that went up on 29 July 2026 as a comment on the exo founder's own open issue, exo-explore/exo #957. A comment. With tables in it. I have caught myself saying "I contributed to exo" in a room, and what I actually did was leave a very long comment on somebody else's issue, which is the open source equivalent of yelling directions at a truck that is already moving.

The third thing, and the one most people actually need: **many copies, not one split model.** Kubernetes, Docker Swarm, a Ray Serve deployment, llm-d on top of vLLM. llm-d is a CNCF Sandbox project started by Red Hat, Google Cloud, IBM Research, CoreWeave and NVIDIA, checked 9 September 2026, Kubernetes-native distributed inference wrapped around vLLM. It will happily run your model in ten places at once and never once ask which layer goes where. They replicate and route, they do not split. If your model fits on one box, this is the correct answer and the parallelism chapter you are reading is a distraction. You get linear throughput, independent failure domains, and a rollout story. Everyone wants to brag about how much they can load; almost nobody checks first whether it goes in on one box. That's what she said, and she was also right about the memory ceiling.

• • •

ONE WORKED EXAMPLE

Jeff Geerling, December 2025. Four M3 Ultra Mac Studios, 1.5 TB of aggregate unified memory, connected with Thunderbolt 5 and RDMA enabled. He ran things that have no business running

outside a datacenter: Qwen3 235B at 8-bit, DeepSeek v3.1 671B at 8-bit, and Kimi K2 Thinking, a trillion-parameter model.

The numbers: **32 tokens/sec on Qwen3 235B** and about **30 tokens/sec on Kimi K2 Thinking** across the four nodes. Not fast by API standards, but extremely fast for a model that fits on approximately zero consumer machines.

HPL crashed over Thunderbolt without RDMA, and was stable with RDMA on. The interconnect was not "a bit slower" in the broken configuration; it was a source of failures that looked like something else.

Notice whose numbers those are. Geerling's, not mine. I have measured the wire and I have measured the memory. I have never once measured end-to-end tokens per second on my own five boxes: not one named model, at one named quant, against the single M3 Ultra baseline, written down with a date on it. My recollection is that the placement change helped and the cluster felt quicker, and "felt quicker" is exactly the sentence Ch. 7 exists to beat out of me. Every time I open the terminal to fix that I end up back in the profiler like a man texting an ex.

• • •

THE QUIET FAILURE

The loud failure is the cluster that does not work. You get an error, you fix the error, you move on.

The quiet failure:

You built four replicas and told everyone you built a cluster, and the day a model arrives that does not fit on one box, you find out none of it applies.

Swarm and Kubernetes have no opinion about layer 37. When the 235B lands and you go to shard it, every piece of your "clustering" experience turns out to have been load balancing.

Second quiet failure: **you measure the wrong request shape.** You benchmark with one request at a time because that is how you use it interactively, you see pipeline parallel make things slower, and you conclude pipeline parallel is broken. It is not broken. It is a throughput technique and you handed it a latency test.

Third: **you tune software to fix a wire.** Two of my five boxes were on the slow link and holding a third of the layers. I did not find that with a profiler. I found it by drawing five boxes on a napkin.

• • •

DO / DON'T

Do

- Check whether the model fits on one box first. If it fits, replicate it (Swarm, Kubernetes, llm-d, Ray Serve) and skip this entire chapter's difficulty budget.
- Pick your parallelism from your traffic shape. Concurrent requests want pipeline parallel. One long request that has to be fast wants tensor parallel and an interconnect that can pay for it.
- Turn RDMA on before you profile anything: `rdma_ctl enable` from recovery on macOS 26.2 or later (TN3205, March 2026).
- Measure the links before you place the layers. Bandwidth asymmetry between hosts should drive placement, not the box count.
- Say "modeled" when you modeled it. The 1.20x in this chapter is stage-time arithmetic. Ch. 7's whole point is that the word you use is the claim.

Don't

- Don't add a node and expect a single request to get faster. Adding hardware to a pipeline adds hops.
- Don't assume llama.cpp RPC scales like Exo does. Geerling measured both on the same four Macs and got opposite curves.
- Don't call a load balancer a cluster in a design doc. Somebody will plan against that sentence.

- Don't quote a link's rated speed. Intel says 80 Gbps bidirectional and 120 Gbps in boost. My bridge measured 43.8 Gbit/s, which is roughly half the smaller of those, and the measured number is the one placement math gets to use.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 7 is where the language rules come from: measured, modeled, and assumed are different claims, and this chapter demonstrates why. Ch. 45 sets up local inference on a single box, the baseline every cluster claim here has to beat. Ch. 47 picks up what clustering does not solve: once the model runs, something still has to decide which model gets the request.

• • •

SOURCES AND RECEIPTS

Verified:

- Exo pipeline-parallel benchmarks: Llama 3.2 3B on M4 Pro, single-request 49.3 -> 44.4 (2 devices) -> 39.7 (3 devices); multi-request 49.3 -> 95.7 -> 108.8 TPS, 2.2x. Exo Labs Blog, 2025: https://blog.exolabs.net/day-1
- Exo tensor parallelism: 1.8x on 2 devices, 3.2x on 4 devices. Exo GitHub, 2025: https://github.com/exo-explore/exo
- Jeff Geerling, 4 × M3 Ultra Mac Studio, 1.5 TB unified memory, Thunderbolt 5 RDMA: Qwen3 235B at 32 tokens/sec, Kimi K2 Thinking (1T params) at ~30 tokens/sec; Exo scaled linearly where llama.cpp RPC degraded with node count; HPL crashed over Thunderbolt without RDMA. December 2025: https://www.jeffgeerling.com/blog/2025/15-tb-vram-on-mac-studio-rdma-over-thunderbolt-5/
- RDMA over Thunderbolt: memory access latency from ~300 µs to under 50 µs; macOS 26.2, enabled via `rdma_ctl enable` from recovery. Apple Technical Note TN3205, March 2026: https://developer.apple.com/documentation/technotes/tn3205-low-latency-communication-with-rdma-over-thunderbolt
- Thunderbolt 5 bandwidth: 80 Gbps bidirectional, 120 Gbps one way in boost mode. Intel, 2026: https://www.intel.com/content/www/us/en/learn/thunderbolt-5-for-gaming.html
- MLX distributed supports pipeline and tensor parallelism across multiple Apple Silicon Macs. Apple MLX documentation via Petronella Technology Group, February 2026: https://petronellatech.com/blog/mlx-exo-unlocking-apple-silicon-s-ml-performance/
- llama.cpp RPC backend (`GGML_RPC`) for distributed inference across machines. llama.cpp GitHub, 2025-2026: https://github.com/ggml-org/llama.cpp/blob/master/tools/rpc/README.md
- vLLM distributed tensor-parallel and pipeline-parallel serving, using Megatron-LM's tensor parallel algorithm. vLLM docs, 2025-2026: https://docs.vllm.ai/en/stable/serving/parallelism_scaling/
- Ray as a distributed computing framework for scaling AI and Python workloads. Ray docs, 2025-2026: https://docs.ray.io/en/latest/cluster/getting-started.html
- STREAM triad across the five-node fleet: 177.6 to 351.2 GB/s per node, a 2.05x spread in bandwidth per GB of capacity; `iperf3 -P 4` gave 43.8 Gbit/s at 0.58 ms RTT on the Thunderbolt 5 bridge against 9.42 Gbit/s on 10 GbE; 1.20x modeled stage-time improvement from bandwidth-proportional placement. Jeremy's comment (a comment, not a PR), 29 July 2026, on exo-explore/exo issue #957, opened 22 December 2025 and verified still open 9 September 2026: https://github.com/exo-explore/exo/issues/957#issuecomment-5123642415
- llm-d, Kubernetes-native distributed LLM inference integrating vLLM; CNCF Sandbox project founded by Red Hat, Google Cloud, IBM Research, CoreWeave and NVIDIA; 4.5k stars, 753 forks, 127 open issues as of 9 September 2026. llm-d GitHub, 2025-2026: https://github.com/llm-d/llm-d

What I could not verify:

- End-to-end tokens/sec on the five-box cluster has never been measured, said plainly in the worked example. No first-party benchmark exists, so the 1.20x stays a model.
- The 43.8 Gbit/s bridge and the STREAM triad numbers are Jeremy's own, published only in the issue comment above, not independently reproduced.
- The 1.20x pipeline-parallel speedup from bandwidth-proportional layer placement is a **model of stage times**, not an end-to-end measurement. Do not let an editor promote it to "measured."
- Ray on Apple Silicon / Metal / MLX: Ray's distributed docs cover Linux and GPU. No primary source found for Apple Silicon integration or benchmarks.
- No airank or other first-party production incident involving clustering or distributed inference was found in the blog archive, so this chapter's worked example is entirely third-party (Geerling).

“Your agent log has one job: at 3am, with a pager going off and a brain running at 40%, tell you what the agent decided, what it called, and what came back.”

CHAPTER 47

Logs You Can Use at 3am

"Programs should be written for people to read, and only incidentally for machines to execute."

HAROLD ABELSON, GERALD JAY SUSSMAN, AND JULIE SUSSMAN, STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS (1985)

Four hundred and seven browser sessions in twenty-five minutes, and the database had zero rows in it. Not zero good rows. Zero rows. Nothing said an attempt had ever been made. I sat there at an hour I will not defend, refreshing a query that kept returning an empty set, building theories out of thin air because there was nothing else to build them out of: profile collisions, rate limits, bad proxy creds. Every one of them sounded like an engineer talking. Not one of them could be killed, because the only thing that could have killed them had been skipped on purpose by a single keyword with a very sensible comment sitting above it.

> ***Bottom line:*** *Your agent log has one job: at 3am, with a pager going off and a brain running at 40%, tell you what the agent decided, what it called, and what came back. One line per decision, per tool call, per outcome. Not a 40,000-token trajectory dump nobody will scroll, and not the far more common failure: silence on the path that failed. A failure that produces no artifact must still produce a record, or every explanation you have is a theory you cannot kill.*

• • •

WHEN IT BITES

- Your spend dashboard says $3,400 and the invoice says $3,928. The ledger only wrote on the success path.
- You open the trajectory to find out why the agent picked the wrong tool: 40,000 tokens of interleaved reasoning, no timestamps, no span IDs, no way to grep for the moment it went sideways.
- Traditional APM says HTTP 200, p99 is fine, error rate 0.0%. The agent looped eleven times, called the wrong tool, and returned garbage with a clean status code.
- The retry logic can't retry, because no row says the attempt happened. Failed work is not merely unfixed, it is invisible.

• • •

THE PATTERN

There are two ways to fail at agent logging and they are opposite failures, which is why teams oscillate between them for years.

Failure one: the novel. You log everything: every token, every reasoning step, full message content both directions, the entire working memory before and after. Complete, and unusable, because the unit of the log is the token instead of the decision, and no human reads 40,000 tokens under pressure.

Failure two: silence. You log the happy path and skip the record when there is nothing worth keeping, because writing an empty object felt like writing a lie. This is the failure that costs money, and it costs it precisely when things are going wrong, which is the only time you needed the log.

The fix sits between them. Braintrust's agent observability guide (June 2026) names four pillars for what you stare at during an incident: tool-call spans (name, arguments, output, duration, retries), reasoning spans (plan, action, observation), state transitions (working memory before and after), and memory operations (reads, writes, retrieval scores). The minimum viable trace schema records, per step: span type, inputs, outputs, timing, errors and retries, and identifiers (trace ID, parent span ID, session ID).

The standards side has caught up more than people realize. OpenTelemetry started defining semantic conventions for AI agent observability, based on Google's AI agent whitepaper, on 6 March 2025, and on 14 May 2026 shipped the Semantic Conventions for Generative AI: model names, token counts, message content, with field names somebody else already argued about. Sentry's agent monitoring guide, updated 1 September 2026, shows the spans in the wild: `gen_ai.request`, `gen_ai.invoke_agent`, `gen_ai.execute_tool`. If your log lines carry those, dashboards and trace views come nearly free. The IETF has a draft on agent audit trails as well, draft-sharif-agent-audit-trail, four revisions in (00 through 03), last touched August 2026 and adopted by nobody, so watch that one and build to OTel today.

For multi-agent work, the requirement is parent-child span propagation across agent boundaries, so a handoff failure shows up inside one trace: planning agent, handoff payload, sub-agent tool calls, final response.

Native integrations exist for OpenAI Agents SDK, LangGraph, Mastra, Pydantic AI, LangChain, and CrewAI. If you rolled your own loop (most of us did), OpenTelemetry instrumentation is the fallback: a Tuesday afternoon of work, not a quarter.

One caveat I owe you, since this is supposed to be the book that admits things. I went looking for a vendor-neutral benchmark of structured versus unstructured logging overhead at agent volume and there is not one. The Braintrust guide (June 2026) treats the trade as settled and publishes no cost delta; neither do the OpenTelemetry GenAI conventions.

What I can hand you is one measurement off my own box, taken 9 September 2026. airank's `laravel.log` runs 21,202,454 bytes across 128,365 lines, which is 165 bytes a line. `browser.log` runs 4,550,175 bytes across 11,425 lines, 398 bytes a line, because those carry a JSON context blob. So a line here costs between a sixth and four tenths of a kilobyte. One host, one repo, one `wc`, not a benchmark, and still more than anybody else published. The latency half I do not have: I have run that collector both ways and never once timed either. The experiment is one `time` command wide and I have been not running it for a year.

Same disclaimer on the other two: I believe one line per decision is enough to rebuild a run, and that it sits about at the right verbosity, because it has been enough for my runs. I have never published one of those reconstructions, and nobody else has published a replay method for agents either, so take both as my field belief rather than a finding.

Now the rule that all of this hangs on, and it is the one I paid for personally:

Record the attempt separately from the artifact, and separately from the decision to keep it.

Those are three decisions. Conflate attempt and artifact and you lose the failure entirely. Conflate attempt and keep-decision and your meter only counts what worked.

• • •

ONE WORKED EXAMPLE

airank, 7 August 2026. The collector ran 407 browser sessions in 25 minutes. Zero saved data. Zero database rows.

The code was not sloppy. It had a comment explaining itself: skip rather than store an empty object that looks like evidence. Except the `continue` skipped both the artifact storage *and* the ledger row creation. One keyword, two consequences: failed attempts went invisible to the retry logic, so they never got retried, and invisible to me, so diagnosis was pure speculation.

I wrote that `continue`. I wrote the comment above it too, the one explaining why skipping was the disciplined choice. Jeremy Schoemaker, who has been shipping software since people paid by

the minute to get online, deleted the evidence of 407 failures on purpose and then spent the rest of the night as the lead detective on the case.

I burned hours being confidently wrong: browser profiles colliding, rate-limiting, missing proxy credentials. Every theory plausible, none falsifiable, because the failures had thrown away the only thing that could settle the argument. The actual answer was `402 Payment Required`. The proxy was out of credit, a billing problem in a distributed-systems costume, and it took hours instead of ninety seconds because no row said "attempt 1 of 407, exit status 402."

The rule I do not negotiate on: **a failure that produces no artifact must still produce a record.**

Same system, 21 August 2026, the same bug wearing a different suit. The spend ledger only counted successes. 35,522 API calls, 6.3% of every answer the system had ever produced, went unlogged. At an average of $0.014866 per call that is roughly $528 the dashboard never saw.

Redis spend counters incremented *before* the branch. The durable ledger got written at the *end* of the handler, after an early return on unrecognized brands. Two ledgers: the rate-limit ceiling saw every dollar and stopped the system correctly, while the dashboard I actually looked at saw none of them, both technically working, disagreeing by $528.

I built both of those counters. I looked at the wrong one for months and felt informed, which is the expensive kind of feeling. A dashboard understating the bill by $528 while the rate limiter quietly knew the real figure the whole time is not a monitoring system, it is a guy who never checks his other inbox. The unlogged path was the generic-query path with no brand resolution, exactly the traffic that was growing.

• • •

THE QUIET FAILURE

The loud failure is the 40,000-token trajectory: enormous, and everyone can see they are not reading it.

The quiet failure:

Your logs are complete on the success path and empty on the failure path, so your dashboard is a survivorship chart and you do not know it.

This is worse than no logging: no logging makes you humble, partial logging makes you confident. The airank spend dashboard was not blank. It showed a number, wrong by $528, looking exactly as authoritative as a right one.

Second quiet failure: **you log the artifact and call it the record.** The artifact only exists when things worked. Every retry bound and every failure-rate calculation needs the attempt count, and that count lives in a row you have to write on purpose, on the path where nothing else got written.

• • •

DO / DON'T

Do

- Write the ledger row before the work, not after. Attempt, artifact, keep-decision: three separate writes, in that order.
- Use the OpenTelemetry gen_ai conventions (`gen_ai.request`, `gen_ai.invoke_agent`, `gen_ai.execute_tool`) instead of your own field names. They landed as the GenAI semantic conventions on 14 May 2026 and got their own repo in OTel core v1.42.0 on 24 August 2026, and free dashboards beat a bespoke schema you maintain by yourself.
- Make every span carry trace ID, parent span ID, and session ID, and propagate parent-child spans across agent handoffs, so a planner-to-sub-agent failure appears in one trace a 3am query can filter.
- Log the error class as a field, not as text inside a message. `402` is greppable. "the proxy seems unhappy" is not.
- Reconcile your two spend sources on a cron: Redis counter versus durable ledger, diffed daily, alert on drift over 1%.

Don't

- Don't `continue` past a failure without writing a row. That single keyword cost 407 sessions and hours of wrong theories. Nine characters of tidiness, and I got pwned by my own code comment.
- Don't dump full trajectories as your primary log. Keep them if storage is cheap, but the readable one-line-per-decision stream is what you operate on.
- Don't trust APM green as agent health. HTTP 200 with a looping agent is the normal case, not the edge case.
- Don't let the spend counter and the spend ledger diverge in the code path. One incrementing before the branch and one after means two ledgers.
- Don't skip a record because the failure "produced nothing worth keeping." That it produced nothing is the finding.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 35 is where "done" becomes a measurement instead of a claim. This chapter is the infrastructure that makes those measurements survive the night: if the number is not in a queryable row with a timestamp and an error class, it is a memory, and memories lose arguments to confident theories. Ch. 15 covered the plan that diverges from execution, uncaught because the trajectory is unreadable while the plan is a clean markdown file. Fix the log and the plan stops winning by default. Ch. 48 takes the record forward into alerting, replay, and who gets woken up.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("not a novel, not silence"), argument, not citation.
Verified:

- OpenTelemetry Semantic Conventions for Generative AI standardize recording of GenAI operations with model names, token counts, and message content; the conventions moved to their own semantic-conventions-genai repository in OTel core v1.42.0 on 24 August 2026. OpenTelemetry Blog, 14 May 2026: https://opentelemetry.io/blog/2026/genai-observability/
- GenAI observability project within OpenTelemetry defining semantic conventions for AI agent observability, based on Google's AI agent whitepaper. OpenTelemetry Blog, 6 March 2025: https://opentelemetry.io/blog/2025/ai-agent-observability/
- Structured tracing with `gen_ai.request`, `gen_ai.invoke_agent`, `gen_ai.execute_tool` enabling both dashboards and detailed trace visualization. Sentry Blog, updated 1 September 2026: https://blog.sentry.io/ai-agent-observability-developers-guide-to-agent-monitoring/
- Agent audit trail Internet-Draft, versions 00 through 03, last activity August 2026, still a draft and not an adopted standard. IETF Datatracker, August 2026: https://datatracker.ietf.org/doc/draft-sharif-agent-audit-trail/
- Four pillars of agent observability: tool-call spans, reasoning spans, state transitions, memory operations. Braintrust, June 2026: https://www.braintrust.dev/articles/agent-observability-complete-guide-2026
- Minimum viable trace schema (span type, inputs, outputs, timing, errors/retries, trace/parent-span/session IDs, queryable by tool name and error class); APM's inability to show looping, wrong-tool calls or hallucination behind HTTP 200; multi-agent parent-child span propagation; native integrations for OpenAI Agents SDK, LangGraph, Mastra, Pydantic AI, LangChain, CrewAI with OTel as fallback. All Braintrust, June 2026, same URL.
- 407 sessions, no evidence, airank, 7 August 2026. `continue` skipping both artifact storage and ledger row; `402 Payment Required`. `~/Projects/airank/blog/2026-08-07-four-hundred-and-seven-sessions-no-evidence.md`
- The meter only counted what worked, airank, 21 August 2026. 35,522 unlogged calls, 6.3% of lifetime answers, ~$528 at $0.014866 average; Redis counter before the branch, durable ledger after

an early return. `~/Projects/airank/blog/2026-08-21-the-meter-only-counted-what-worked.md`

- Log line size, measured live 9 September 2026 with `wc -c` and `wc -l` in `~/Projects/airank/storage/logs/`: `laravel.log` 21,202,454 bytes / 128,365 lines = 165 bytes per line; `browser.log` 4,550,175 / 11,425 = 398. One host, stated as a measurement, not a benchmark.

What I could not verify:

- No vendor-neutral benchmark comparing OpenTelemetry versus JSON-LD versus custom schemas. The OTel recommendation rests on ecosystem support, not measured superiority.
- Log immutability and append-only semantics as tamper protection: no usable source found. Flag for Ch. 48.

“*Everything still looks up.*”

CHAPTER 48

Cron Died and Nobody Noticed

"In complex systems, malfunction and even total nonfunction may not be detectable for long periods, if ever."

JOHN GALL, SYSTEMANTICS: HOW SYSTEMS WORK AND ESPECIALLY HOW THEY FAIL (1975)

Three monitoring checks existed to catch the thing that took airank down on 7 August 2026. One was gated behind an environment variable nobody set in production. One was a method named `reapStale()` with zero callers, merged and reviewed and never once invoked. One computed the correct verdict, put it in an exit code, and then ended its shell line with `|| true`. All three were written by people who cared. All three were documented in the README. The README was the most confident document in the repo, and every word of it was about intentions.

The scheduler had been dead for two days and the dashboard was a wall of green. No service crashed. No alert fired. No 5xx, no page. Thirteen scheduled tasks (score computation, collection top-up, table pruning, cache warming) had simply stopped happening, and every monitor I owned was cheerfully reporting that nothing was wrong. A job that does not run does not throw. Somebody had commented out one cron line during a migration with a `#` and a note meaning "temporarily," and temporarily lasted 48 hours because the only symptom was data quietly getting older.

> ***Bottom line:*** *Everything still looks up. That's the outage. Uptime checks measure things that respond to being asked. Scheduled agent work does not respond to being asked. It is a pulse, not a service, and the only way to monitor a pulse is to notice when it stops, which requires a system whose default state is alarm and whose off switch is the heartbeat itself. If your agent's scheduled work has no dead-man's-switch, you do not have monitoring. You have decoration.*

• • •

WHEN IT BITES

- Your Ralph loop (Ch. 14) is cron-based. Cron stops firing. The loop was the whole product, and the process list shows nothing wrong because there is nothing to show.
- A queue worker dies. Horizon reports the supervisor as active while the underlying worker has been reaped. Jobs pile up in `jobs` and nobody looks at that table on a Tuesday.
- A deploy removes a dependency the scheduled job needed. The job dies at import, at 03:00, in a log nobody tails, and the first to find out is a customer three weeks later asking why the report stopped.

• • •

THE PATTERN

There are two kinds of failure and your monitoring only understands one of them.

The first is a **loud failure**: something ran, something threw, an exception got a stack trace and a row in an error table. Everybody instruments this one, because it hands you an artifact. Somebody

gets paged.

The second is an **absence**. Nothing ran. No exception, because there was no execution. No error rate, because there were no requests. Datashelter put it plainly in June 2026: "In many incidents, the first sign of failure is silence. A cron job stops running. A server is decommissioned. A backup agent crashes after an update."

The SD Times finding from August 2026 is the one to tape to the monitor: "Not one serious outage began with a bad model output. Every one began in the plumbing around the model." Scheduler config. A missing dependency.

The fix has a name older than any of this: the **dead-man's-switch**. Crontap's May 2026 framing is the cleanest one-liner: "A dead man's switch is the developer pattern for 'alert me when the silence is the problem.'" Instead of the job telling you it failed, the job tells you it lived. A successful run pings an external monitor. The monitor's default state is alarm.

The critical word is **external**. A heartbeat monitor on the same host, under the same scheduler as the thing it watches, dies with it and dies silently.

Tighten wrote up the queue-worker version in October 2022: "For those of you who have ever been unpleasantly surprised by a silent but dead queue worker…"

An arXiv study from June 2026 (2606.14589) documented 22 incidents over eight weeks in a production LLM agent system with 40+ scheduled jobs. One meta-pattern, a failure whose error signal never reaches a human in actionable form, showed up at least **28 times across those 22 incidents**.

This is the part where I wanted a vendor postmortem, so somebody other than me could be the idiot in this chapter. I could not find one. AWS, Google, and Stripe archives, searched for a single published incident whose only symptom was silence: zero. No provider publishes how many scheduled jobs quietly stop firing in a year. No regulator publishes a HIPAA or SOX finding that says the job stopped and nobody noticed for eleven days. Datadog, New Relic, Grafana, and Splunk all sell heartbeat monitoring and none publishes a median time to alert on a job that never started. That hole is this chapter's argument showing up as a research problem: a failure with no artifact generates no paperwork either. So take "most production systems run scheduled work with no heartbeat on it" as my field observation, not a count.

• • •

ONE WORKED EXAMPLE

airank, 7 August 2026. Not one alarm. Three.

The trigger was a database host that stopped accepting connections. Root cause: two settings individually harmless and jointly fatal, MariaDB's default `max_connect_errors=100` plus `skip_name_resolve` left OFF. On a LAN with no PTR records every connection attempts a reverse lookup, every lookup fails, and every failure counts toward that 100. Real bug, real fix, not what this chapter is about.

None of them fired.

Alarm one: staleness detection, gated behind an environment variable never set in production. The code ran the check, read the flag, found nothing, returned.

Alarm two: worker liveness. A method named `reapStale()` that did the right thing and had **zero callers**.

Alarm three: exit-code detection. Both container healthchecks computed the correct verdict and discarded it. From the blog post that day: "The `|| true` throws away the exit code carrying every verdict the command computes."

I wrote all three. Jeremy Schoemaker, who has been shipping software since before `cron` was somebody else's problem. My monitoring was Mahir's homepage: enormous confidence, giant friendly banner, absolutely nothing behind it. I kiss you!!! I alert you!!! Neither one was true.

The outcome: `skip_name_resolve=ON`, the staleness check moved into the collector's own healthcheck where it actually runs, and then the part that stung. Once the alarms worked they turned up a throughput problem sitting there the whole time: **60 valid observations against 1,463 phrases**, filtered out by a brand-recognition gate. A second outage nobody had ever seen.

Eight days later, 15 August 2026, the scheduler-dead-for-two-days incident. One commented-out cron line, thirteen dead tasks, zero alerts. The fix was a dead-man's-switch: the last entry in the schedule pings an external monitor every minute, three missed beats page a phone.

The line I wrote that night is the one I would put on the wall: **"The dead-man's-switch cost eleven lines. The outage it ends cost two days, and nobody can say which two days the next one would have taken."**

Getting pwned by a `#` is a special feeling.

And Ralph, since somebody always asks. My Ralph loop has never lost a night to a dead cron. It has lost time three other ways, each worse, because the process table kept insisting everything was fine.

On 7 September 2026 a Mac running detached collectors under `caffeinate -i -w <pid>` dropped into Low Power Sleep at one percent battery and stayed down almost four hours, until somebody plugged it in. Both PIDs survived the entire nap. `ps` was delighted. Not one data timestamp moved. Idle-sleep prevention does not prevent critical-battery hibernation.

Before that, a pruning loop launched as `ssh host 'nohup ./loop.sh &'` was gone hours later, no log line, no exit status. The symptom was never "the pruner stopped." It was `~/.omlx/cache` back at 254 GB from 196, and `/` at 100 percent with 116 MB free.

The inverse is funnier. Three identical 20-minute crons stacked up by repeated `/loop` invocations, so the loop fired three times a window and stepped on its own state. Nothing errored, because a duplicate schedule is a perfectly valid schedule.

• • •

THE QUIET FAILURE

Your monitoring counts events, and the failure produced no events, so the count is zero, and zero reads as healthy.

The cleanest demonstration I have is not even a cron story. **airank, 19 August 2026:** a load balancer node in one availability zone died and stopped accepting connections for roughly 14 hours. A connection that never completes never becomes a request, never enters `RequestCount`, and cannot appear in an error rate. Twenty consecutive checks reported 200s, zero 5xx, healthy targets, while a third of traffic got connection timeouts. Invisible by construction, not by oversight. (I later retracted a detail of that post about IP ownership, which is its own lesson about a guy publishing infrastructure claims at 2am on four hours of sleep.)

Second quiet failure: **the alarm exists, and its existence is treated as coverage.** A written, tested, documented, unwired alarm feels like a solved problem. It is the animated "Under Construction" gif of infrastructure: it tells you somebody had a plan for this spot in 1998 and then went to lunch.

Third: **the healthcheck measures the wrong noun.** A container reporting healthy proves the container is running. It does not prove it is doing work, or *correct* work. On 8 August 2026 I had a healthy container serving a rotated-away bearer token, and team creation silently skipping subscription because a config key was never defined. Every one reported success. The wipe that same day left zero observations at 18:43 and could not be traced, because binary logging was off. I recovered 577 of the 576 lost observations from artifact storage. The monitoring did not save me. A side effect of a storage decision did.

• • •

DO / DON'T

Do

- Put a dead-man's-switch on every scheduled job that matters. Last line of the schedule, ping an external monitor, three missed beats pages a human. "No successful run in 90 minutes" catches the whole class that never ran.
- Host the monitor somewhere that cannot die with the thing it watches. Different host, different network, ideally a different company.
- Audit for alarms that are not wired: grep for `|| true` (each a discarded verdict), for methods with zero callers like `reapStale()`, and for env-gated safety checks whose var is in `.env.example` but not **in production**.
- Measure the work, not the process. Rows written, jobs completed, observations collected. `60 against 1,463` would have screamed if anyone had been looking.

- Test your monitoring by breaking the thing on purpose. Stop the worker. Comment out the cron line. If nothing pages you in the window you promised, you learned it now instead of on day two.

Don't

- Don't trust a green dashboard as proof that scheduled work is running.
- Don't count "documented in the README" as wired. Three airank alarms were documented and none was connected to anything.
- Don't assume `docker restart` re-reads `.env`. It does not.
- Don't monitor only aggregates. A per-AZ, per-shard, per-worker breakdown catches the failure a third of your traffic is having while the average stays flat.
- Don't comment out a cron line "temporarily" without a dated reminder attached to a human. Temporarily is two days, and only if you are lucky.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 14 is the Ralph loop, scheduled with cron. This chapter is the tax on it, and the cheapest fix in the book (eleven lines) makes the loop's silence audible. Ch. 38 holds the general observability principle; this is the case where the instrument and the failure are built so they cannot meet. Ch. 49 is what happens after you see the failure. Ch. 15 is the sibling argument: a plan is not the work, and a written alarm is not a wired one.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("everything still looks up, that's the outage"): argument, not citation.
Verified:

- Cron and scheduled work fail silently; the first sign of failure is frequently silence itself. Datashelter, June 2026: https://datashelter.tech/blog/backup-monitoring-dead-mans-switch
- Dead-man's-switch as the standard pattern: scheduled job pings an external monitor, silence triggers the alert. Crontap, May 2026: https://crontap.com/blog/dead-man-switch-explained-for-developers
- Serious outages in scheduled agent systems begin in the plumbing, not in model output. SD Times, August 2026: https://sdtimes.com/agentic-ai/your-agents-arent-failing-theyre-not-running/
- Queue workers die silently while dashboards report healthy. Tighten Co., October 2022: https://tighten.com/insights/are-your-queue-workers-working/
- 40+ scheduled jobs, 22 incidents over eight weeks, one meta-pattern (failure signal never reaching a human) manifesting at least 28 times. arXiv, June 2026: https://arxiv.org/abs/2606.14589
- Three alarms, none wired: staleness check disabled by absent env var, `reapStale()` with zero callers, exit code discarded by `|| true`. MariaDB `max_connect_errors=100` plus `skip_name_resolve=OFF`. Post-fix: 60 observations against 1,463 phrases. airank, 7 August 2026: `~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`
- Scheduler dead two days, thirteen tasks stopped, no alerts; eleven-line dead-man's-switch. airank, 15 August 2026: `~/Projects/airank/blog/2026-08-15-the-night-the-loop-ran.md`
- Load balancer node dead ~14 hours in one AZ; twenty consecutive checks reported 200s while a third of traffic timed out. IP-ownership detail retracted same day. airank, 19 August 2026: `~/Projects/airank/blog/2026-08-19-the-outage-my-monitor-couldnt-count.md`
- Everything reported success: untraceable wipe at 18:43 with binary logging off, webhook signature failures, healthy container serving a rotated-away token, 577 of 576 observations recovered. airank, 8 August 2026: `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`

- Low Power Sleep at 1% battery for nearly four hours with PIDs alive and timestamps frozen (7 September 2026), and a `nohup` loop over SSH dying with its session, leaving `~/.omlx/cache` at 254 GB and `/` full with 116 MB free. `~/.claude/skills/background-jobs-die-silently/SKILL.md`
- Three identical 20-minute crons from repeated `/loop` invocations, firing 3x per window, no error. `~/.claude/skills/killed-session-wreckage/SKILL.md`, 24 August 2026.
- `#CUTOVER#` cron line, two days, thirteen tasks, heartbeat-inside-the-schedule fix. `~/.claude/skills/scheduler-dead-mans-switch/SKILL.md`, 15 August 2026.

What I could not verify:

- The five unsourceable items (vendor postmortem, platform frequency data, a regulator finding, a detection-latency benchmark, heartbeat-adoption survey data) are named in the text and no number is claimed for any of them.
- No record exists of Ralph itself dying to a dead cron. The three Ralph failures printed here are the documented ones, and all three are process-alive-work-stopped, not cron-stopped.

"*Unattended overnight work is not a model capability.*"

CHAPTER 49

It Works While You Sleep

"Early to bed and early to rise, makes a man healthy, wealthy and wise."

BENJAMIN FRANKLIN, POOR RICHARD'S ALMANACK (1735)

Eleven agents, two workflows, one MinIO bucket, one database. By morning the evidence said eight tests were failing on clean main, three merges had blown up, and a toolbar badge still needed building. Three alarms, all ugly, all sitting there waiting with the coffee. I read them in order and started fixing the first one. The tell was small enough that I walked past it twice: the assertion values were not the same from one run to the next. Real regressions do not do that.

At 2am on 13 August 2026 a review agent on airank ran a SELECT against production and found that an offload job I had already approved would have overwritten 337,353 rows of archive pointers. I was asleep. I did not approve, review, or know about that catch until coffee. The machinery that found it was not the smart part of the stack. It was a query.

> ***Bottom line:*** *Unattended overnight work is not a model capability. It is a verification capability. The agent is allowed to run while you sleep exactly as far as something cheap and deterministic can tell you in the morning whether it worked, and not one inch further. If the only artifact waiting for you at 7am is a report of what the agent would do, you did not run an overnight job. You ran a very expensive brainstorm and paid for eight hours of GPU time to have it read nicely.*

The thing that has to run all night is the check, not the model.

• • •

WHEN IT BITES

- You wake up to a green summary saying "all four implementations complete" with no test output, no diff count, no query result. The model grading its own homework.
- Eleven agents run in parallel against one MinIO bucket. Eight tests fail. You spend the morning debugging a regression that does not exist.
- The scheduler dies because logrotate rotated its output file and cron cannot write to a root-owned directory. Nothing errors, nothing pages, nothing is reporting anything at all.
- A shell flag (`noclobber`, in my case) refuses to overwrite a log file, the merge commands never run, and the failure gets reported to you as a git conflict. Jeremy Schoemaker, twenty-five years of living in a shell, pwned by his own dotfiles.

• • •

THE PATTERN

Among the longest-running Claude Code sessions, the 99.9th-percentile time the agent works before stopping nearly doubled in three months, from under 25 minutes to over 45 minutes (Anthropic, February 2026). Same paper: new users run full auto-approve in roughly 20% of sessions, experienced users in over 40%.

Forty-five minutes is not a night. The gap between "45 minutes at p99.9" and "burned down ten jobs by morning" is filled by structure, not a better model. Ralph Loop, which shipped in Anthropic's official Claude Code plugin marketplace as `ralph-wiggum`, is the crude version: an autonomous loop that runs for hours unattended, marketed for "night batch jobs that need to be finished in the morning" (Paddo.dev, December 2025; Obvious Works, March 2026). Finished in the morning. That's what she said.

The loop is the easy half. The hard half is the sentence Zernie put better than I have managed to: an agent can run nearly unattended wherever you can check its output cheaply, deterministically, and a thousand times a night, and wherever you cannot, you are the bottleneck and a smarter model changes nothing about that (Zernie, July 2026).

So the pattern is four parts, and skipping any one of them turns your night into a report:

1. The goal has a shape, not a mood. Chapter 8's ladder: PRD, milestones, phases, tasks, tests, prompts, build. By the time an overnight job starts, "every task passes its test" has to mean a specific command with a specific exit code. Anything I cannot phrase as a command is a daytime job.

2. The check is deterministic and it runs without me. In commander-in-chief, the Godot project I have been building since July 2023, the simulation core in `src/sim/` uses fixed 16.16 integer math and a `SimRng` xoshiro128** generator. A linter (`tools/lint_sim.gd`) fails the build if anyone sneaks in a float, an engine RNG call, a `Time.*` call, or scene-tree access. `test_determinism.gd` checks FNV-1a golden checksums across runs, producing bit-identical state on x86_64 and Apple Silicon. The formal name is deterministic simulation testing (DBOS, July 2025; Zernie, July 2026).

3. Something external holds the state. Long-running agents have to solve persistence, recovery, and verification with a state layer that lives outside the model's context window (Addy Osmani, Elevate, April 2026).

4. Remediation is bounded, investigation is not. Autonomous investigation is reliable; autonomous remediation stays bounded and supervised (DevSecOps.ae, August 2026). My night jobs get free rein to read production, run tests, count artifacts, and write code. They do not get an unbounded `DROP`, an unreviewed schema migration, or a deploy to a customer-facing box without a gate.

• • •

ONE WORKED EXAMPLE

airank, the night of 13 August 2026. Two overnight runs.

The first was a council marathon: four implementations carried to completion before morning. Database optimization, toolbar readiness, webserver tuning, centralized logging. All four landed. Most were saved from their own authors by background machinery that had nothing to do with intelligence.

Save one is the 337,353 rows from the top of this chapter: a query, not a code review.

Save two: the nginx microcache config failed curl tests twice, once on a hyphenated cookie name, once because it stripped `Set-Cookie` unconditionally, which `nginx -t` never flags and which breaks every login on the site. Fixed, it took the site from 16.6 RPS at a p50 of 1.13 seconds to 1,873 RPS at a p50 of 10.3ms, a 113x speedup that would have shipped attached to a total login outage.

Same night, second run. Eleven unattended jobs across two workflows, an overnight survey and a todo-cycle. Three times the evidence claimed something was broken. Three times the evidence was lying.

Eight tests "failing on clean main." Eleven agents were running parallel suites against one shared MinIO bucket and one shared database, and the fixtures were colliding. I set that up myself, and it is *Snakes on a Plane* (2006): you know how it ends from the title, and I booked the flight anyway. Re-run solo on a quiet fleet, 17 of 17 pass.

"Three merge failures." zsh's `noclobber` flag refused to overwrite a log file, so the merge commands never executed. `merge=1` was a shell error wearing a git conflict's clothes. The kill was counting: `git log --oneline` showed one merge where four should be. Re-run with `>|`, four clean merges.

Eleven jobs landed, three false alarms killed. Both nights I contributed nothing but the goal file and my continued unconsciousness.

Both nights went in the win column, which is the part I do not trust. Every overnight failure I have a record of got caught by a query, a curl, a test, or the morning log sweep, and I cannot show

you the one that got past all four. That is not proof it never happened. A file drawer full of caught failures is exactly what a system with an uncaught one looks like, because the miss does not file a report. Read these two nights as the shape of the thing, not as a batting average.

• • •

THE QUIET FAILURE

The loud failure is the overnight run that crashes at 1am and leaves you a stack trace. Annoying, cheap, obvious.

The quiet failure:

The agent finishes, declares success, and the proof is a paragraph it wrote about itself.

"Implemented centralized logging. All changes verified." No exit code, no row count, no curl output, no `git log --oneline` you can read in nine seconds. This is the Ch. 20 seam again: when done is not a measurement, the most articulate artifact in the room wins by default.

Second: **your verification shares infrastructure with your work.** Eleven agents writing to one bucket cost me a morning of fake test failures. A non-deterministic check is a coin flip with a log file.

Third: **nothing watches the watcher.** The scheduler died at logrotate time and no alarm fired, because the alarm was downstream of the scheduler. That is a smoke detector mounted inside the fire, and a dead cron produces silence that looks like a quiet night.

During the migration two days later, the scheduler's cron line got commented out with a marker meaning *temporarily*, and temporarily ran from 13 to 15 August 2026. Thirteen scheduled tasks stopped. Every dashboard stayed green the whole time, because a scheduler is not a service, it is a pulse, and nothing was measuring the pulse.

The fix shipped on 15 August and it is eleven lines at the bottom of `routes/console.php`: the schedule's own last entry pings an Uptime Kuma push monitor every minute, and the monitor, `airank-scheduler-alive`, has a 180-second window, so three missed beats page a phone. It reads the URL through `config()` and not `env()`, because `env()` returns null under `config:cache` and I would have built a dead man's switch that was itself dead. Eleven lines buy back the two days I did not know I had lost.

• • •

DO / DON'T

Do

- Write the goal so that every task names its own passing command. I never saved the block from 13 August and the worktrees got cleaned up, so what follows is the shape from memory, not a transcript: `/goal`, plus a workflow, plus "every task passes its test," plus "work autonomously, I have to sleep." The fourth clause is only safe because of the third.
- Make the checks deterministic. Fixed-point math, seeded RNG, no wall-clock reads, no shared network state. Bit-identical output across machines is what makes a 3am failure believable.
- Isolate every parallel agent's fixtures: separate buckets, databases, worktrees. Shared infrastructure turns eleven honest suites into one liar.
- Leave a proof, not a report. Row counts, exit codes, curl output, `git log --oneline`, a diff stat. Something a person can check in under a minute.
- Put a dead-man's switch on the scheduler itself, and page from somewhere the cron cannot kill. Then re-verify the premise before implementing: a four-hour-old survey is fiction after eight deploys.

Don't

- Don't run unattended where the check is a human reading output. Zernie's line is the boundary.
- Don't accept "task complete" as an artifact. It is a sentence.
- Don't let a night job do unbounded remediation. Schema changes, destructive writes, and customer-facing deploys get a gate and a morning human.

- Don't trust a failure any harder than you trust a success. Three of my eleven jobs reported breakage that did not exist.
- Don't debug the reported cause. `merge=1` was zsh, not git.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 built the ladder that makes a `/goal` block executable. Ch. 14 is the Ralph loop, and this chapter is that loop with the lights off and no one to ask. Ch. 15 is why the plan is not the work, the same argument moved eight hours later. Ch. 36 is the verification layer this chapter leans on entirely: no cheap deterministic check, no unattended night. Ch. 60 is where this scales past one machine and one person's sleep schedule.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's: argument, not citation.
Verified:

- Claude Code p99.9 autonomy nearly doubled in three months (under 25 minutes to over 45); full auto-approve in ~20% of new-user sessions, over 40% for experienced. Anthropic Research, February 2026: https://www.anthropic.com/research/measuring-agent-autonomy
- Ralph Loop (`ralph-wiggum`) in Claude Code's official plugin marketplace. Paddo.dev, December 2025: https://paddo.dev/blog/ralph-wiggum-autonomous-loops/
- Ralph Loop for "night batch jobs that need to be finished in the morning." Obvious Works, March 2026: https://www.obviousworks.ch/en/agentic-coding-tools-2026-the-7-frameworks-that-take-your-development-to-a-new-level/
- Verification bounds unattended operation. Zernie, July 2026: https://zernie.com/blog/verification-is-all-you-need/
- Deterministic simulation testing. DBOS, July 2025: https://www.dbos.dev/blog/how-to-test-durable-execution
- Agentic SRE: reliable investigation, bounded remediation. DevSecOps.ae, August 2026: https://devsecops.ae/agentic-sre-governance-autonomous-remediation-2026/
- State layer outside the context window. Addy Osmani, Elevate, April 2026: https://addyo.substack.com/p/long-running-agents
- Worked example 1, airank, 13 August 2026: four implementations completed; 337,353 archive-pointer rows saved by a production query; cookie-name and `Set-Cookie` bugs caught by curl; nginx microcache 16.6 RPS / p50 1.13s to 1,873 RPS / p50 10.3ms. `~/Projects/airank/blog/2026-08-13-three-saves-in-one-night.md`
- Worked example 2, airank, 13 August 2026: eleven unattended jobs; fixture collision on shared MinIO and database (17/17 solo); zsh `noclobber` masquerading as git merge conflicts. `~/Projects/airank/blog/2026-08-13-three-liars-in-one-night.md`
- Worked example 3: deterministic sim core, commander-in-chief, July 2023 to present. Fixed 16.16 math, `SimRng` xoshiro128**, `tools/lint_sim.gd`, `test_determinism.gd` FNV-1a checksums, bit-identical across x86_64 and Apple Silicon. `~/Projects/commander-in-chief/CONTRIBUTING.md`
- Scheduler death by logrotate into a root-owned `/var/log`, caught by the morning log sweep; already-shipped toolbar badge requested by a four-hour-stale survey, closed as a verified no-op. airank, 13 August 2026: `~/Projects/airank/blog/2026-08-13-three-saves-in-one-night.md`
- Second scheduler death: cron line commented out during the migration with a "temporarily" marker, 13 scheduled tasks dead 13 to 15 August 2026 with every dashboard green. airank, 15 August 2026: `~/Projects/airank/blog/2026-08-15-the-night-the-loop-ran.md`
- Dead-man's switch, shipped 15 August 2026: eleven lines in `routes/console.php` pinging an Uptime Kuma push monitor every minute, monitor `airank-scheduler-alive` with a 180-second

window, URL read via `config('air.scheduler_kuma_push_url')` because `env()` is null under `config:cache`. airank source: `~/Projects/airank/routes/console.php` and `~/Projects/airank/config/air.php`

What I could not verify:

- No documented overnight run that shipped something broken and got past the morning check. Searched `~/Projects/airank/blog/`, 10 to 19 August 2026: every incident on record was caught by a query, a curl, a test, or the log sweep. The body says outright that an archive of caught failures is not evidence there were no uncaught ones.
- The verbatim `/goal` text from the 13 August 2026 runs is not preserved in `~/Projects/airank/.claude/worktrees/` or `.opencode/`. The Do section gives the shape and labels it memory.
- No vendor post-mortems on overnight agent failures in production; no unattended-versus-supervised failure-rate comparison; no metric for how often "task complete" is false.
- No at-scale case study of deterministic simulation testing catching an overnight failure; Zernie is early, DBOS is guidance.
- Gartner's "15% of day-to-day decisions autonomous by 2028" is a forecast, not cited in the body. The July 2026 Hugging Face breach was an attack, not a batch job; excluded.

"Prioritization is impact divided by effort, and almost everybody gets it wrong on the numerator, before ranking ever starts."

CHAPTER 50

What First. What Never.

"I love deadlines. I love the whooshing noise they make as they go by."

DOUGLAS ADAMS, THE SALMON OF DOUBT (2002)

Six hours, three planners, one question: how do we collect a thousand a day. Three architectures came back, each internally coherent, each priced with a straight face, from $0 a month to $5,000 a month. I read all three. I had a favorite. We were forty minutes from picking one when somebody went and looked at what the collector was actually doing on screen, and the thing on screen was not a throughput problem, was not an architecture problem, and did not care about any of the three documents. It cost about twenty dollars to fix. I will get to what it was.

> ***Bottom line:*** *Prioritization is impact divided by effort, and almost everybody gets it wrong on the numerator, before ranking ever starts. You rank a list of guesses with great discipline, and the ordering is perfect and the list is fiction. Re-measure the impact number the same day you rank it. Then do the harder half: kill the work that has no customer, including the work you already paid for. A ranked backlog is not a plan, it is a receipt for what you believed on a Tuesday.*

• • •

WHEN IT BITES

- Your agent produces a beautifully ranked backlog of 22 items and every impact estimate in it is a model's opinion about a number it never queried.
- Someone ships a feature nobody asked for, on time, under budget, and it goes in the deck as a win. It is now permanent maintenance cost with no user.
- You page five times in a night about slow queries, and the reflex is "add indexes," a solution wearing the costume of a diagnosis.
- The team measures deployment frequency because it's easy to instrument, and nobody asks what those deploys delivered (Harness, March 2026).
- The thing at the top of your list is 1.8% of the problem (Ch. 15). You already know how that story ends.

• • •

THE PATTERN

Impact over effort is the right formula. The failure is not arithmetic, it is sourcing. Effort you can estimate, badly but honestly, because effort is about your own team and you have scar tissue. Impact is about the world, and the world doesn't appear in your planning session unless you go get it.

So the first rule is boring: **re-measure the impact number before you rank, not after.** Read counters, row counts, kill counts, whatever the domain's equivalent of `SELECT COUNT(*)` is. The cost of the measurement is usually under two minutes, and the cost of skipping it is the entire ordering.

The second rule is the one people hate: **kill work that has no customer.** An unused feature is not neutral, it is a recurring bill: support cost, test surface, and workflow clutter that distracts users from the tasks they actually came for (Airfocus, February 2026).

Pareto is the reason both rules pay. Roughly 80% of outcomes come from 20% of causes, and in software specifically, about 80% of crashes trace to 20% of reported bugs (Asana, February 2026). If the distribution is that lopsided, getting the ranking approximately right beats executing the middle of the list perfectly, and the 20% is not the part that feels urgent. It is the part with the highest counter, and counters do not lobby.

Third, and this is where agents make it worse: an agent produces a confident theory for anything you point it at, in seconds, formatted. It is Clippy with a PhD. It looks like you're writing a prioritization framework, and it would love to help, and it has never once queried your database. Coherence is free now; evidence is not. The scarce resource is the willingness to spend ninety seconds killing analysis with a counter.

The fourth: deletion deserves the same machinery as construction, meaning user research, a plan, in-app guidance, roadmapping, communication (Airfocus, February 2026), and a senior PM at Adobe/Workfront has removed three entire areas of functionality exactly that way. Most teams treat shipping as a project and killing as a chore, which is backwards.

• • •

ONE WORKED EXAMPLE

airank, 14 August 2026. Five pages in one night. Slow-query storm. The reflex was immediate and unanimous: add indexes.

Two indexes were nominated for deletion as dead weight, on the strength of a leading-column selectivity heuristic, real expertise that is also a proxy. Proxies do not know what your traffic did last month.

Then somebody ran the query that ends arguments: `performance_schema.table_io_waits_summary_by_index_usage`. Actual read counts, per index, from the engine itself.

`cgos_via_search_idx`: **0 reads.** Nobody had nominated it.

`cgos_ad_network_idx`: **298,255 reads.** Nominated for deletion.

Measurement didn't refine the answer, it changed the sign of it. The council retracted every load-bearing claim in front of the read counter, the only correct response and rarer than it should be.

Jeremy Schoemaker sat in that room and agreed that `cgos_ad_network_idx` was dead weight. My own database pwned me with one row of output. The plan was beautiful, though. Ranked, dependency-ordered, color-coded. A ranked backlog with that kind of formatting is a Winamp skin: it really whips the llama's ass, and it is not playing anything.

The outcome: delete `cgos_via_search_idx`, keep `cgos_ad_network_idx`, add one missing index on `product_categories.parent_id`. The paging stopped. Nothing in the original ranked plan survived contact with the data.

The reversal cost, since I promised it. What killed all three architectures on 8 August was a ChatGPT rate-limit modal, sitting in the archived captures the whole time, misclassified by the collector and never once opened by a human. The fix was about twenty-five lines of code for roughly twenty dollars: against the cheapest architecture anybody actually wanted to build, a factor of 200 to 250. The evidence was in the archive before the first planner started typing. Nobody had double-clicked it, and I was the one with a favorite.

The day after, 9 August 2026, two `minio_key` queries took ninety seconds and found 18,567 answers that had never been archived, about $240 of work I had already declared done. Ninety seconds of counter, again, against a theory nobody had checked.

commander-in-chief, v1.2.0, 24 August 2026, same method, different domain: a top-down combat game with a simulation harness, so "ignitions per pre-flight" is a number you can actually run a thousand trials against. Fifty-one owner decisions went into measurement-based review. Twenty-eight came out prioritized, four were defects wearing a decision costume, and twenty-three were killed or reclassified. That is 45% of the list deleted by counters, and every one of those 23 had a human's reasoning behind it, mine included.

Decision D14 was mine to defend: riot-shield flanking, theorized as a consequence of the shield's instant pivot. The measurement, at standoff range (40px) across 16 strafe bearings, **0 kills.** At contact range, **46 kills across 1,512 trials.** The flanking verb does not exist in the game. The recommendation stopped being "tune flanking" and became "give the riot shield a turn rate." D1, the tank bail mechanic, died the same way: 16 of 22 ignitions per pre-flight were ordnance hits and a vestless rider dies on the same tick, so the bail window the UI promises is never reachable.

So the reversal cost is not a rounding error you absorb. It is a working day per planning session and roughly half the backlog. Which sounds expensive until you price the other branch: shipping the $5,000-a-month architecture and paying it monthly, forever, for a modal.

• • •

THE QUIET FAILURE

The loud failure is building the wrong thing badly. It gets caught.

The quiet failure: **you build the wrong thing well, on schedule, and everyone congratulates you.**

Nothing in the process catches this. Tests pass, the deploy is clean, velocity is up. The feature has no customer, and every quarter forward it collects support tickets and complicates a workflow for people who came to do something else. That cost never gets attributed back to the decision that created it: by then the decision is eighteen months old and the person who made it got promoted for shipping.

Second quiet failure: **the ranked list becomes the belief.** Somebody writes an impact estimate in a cell, the list gets sorted, and by Thursday that estimate is a fact in three documents and a standup. Same disease as Ch. 15: a conclusion with a shelf life, formatted identically to a fact.

Third: **you measure what is easy to instrument.** Most engineering productivity programs fail at the measurement selection stage, tracking what the tooling emits rather than what moves a strategic outcome (Harness, March 2026). An agent will optimize whatever number you hand it, forever, with excellent commit messages.

Fourth: **deprioritized is not killed.** Work sitting in a backlog is a liability that reads as an asset. Delete the row.

• • •

DO / DON'T

Do

- Re-measure the impact number the day you rank, from the system, not from memory.
- Prefer a counter to a heuristic when the two disagree. Leading-column selectivity is real expertise and it still got `cgos_ad_network_idx` backwards at 298,255 reads.
- Kill work with no customer, and kill it properly: research, plan, in-app guidance, communication, the same as you'd ship a feature (Airfocus, February 2026).
- Give every rule in your prioritization doc the incident that bought it.
- Let measurement change the sign of the answer, not just the magnitude.
- Find the 20% that carries 80% of the crashes (Asana, February 2026) before polishing the middle of the list.

Don't

- Don't rank estimates. Ranking adds no information to bad inputs, it only makes them look adjudicated.
- Don't track a metric because the tooling emits it (Harness, March 2026).
- Don't treat six hours of planning as sunk value earning the plan a hearing.
- Don't leave killed work in the backlog as "later." Later is a promise your next plan silently prices in.
- Don't skip the ninety-second verification because the fix is obviously done. Obviously-done cost 18,567 records and $240.

- Don't ship a capability your interface promises and your telemetry says is unreachable (the tank bail window: 16 of 22 ignitions were ordnance, and the rider was already dead).

• • •

WHERE THIS SITS IN THE BOOK

Ch. 15 established that a plan is a bet and every priority rests on a number you should re-measure before acting, which is where the 78.1% versus 1.8% split came from. This chapter is the ordering problem downstream of that: once you can re-measure, what do you do first and what do you refuse to do at all. Ch. 20 is the other half, since if "done" isn't a measurement then nothing in your ranking can be closed. Ch. 58 takes the killing discipline into the organization, where work with no customer has a headcount attached and the deletion gets political.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (re-measure before ranking; kill work with no customer): argument, not citation.

Verified:

- Most engineering productivity programs fail at the measurement selection stage, tracking what is easy to instrument rather than what influences strategic outcomes; deployment frequency without context is a vanity metric. Harness, March 2026: https://www.harness.io/blog/measuring-developer-productivity-prove-impact
- Pareto principle: 80% of outcomes from 20% of causes; in software, 80% of crashes from 20% of reported bugs. Asana, February 2026: https://asana.com/resources/pareto-principle-80-20-rule
- Unused features cost money to support and complicate user workflows; deprecation should get the same treatment as development (user research, planning, in-app guidance, roadmapping, communication); a senior PM at Adobe/Workfront (Vazgen Babayan) has removed three areas of functionality this way and is working on a fourth. Airfocus, February 2026: https://airfocus.com/blog/how-to-kill-product-features-without-losing-customers/
- Worked example 1: airank, 14 August 2026. `cgos_via_search_idx` 0 reads (deleted), `cgos_ad_network_idx` 298,255 reads (retained after nomination for deletion), one index added on `product_categories.parent_id`. `~/Projects/airank/blog/2026-08-14-the-optimization-was-a-deletion.md`
- Worked example 2: airank, 8 August 2026. Six hours, three planners, architectures $0 to $5,000/month, all wrong; cause was a misclassified ChatGPT rate-limit modal; fix ~25 lines, ~$20; produced a 20-plus rule constitution. `~/Projects/airank/blog/2026-08-08-ten-articles-we-did-not-set-out-to-write.md`
- Worked example 3: airank, 9 August 2026. Two `minio_key` queries, ninety seconds, resolved a prior cost of 18,567 unarchived answers (~$240). `~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md`
- Worked example 4: commander-in-chief v1.2.0, 24 August 2026. 51 decisions to 28 prioritized plus 4 misfiled defects, 23 killed or reclassified (45%). D14: 0 kills at 40px across 16 strafe bearings, 46 kills / 1,512 trials at contact. D1: 16/22 ignitions per pre-flight were ordnance hits. `~/Projects/commander-in-chief/DECISIONS.md`

What I could not verify:

- No peer-reviewed sources found for impact/effort prioritization frameworks. MoSCoW, RICE and Kano are named in the Airfocus piece with no underlying study to cite. Kept out of the chapter body.
- No quantified case studies located for lifetime cost of never-used features at named companies. The Airfocus claim is directional, not measured.

- No research located on the organizational cost of building the wrong thing well versus the right thing badly. "The quiet failure" is field-observed and stated as such.
- Google's moonshot-killing practice (2016-2020) is referenced in the chapter direction but no sources were located. Deliberately not cited.
- Ch. 15's 78.1% versus 1.8% figures are internal to airank, not independently verifiable outside the project blog. Referenced here, not re-argued.
- No external research located on decision-reversal cost as a general figure. The two priced reversals here (6 hours on 8 August 2026; 23 of 51 decisions on 24 August 2026) are Jeremy's own projects, not an industry baseline.

CHAPTER 51

Don't Text the Wrong Person

"There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery"

MATHIAS VERRAES, TWO HARD PROBLEMS WITH DISTRIBUTED SYSTEMS, VERRAES.NET (2015)

The send tool times out at thirty seconds. The SMS actually left at twenty-eight. The retry fires, and at 2:41am a customer's phone buzzes twice with the same message out of a system I wrote, while the dashboard stays green, because both requests succeeded. That is the version I know personally. The version that made the news started with a caching upgrade on a Monday and ended nine hours later with strangers reading each other's billing addresses. My home agent has never done either: it keeps a session per person and knows exactly which human is which, and Georgia has decided that one of those sessions is hers, sleeping by the Mac Studio until dinner shows up on schedule.

ChatGPT Plus subscribers clicking "Manage Subscription" were looking at somebody else's first and last name, billing address, credit card type, expiration date, and last four. That is OpenAI's own disclosure of what happened on March 20, 2023, and the number they published was 1.2% of Plus subscribers. Somebody swapped a Redis client and the sessions stopped being sessions.

> ***Bottom line:*** *The instant your agent serves more than one human, the interesting bug stops being "did it answer well" and becomes "did it answer the right person, once." Two things carry identity in a multi-user agent: the memory it reads and the send it performs. Both default to wrong. Memory defaults to a shared pool because that is what a cache is, and sends default to at-least-once because that is what a retry is. You get cross-user leaks on one side and "I already sent that" on the other, and neither shows up in your eval suite, because your eval suite has one user in it.*

• • •

WHEN IT BITES

- Your agent has a shared vector index, and the top-k retrieval for user B pulls a chunk written by user A because it scored 0.91 on cosine similarity.
- A caching layer gets swapped and request-to-user binding is now a race. (See: March 20, 2023.)
- A "share" button generates a URL you did not think was crawlable, and Google indexes 300,000 conversations. (See: Grok, August 2025.)
- Someone deploys your agent stack on a box with the inference server bound to 0.0.0.0 and no auth, and now the internet reads prompts. UpGuard found 117 of those actively leaking on August 27, 2025.
- Your prompt template interpolates `{user_name}` from a variable that a previous turn's tool result overwrote, and the agent opens with the wrong name, committing to it.

• • •

THE PATTERN

Every one of these is the same shape wearing different clothes: something that should be scoped to one person ends up scoped to the process, the request pool, the index, or the filesystem. Nothing complains, because widening scope is not an error condition, it is a performance optimization.

Three places identity leaks:

All three were me. Jeremy Schoemaker shipped a shared vector index with a `user_id` column and called it isolation, then a module-level current-user helper, then a retry path with no dedupe key. Twenty-five years of shipping software and I reinvented sending an AIM message to the wrong window (1999), except mine did it at machine speed and billed me for the tokens.

1. Memory. Ch. 21 covered what memory is for; here is what it costs. A shared store with a `user_id` column is not isolation, it is a filter you have to remember to apply on every read path, forever, including the one added next Thursday at 11pm. What survives is a key prefix you cannot query across: a namespace per user, an index per user, or a query builder where user-scoped is the only constructor and the unscoped one does not exist. Make the wrong thing unwriteable, not discouraged.

2. Request context. The user id needs to travel with the request, not live beside it. Globals, module-level singletons, thread-locals in an async runtime, and "current user" helpers are all the same bug waiting for enough concurrency. The 2023 Redis incident is the canonical version: an async client where a canceled request left a response in the connection pool and the next requester got it. It only comes apart when enough people hit it at once, which, yes, that's what she said.

3. Egress. This is the one that gets you sued rather than embarrassed. Sending is not idempotent, HTTP timeouts do not tell you whether it happened, and LLMs retry enthusiastically. Every send needs a key computed deterministically from content plus recipient plus intent, checked and written in the same transaction as the send. If the key exists, do nothing and report success.

The fourth place: the model itself is not a boundary. Anything in the context window is fair game for the next token. Put user A's data in the prompt so the agent can "compare accounts" and you have already leaked; no instruction un-leaks it. Nothing pops up to say "you're about to show user B user A's card." The last software that volunteered that kind of thing was Clippy (Office 97), and we killed him for it, and now we get 1.2% of Plus subscribers instead.

Multi-user failure is asymmetric: user A usually never finds out, user B sees the leak. Discovery is a customer emailing you a screenshot, a detection system with a lag measured in weeks.

• • •

ONE WORKED EXAMPLE

Grok, August 21, 2025. The share feature generated public URLs for conversations. Users pressed share thinking they were handing a link to one friend. Google's crawler thought otherwise. BBC News reported a search that day finding nearly 300,000 Grok conversations indexed: medical questions, drug interactions, the things people type when they believe nobody is watching.

Outcome: the conversations stayed indexed. The design failure was not novel; OpenAI and Meta shipped share features with the same crawlable-by-default property first. Three companies with more security engineers than my fleet has cores each independently decided that "share" meant "publish." Napster (1999) taught a generation that a file with a public URL belongs to everybody, and we are still shipping the button that forgets it. I have shipped that button too, with a link I told exactly one person about and a robots.txt I never wrote.

The uglier version came six days later. On August 27, 2025 UpGuard published an investigation into role-play chatbots on misconfigured llama.cpp deployments: roughly 400 exposed systems, 117 actively streaming user prompts to the open web with no authentication. Among the exposed content, researchers found scenarios describing the sexual abuse of children. No breach, no exploit, no attacker. Someone bound a port and shipped.

The 2:41am double text at the top of this chapter was mine, and it comes with a caveat. I went looking and cannot produce the product, the date, the head count, or the fix commit. The retry path had no dedupe key, a message that had already left at twenty-eight seconds went out again at thirty, and a handful of phones buzzed twice in the middle of the night. At-least-once delivery is a fine property for a queue and a miserable one for a text: the thing finished, then finished again,

unasked, while the dashboard sat there green and proud of itself. That is a memory, not a receipt, and I am not dressing it up as one.

What I can put a path and a date on is worse in a different direction. My messaging bridge notes, `~/.claude/skills/agent-messaging-bridge/SKILL.md`, 3 August 2026, exist because an agent with no relay tool was asked to pass a message to a third party and answered **"Sent! [ok]"** to a real human, who then waited for a message that was never addressed to anybody. Not the wrong recipient. No recipient, reported as a completed delivery, in a transcript that reads like success.

Same file: my daemon relayed a note into a thread without telling the agent. The user answered the relay, and the agent, holding a transcript with that page torn out of it, said **"Tell who? (?)"** and got filed as a model that cannot hold a conversation.

On the wrong number specifically I have a near-miss, labeled as one. Amber's production sender is `+19403012214`. A retired sandbox sender, `+17374294452`, is still sitting on that account, and the standing rule in `~/.claude/skills/amber-hermes-ops/SKILL.md` runs one line: never point config at it. Every text she has sent left from the right number because of a line of documentation and nobody editing a config at midnight.

• • •

THE QUIET FAILURE

The loud failure is the leak that makes the BBC, because a stranger's credit card ends up on someone's screen.

Isolation degrades gradually and passes every test you have, because your tests have one user.

You wrote the retrieval test with a fixture named `test_user`. It passes, and it will pass forever: the failure needs a second user with overlapping content, concurrent load, and a shared cache, none of which exist in the fixture. Ch. 30 applies here: a test that cannot fail is not evidence, it is a green square.

Second quiet failure: **the duplicate send nobody reports.** A customer who gets the same text twice mostly does not file a ticket, they downgrade their opinion of you by a notch. Instrument it instead: count sends by dedupe key, alert on any key above one. If that counter has never fired, verify it works.

Third: **the attachment problem makes leaks worse than the data implies.** Public Citizen's January 27, 2026 report covers the GPT-4o to GPT-5 swap in August 2025 and Replika's February 2023 rollback of intimate features: users reacted the way people react to losing a partner, and Harvard Business School researchers agreed. Italy fined Replika's developer 5 million euros in May 2025 [unverified]. When your agent is the thing someone confides in, the session boundary is the whole product promise. Break it once and you are handling a betrayal, not a bugfix.

• • •

DO / DON'T

Do

- Key every store by user at the namespace level: separate index, prefix, or collection, not a WHERE clause. The unscoped query should not be expressible.
- Pass the user id explicitly through every call frame. No globals, no thread-locals, no "current user" helper.
- Give every outbound action a deterministic idempotency key (recipient + content hash + intent + time bucket), written in the same transaction as the send. Existing key: return success, do nothing.
- Write the canary test: user A stores a unique string, user B queries every read path, assert zero. Put it in CI.
- Treat share links as publication. Set noindex, set expiry, and make the copy say what the button does.
- Audit the context window before every model call: another user's data there means isolation failed, and the prompt cannot save you.

- Bind local inference servers to localhost and put auth in front. UpGuard found 117 that skipped this in one scan.

Don't

- Don't let a timeout imply the action did not happen. Timeouts are silence, not evidence.
- Don't call single-user tests isolation coverage. They pass by construction.
- Don't measure leaks by ticket volume. Cross-user exposure is invisible to the person who was exposed.
- Don't ship "memory" before you have a per-user deletion path. Building it under deadline is how you delete the wrong rows.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 21 covered memory. This chapter is the tax on it: the second user arrives and memory becomes the leak surface. Ch. 30's evaluation problem is the same bug in different clothes. Ch. 52 covers what happens when isolated sessions have to coordinate anyway.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (isolation is an engineering requirement, and the model is not a boundary). Argument, not citation.

Verified:

- ChatGPT Redis incident, nine-hour window on March 20, 2023: another user could see name, billing address, card type, expiration date, and last four. Help Net Security, March 27, 2023: https://www.helpnetsecurity.com/2023/03/27/chatgpt-data-leak/
- Scope: payment information of 1.2% of ChatGPT Plus subscribers, March 20, 2023, 1-10am PT. The Hacker News, March 25, 2023: https://thehackernews.com/2023/03/openai-reveals-redis-bug-behind-chatgpt.html
- Grok conversations indexed by Google without users' knowledge; a search on August 21, 2025 found nearly 300,000 (reporting range 300,000-370,000). BBC News, August 21, 2025: https://www.bbc.com/news/articles/cdrkmk00jy0o
- Misconfigured llama.cpp role-play chatbots: ~400 exposed systems, 117 actively leaking user prompts to the open web, including scenarios describing child sexual abuse. UpGuard, August 27, 2025: https://www.upguard.com/news/security-flaw-in-ai-chatbots-exposes-explicit-user-fantasies
- GPT-4o to GPT-5 replacement in August 2025 produced strong backlash from emotionally attached users. Public Citizen, January 27, 2026: https://www.citizen.org/article/counterfeit-companionship-big-tech-ai-chatbots/
- Replika's February 2023 rollback of intimate features produced reactions Harvard Business School characterized as typical of losing a human partner. Public Citizen, January 27, 2026 (same URL).

What I could not verify:

- Italy's data protection authority fined Replika's developer 5 million euros, May 19, 2025. Reuters: https://www.reuters.com/sustainability/boards-policy-regulation/italys-data-watchdog-fines-ai-company-replikas-developer-56-million-2025-05-19/ (unverified; confirm the euro figure against the Garante's own release before print).
- Agent with no relay mechanism replying "Sent! [ok]" to a real person; daemon-injected messages producing "Tell who? (?)" because one visible thread was two transcripts; per-person isolation required to be structural because a prompt instruction leaks. `~/.claude/skills/agent-messaging-bridge/SKILL.md`, 3 August 2026.

- Amber's production LoopMessage sender `+19403012214`, and the retired sandbox sender `+17374294452` still on the account under a standing never-point-config-at-it rule. `~/.claude/skills/amber-hermes-ops/SKILL.md`, 3 August 2026.

What I could not verify:

- No misdirected send to a wrong human exists on record. Searched 2026-09-09 across the skills, blog archives, and repos: what exists is the fabricated-delivery case, the transcript-desync case, and the sandbox-sender near-miss, all printed above and labeled. The double-send is Jeremy's memory only, with no product, date, count, or SHA.
- No public case was found of a duplicate-send incident at scale, or of a cross-user visibility breach at a 2025-2026 companion bot with a confirmed affected-user count. The 2026 record is regulatory fines and misconfiguration exposure, not user-to-user leaks. Both appear here as field-observed and say so.
- Meta AI and Google Gemini stay out. Public Citizen references Meta's discover feed and a Gemini April 2025 incident; neither is a confirmed cross-user isolation failure with a count. Do not upgrade either without a primary source.

“Nothing you type in a chat changes a single weight.”

CHAPTER 52

Yelling Doesn't Make It Learn

"*Never attempt to teach a pig to sing; it wastes your time and annoys the pig.*"

ROBERT A. HEINLEIN, TIME ENOUGH FOR LOVE: THE LIVES OF LAZARUS LONG (1973)

Round three of the design jury, 12 August 2026. The jury flagged rows that had been sitting in pending forever, and I sent it back to check because I was certain it was a false alarm. It came back and agreed with me: those rows were intentional, tracer rows for airank's own instrumentation, permanently pending by design, working exactly as built. Case closed, go home. Except pulling that same thread one more inch turned up something underneath it that had been shipping to paying customers the whole time, and not one word of it was found by anybody correcting anybody in a chat window.

The rule I can actually document goes like this. 9 August 2026, airank. A P0 came down to two read-only SQL queries, and both of them had to say `UTC_TIMESTAMP()` and not `NOW()`, because `NOW()` on that box returns Central Daylight and lands five hours off. That is the whole difference between "the fix is verified live" and "the fix is verified at some point later this evening." One line of knowledge. The only interesting question about it is where that line lives. In my head at 1am it lives about ninety seconds. Typed into a chat window it lives until the window closes. Written into a file the agent reads at the top of every run, it lives until I delete it.

I know which of those three I reach for when it is late, and it is not the file. That same P0 is the one where I reported $738 of lost API calls when the real figure was 18,567 unarchived failures at $0.0129 each, about $240. Off by a factor of three, because I never checked my own per-call price before typing it into a priority ticket. Twenty-five years of shipping software and I cannot do one line of arithmetic. Somewhere in there I should have heard the little paperclip: *it looks like you're arguing with a math problem, would you like help with that?* Total n00b move. Nothing in that whole day got fixed by anybody being firmer in a chat window. It got fixed by commit `33a9188` and then two queries run against production with the correct timestamp function.

> ***Bottom line:*** *Nothing you type in a chat changes a single weight. The model that answers your fifth correction is byte-identical to the one that made the first mistake. What actually adapts is everything around the model: a skills directory on disk, a memory file the agent reads at the top of the session, an eval that fails when the bad behavior comes back, and (if you want real weight movement) a LoRA. Correction in chat is a temporary patch with a lifetime measured in tokens. Correction in a file is permanent. Yelling is the version that feels the best and lasts the least.*

• • •

WHEN IT BITES

- You correct the same thing four sessions in a row and think the tool is broken. Your correction has nowhere to live.
- Somebody on your team says "I trained it on our codebase" and means they pasted a style guide into a chat once, in March.

- A behavior you fixed in a long session comes back the moment context compacts, and you cannot tell whether the model got dumber or your instruction fell out of the window.
- You pay for a fine-tune to fix a formatting preference that a 40-line skill file would have fixed for free.

• • •

THE PATTERN

Three different mechanisms get called "learning" and only one of them touches weights.

In-context learning. You put examples or corrections in the prompt. The model conditions on them. It is fast, it is free, it works well, and it is gone when the conversation is gone. The model is the same mathematical function with a longer input (genai.stackexchange.com, 10 August 2025, flagged unverified because it is a Q&A answer, not a paper; the mechanism itself is not in dispute anywhere). The bug is that the change is stored in the transcript, and the transcript is not the product.

Harness adaptation. You write the correction to disk somewhere the agent reads on every run. Anthropic shipped Agent Skills on 16 October 2025, described in their own engineering post as composable filesystem-based capabilities that extend Claude without updating model weights (anthropic.com/engineering, 16 October 2025).

My own version of that, counted on 9 September 2026: 75 `SKILL.md` files in `~/.claude/skills/`, 1.5 MB of them all together. Seventy-five corrections I got annoyed by at least twice. I can give you that count to the file because `find` can count. What I cannot give you is which of the 75 actually get opened when it matters, because I never put a date on reviewing them, which makes me the guy in the "Don't" list at the bottom of this chapter. Writing it down is the cheap half. Knowing which written-down thing still earns its keep is the half almost nobody does, me included.

Memory is the same class. Simon Willison pulled apart Claude's memory feature on 12 September 2025 and found it works through visible tool calls, `conversation_search` and `recent_chats`, not some mystical injection layer. Anthropic then unified memory across Claude and Cowork on 25 August 2026 with it on by default for free, Pro and Max (CNET, 25 August 2026). Delete the store and the model is exactly as ignorant as it was before you met it.

Weight updates. This is the only one that is literally learning, and you have to pay for it deliberately. arXiv:2511.00130 (31 October 2025) is the one to read: LoRA gives the best balance for instilling new skills with minimal damage to base knowledge, while full supervised fine-tuning is highly susceptible to catastrophic forgetting. Ch. 7 covers the mechanics. The decision rule here is simpler: in-context is fast and capped by context length, LoRA and SFT are what you use when the thing you want cannot be said in a file (arXiv:2511.00130, 4 November 2025).

So the ladder, cheapest first: say it in the prompt, write it in a skill, write an eval that catches it coming back, and only then move weights.

Rung four I am taking on the paper's word and not my own. I went looking on 9 September 2026 for one case in my own work where a skill file failed and a weight update fixed it, and found zero: nothing in the airank blog, nothing in strip-club-sim, nothing in eldritchdm. Seventy-five skill files on disk and I cannot produce a single one that had to be promoted to a LoRA. I read that as a fact about the size of my problems rather than a fact about LoRA, so treat rung four as a hypothesis with a paper behind it and no receipt from me.

The rung nobody skips and everybody should: **the eval**. A skill file tells the agent what to do. An eval tells *you* whether it did. Without one you are running on vibes, and vibes have a known failure mode covered in Ch. 21: the behavior degrades, you do not notice, and you attribute the eventual blowup to a model update that never happened.

• • •

ONE WORKED EXAMPLE

airank, 12 August 2026. A multi-model design jury was running evals across 9 surfaces. Nothing about the setup was clever: deploy between rounds, re-jury the live state, see what it says now.

Round three, the jury caught a bug the jury itself had shipped two rounds earlier: timestamp parsing was broken in Chrome, silently wrong by five hours, and showing `NaN` in Safari. Nobody had

corrected the model about timestamps. Nobody yelled. The loop caught its own regression because the loop existed, ran against a deployed state, and had a place to report.

Pulling that same tracer-row thread found the actual defect underneath: **those rows were being published as user-facing queue position data.** Internal instrumentation was showing up in the product as a number a customer would read and believe.

• • •

THE QUIET FAILURE

The loud failure is the guy who thinks he "trained" ChatGPT by arguing with it: easy to spot, mildly funny, harmless to anyone but him.

The quiet failure:

You get a real behavior change in a long session, ship on it, and never write it down.

This one is nasty because it works. Hour three of a session, the agent has absorbed twelve corrections and is genuinely excellent, so you keep going. The session ends or compacts, and every one of those corrections evaporates at once, and the next session's output is worse in a way you will describe to somebody as "the model seems dumber today." It is not. You are just running the un-patched build for the first time in eight hours.

Second quiet failure: **your memory store fills with conclusions and you treat them as facts.** Automatic memory (on by default since 25 August 2026) writes what happened in a session. Ch. 15 is about the plan that outlives its own truth; a memory file has the same rot, except now it is being read into every session automatically and nobody scheduled a review.

Third: **you fine-tune for something a file would have fixed.** Full SFT is the loudest, most expensive version of writing down a preference (arXiv:2511.00130). Prove the skill file failed before you move a weight.

• • •

DO / DON'T

Do

- Write corrections to disk the second time you give the same one. First time is a fluke, second is a pattern.
- Keep skills small and loaded on demand: a 9,000-token system prompt of accumulated grievances is worse than four 200-token files. The 9,000-token one goes in hard, barely fits, and nobody gets all the way through it. That's what she said.
- Pair every behavioral rule with a check that fails when the rule breaks, and re-run it against the deployed state between rounds. Rule without eval is a wish (Ch. 21).
- Run the refresh-resume pattern: at the end of a session, harvest the durable lessons into skills and write a handoff file with what is true right now. Then clear. The context window is a workbench, not a filing cabinet.
- Separate facts from conclusions in the memory file. Paths, commit SHAs and dates survive the trip. "The bottleneck is X" does not.

Don't

- Don't say "I trained it" about anything you typed in a text box.
- Don't diagnose a model regression from one bad session; check whether your context got compacted first.
- Don't let automatic memory accumulate unreviewed. Put a date on the review.
- Don't rely on a rule staying in effect just because you wrote it once in a file, in March, and never checked. An unchecked CLAUDE.md is a GeoCities page with a spinning UNDER CONSTRUCTION gif on it: somebody meant it in 1999, nobody has opened it since, and it is still technically serving.
- Don't yell. It works for about four messages, and it feels like an accomplishment, which is the expensive part.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 7 is the weight-update chapter: LoRA, SFT, what it costs and when it is worth it. This chapter argues for not going there first. Ch. 13 is context management, the mechanism underneath the quiet failure here: your corrections live in the window, and the window is finite and gets compacted. Ch. 21 is evals: skills are how you write the correction down, evals are how you find out it stopped working. Ch. 61 closes the loop on what the harness actually is, once you accept that the model is a fixed function and everything that improves is around it.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (weights don't move from chat; the harness is what adapts). Argument, not citation.

Verified:

- Claude memory works through visible tool calls (`conversation_search`, `recent_chats`), not automatic injection. Simon Willison's Weblog, 12 September 2025: https://simonwillison.net/2025/Sep/12/claude-memory/
- Anthropic unified memory across Claude and Cowork, on by default for free, Pro and Max. CNET, 25 August 2026: https://www.cnet.com/tech/services-and-software/anthropic-unifies-memory-across-claude-and-cowork/
- Agent Skills introduced as composable filesystem-based capabilities that extend Claude without updating model weights. Anthropic Engineering Blog, 16 October 2025: https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills
- LoRA gives the best balance for instilling new skills with minimal impact on base knowledge; full SFT is highly susceptible to catastrophic forgetting; in-context learning is fast but capped by context length. arXiv:2511.00130 (Bohnet et al.), 31 October to 4 November 2025: https://arxiv.org/abs/2511.00130
- Model weights do not update from conversation; the model is the same function over a longer context. genai.stackexchange.com, 10 August 2025, marked **unverified** (community Q&A, not a primary source): https://genai.stackexchange.com/questions/2463/why-doesnt-chatgpt-learn-from-its-interactions-with-users
- P0 verification came down to two read-only SQL queries using `UTC_TIMESTAMP()`, because `NOW()` on that host is CDT and five hours off; 18,567 unarchived failures at $0.0129 each, about $240, first reported as $738; fix commit `33a9188`. airank build log, 9 August 2026: `~/Projects/airank/blog/2026-08-09-the-plan-that-did-not-spend.md` (same incident in Ch. 15)
- Author's own skills directory: 75 `SKILL.md` files, 1.5 MB total, in `~/.claude/skills/`. Counted on the author's machine, 9 September 2026 (`find ~/.claude/skills -name SKILL.md | wc -l`, `du -sh`)
- Worked example: the jury that caught its own regression, airank, 12 August 2026. `~/Projects/airank/blog/2026-08-12-the-jury-caught-its-own-regression.md`

What I could not verify:

- Skills counts here are the author's machine on 9 September 2026 (75 files, 1.5 MB). Which of them see regular use, and how often the refresh-resume cycle actually runs, are not instrumented. The chapter says so rather than guessing.
- No published Anthropic paper states outright that models do not learn from chat corrections; the claim is implicit in the memory and skills docs, never headlined.
- No documented production incident from a major provider showing damage from users mistaking chat "learning" for weight updates. Field-observed, told as such.

- arXiv:2511.00130 covers LoRA vs SFT vs in-context learning from a data-scarce angle, not the "why yelling does not update weights" angle.
- No official Anthropic statement on a Claude fine-tuning API timeline. The reference is an engineering blog post, not the Platform docs. Do not imply availability in print.

CHAPTER 53

Stop Calling Wandering Research

"Ultimately this must remind us of the famous drunk who looked for his wallet, not where he had lost it, but under the street lamp, "because the light is better there," or of the doctor who gave all his patients fits because that was the only sickness he knew how to cure."

ABRAHAM H. MASLOW, MOTIVATION AND PERSONALITY, 2ND EDITION (1970)

Wells Fargo was the #1 answer for "best wireless earbuds" 126 times. One hundred and twenty-six ranked sessions, in a product I built, sitting in a database I own, with a detector already running whose entire job was catching exactly this. The detector said clean. The dashboard said Wells Fargo makes great earbuds. Everything in the pipeline was working as written, every test was green, and nobody on the project noticed for days. The reason it survived is not that the code was wrong. The code was fine. It was looking in the wrong half of the room.

An agent opens fourteen files, reads a third of each, writes a confident summary, and calls it research. Nobody knows which files it skipped, or why it stopped at fourteen. It stopped because the context got long and the model got bored, and "bored" is not a stopping rule. You approved it anyway, because the summary read well and you had a standup in nine minutes.

> ***Bottom line:*** *Research is a search over a space, and a search needs three things a wandering agent never has: a definition of the space, more than one way to cut it, and a rule that says when it's empty. Reading files until something interesting shows up is sampling biased toward whatever sorted first alphabetically, and a smarter model doesn't fix that. What fixes it: multi-modal search (by container, by content, by entity, by time), a loop that runs until new results stop being new, and a critic whose only job is to say what was never looked at.*

• • •

WHEN IT BITES

- Your agent reports "no results" and you believe it. It ran one query shape, got nothing, and turned an empty query into an empty world.
- The summary cites six sources. There were four hundred. Nothing in the output tells you the ratio.
- Two runs on the same question return two different answers, both looking complete, because both wandered somewhere different.
- An extractor marks 1,461 items "nothing found" and the data is sitting right there in a shape it never learned to see.
- Someone asks "did you check the logs from before the deploy" and the answer is a long pause. Time was never an axis.

• • •

THE PATTERN

Tree of Thoughts (arXiv, May 2023) took Game of 24, where GPT-4 with chain-of-thought scored **4%**, and got **74%** by making the search structure explicit: generate candidate intermediate steps, evaluate them, keep branching, backtrack when a branch dies. Same model, same task. One version had a search space with edges and the other had a monologue. That is a benchmark score, not a production incident, and I have not found anybody who published the same comparison against a live system.

Then it scaled. Anthropic's multi-agent research system (June 2025) put Claude Opus 4 with Sonnet 4 subagents against a single Opus 4 agent on an internal research eval and measured a **90.2%** improvement. The same writeup says token usage explains **80% of the variance** on BrowseComp. Those are one finding: what makes research work is covering more of the space, and subagents buy coverage in parallel instead of serially blowing your context window.

The evaluation side caught up in 2026. DeepResearch Bench (May 2026) is 100 PhD-level research tasks across 22 fields, and it does not score prose. It scores citation accuracy and **effective citations**: Gemini-2.5-Pro Deep Research scored 48.88 overall with an average of **111.21 effective citations**, OpenAI Deep Research 46.98. AgentSearchBench (arXiv, April 2026) scored the **search trajectory** itself. Between the two of them they score whether the citations exist, whether they hold the claim up, and whether they earned their place. Page count is not on the list.

Four cuts, not one. These are my labels off my own whiteboard, not a standard anybody voted on, and not identifiers in my code either: I grepped airank's `app/` and every skill under `~/.claude/skills/` on 9 September 2026 and the four words appear nowhere outside this manuscript. Search by *container* (which files, buckets, tables, repos exist at all), by *content* (grep, embeddings, full text), by *entity* (this brand, this user id, this error code, everywhere it appears), and by *time* (what changed between two timestamps). A wandering agent does content only, one query, usually the user's own phrasing. Container search finds the 300 files nobody grepped because they were gzipped; time search finds the thing that was true last Tuesday. What each cut costs per run I have not measured, so I cannot tell you which one to buy first when the budget is tight.

Loop until dry. No benchmark I have found writes this one down, so take it as shop practice and not a published criterion. The stopping rule isn't "I have enough." It's: run the next query, and if it returns nothing that wasn't already in the set, the vein is out. "Enough" is a feeling, calibrated on training data, not on your bucket.

A completeness critic. A second pass whose only job is answering "what did the first pass never look at," expressed as a set difference: containers enumerated minus containers opened. Not a quality review, a coverage review. LLM-as-judge with a fixed rubric (factual accuracy, citation accuracy, completeness, source quality, tool efficiency) tracked human judgment most consistently in Anthropic's June 2025 writeup. Give the judge the rubric or it grades vibes.

Zero output is a signal, not an absence. Look at how the nothings cluster. Random nothing means the data isn't there. Clustered nothing (all one file type, all one category, all one date range) means your reader is broken and is reporting its own blind spot as a fact about the world.

• • •

ONE WORKED EXAMPLE

airank, 6 August 2026. The ChatGPT collector marked **1,461** product-listing phrases as `no_products_extracted` and stalled the run. The obvious read: ChatGPT stopped returning products. Wrong, and expensive: it required no further work to believe.

The data was there, in four different HTML shapes: carousel product cards, semantic tables whose columns weren't positional, and medallist rankings. Pages carried 5 to 11 citations each. The extractor knew one shape and reported the other three as absence, measuring its own coverage gap and printing the result as a fact about ChatGPT.

I wrote that extractor. I shipped the one shape it knew, watched it mark 1,461 phrases as nothing-found, and my first instinct was to go argue with OpenAI about it. That is not a model problem. That is me getting pwned by my own `if` statement.

What broke it open wasn't a better parser. It was cutting the zeros a different way: cluster the `no_products_extracted` rows by category and file type, and the nulls clump. A two-minute `GROUP BY` reversed the diagnosis.

The same run turned up a truncation bug chopping multi-word brands: `Land Rover Defender` became `Land`, and **112 brands** were unreachable by name because the index held a word that was never the brand. Only entity-axis search surfaces that: it asks where a brand appears and gets nothing for a brand that visibly appears everywhere.

Outcome: pages got saved to object storage (109KB gzipped each), keyed by phrase plus reason, so the raw evidence survived the extractor's opinion of it. And the framing changed. Instead of ranking brands, attributes became first-class entities: *HubSpot owns "best overall for small business" in 67% of sessions, n=9, CI 41-86%*, a sentence that carries its own sample size and uncertainty where the brand ranking it replaced carried neither.

Two days later, 8 August 2026, the finding I still think about. Wells Fargo ranked **#1 for "best wireless earbuds" 126 times**, because an unguarded `WF` alias matched Sony's model number `WF-1000XM5`. A detector existed to catch alias collisions. It was blind to this one because it checked phrase vocabulary and never checked answer text. It searched exactly the half of the space where the bug was not.

I specced that detector. I decided phrase vocabulary was where aliases collide, because that is where I would have put a collision if I were writing one, and Sony did not consult me when it named `WF-1000XM5`.

• • •

THE QUIET FAILURE

The loud failure is the agent that says "I could not find anything." Loud gets fixed, because a human reads it and goes looking.

The quiet failure:

The agent finds something, writes it up well, and the write-up gives you no way to know what fraction of the space it covered.

Six citations look identical whether the pool was six documents or six thousand. This is why DeepResearch Bench counts effective citations and AgentSearchBench scores the trajectory: a reader reads confidence as coverage every time.

Second quiet failure: **you accept a code fix as a verified fix.** On 9 August 2026 the airank council's P0 was two read-only queries. Commit `33a9188` had shipped archival to MinIO with three green tests and was marked done. Code-fixed and production-verified are different states, so they ran it: last 15 minutes, **130/130 captures archived, 624/624 observations archived**, plus one skipped object pulled back out to prove retrieval worked (phrase-1463, 1,859 chars, 5 citations, `model=gpt-5.6-luna`). Ninety seconds of querying converted a hopeful state into a known one. Phrase coverage went from 27% to 77%.

Third: **you tune the model when the problem is the axis.** The alias detector was not dumb, it was pointed at phrase vocabulary. A better model checking the same half of the space finds the same nothing, faster.

• • •

DO / DON'T

Do

- Enumerate the space first. Count the containers, count the ones you opened. That ratio goes in the output.
- Search on four axes, my labels: container, content, entity, time. If the answer only ever came from one, you have a sample, not a search.
- Make the stopping rule "the last query returned nothing new," and log the query that came up dry. That log is the proof.
- Cluster your zero-result rows. Clumping by category or file type means the reader is broken, not the source.
- Run a completeness critic as a separate pass, with a rubric, and buy coverage with subagents when the question is wide. 90.2% over single-agent, at roughly 15x the tokens of single-turn chat.

- Keep the raw artifacts and point the detector at the output text, not just the input vocabulary. `WF-1000XM5` was in the answer, never in the phrase.

Don't

- Don't accept "no results" from one query shape. Empty query, not empty world.
- Don't let citation count stand in for coverage. Effective citations is a metric because raw count was gameable.
- Don't mark a fix done because the tests are green. Green tests and a production query are different states.
- Don't ship a research summary with no denominator anywhere in it.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 18 is tool design: this is what happens when the tools are fine and the search over them is not. Ch. 22 is evaluation, and this is the case study for scoring the trajectory and not only the answer, because a wandering run and a thorough run produce identical-looking prose. Ch. 15 is the plan that is not the work, same rule one level down: the cheapest verification outranks the best theory. Ch. 52 and Ch. 54 sit either side as the rest of the research loop.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (research is a search with a stopping rule): argument, not citation.
Verified:

- Tree of Thoughts: structured exploration over intermediate reasoning steps; GPT-4 chain-of-thought 4% on Game of 24, ToT 74%. arXiv (NeurIPS 2023), May 2023: https://arxiv.org/abs/2305.10601
- Multi-agent research system: Claude Opus 4 with Sonnet 4 subagents outperformed single-agent Opus 4 by 90.2% on internal research eval. Anthropic Engineering Blog, June 2025: https://www.anthropic.com/engineering/multi-agent-research-system
- Token usage explains 80% of variance on BrowseComp; multi-agent systems use ~15x the tokens of single-turn chat. Same source, June 2025.
- LLM-as-judge with a single rubric (factual accuracy, citation accuracy, completeness, source quality, tool efficiency) was most consistently aligned with human judgment. Same source, June 2025.
- DeepResearch Bench: 100 PhD-level tasks across 22 fields, scored on citation accuracy and effective citations. ICLR 2026, May 2026: https://deepresearch-bench.github.io/
- Leaderboard: Gemini-2.5-Pro Deep Research 48.88 with 111.21 average effective citations; OpenAI Deep Research 46.98. Same source, May 2026.
- AgentSearchBench: task-performance-based relevance and search trajectory quality. arXiv, April 2026: https://arxiv.org/html/2604.22436v1
- OSWorld-Verified: best agents 86.1% vs 72.36% human baseline. O-Mega AI, October 2025: https://o-mega.ai/articles/the-best-ai-agent-evals-and-benchmarks-full-2025-guide
- Worked example 1: 1,461 phrases marked `no_products_extracted`, four HTML shapes, 112 brands lost to truncation, 109KB gzipped pages, attribute-as-entity reframing (n=9, CI 41-86%). airank, 6 August 2026: `~/Projects/airank/blog/2026-08-06-the-data-was-there-the-whole-time.md`
- Worked example 2: three-arm browser measurement (467.3 / 68.3 / 468.6 req per question); Jaccard 0.60 isolation test; Wells Fargo #1 for "best wireless earbuds" 126 times via `WF` matching `WF-1000XM5`. airank, 8 August 2026: `~/Projects/airank/blog/2026-08-08-the-council-ran-the-experiment.md`

- Worked example 3: commit `33a9188` code-fixed vs production-verified; 130/130 and 624/624 archived; phrase-1463 retrieved; coverage 27% to 77%. airank, 9 August 2026: `~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md`
- The four axis names are Jeremy's own labels, not code identifiers or a documented taxonomy: grepped `~/Projects/airank/app` and `~/.claude/skills/` on 9 September 2026, zero hits outside this manuscript. The text says so where it introduces them.

What I could not verify:

- Per-axis cost is not measured. No per-run token or dollar figure exists for container, content, entity or time search in airank, so the chapter ranks none of them by price and publishes no table.
- Loop-until-dry, premature loop termination as a named failure, and per-strategy token efficiency (breadth-first vs depth-first) are field practice here, not published findings; hedged as such in the text. The 15x figure covers multi-agent vs single-turn chat only.
- No public incident write-up compares structured search methodology against production agent measurements; the Tree of Thoughts numbers are benchmark, and the text says so.

CHAPTER 54

Give It Back (OSS Opens Doors)

"A project some random person in Nebraska has been thanklessly maintaining since 2003"

RANDALL MUNROE, XKCD #2347, DEPENDENCY (2020)

At 10:37 UTC on 7 August 2026 I was not contributing to open source. I was trying to keep my own agent alive. It talks to MCP servers over an SSE stream, the server sent a long run of frames the client had no use for, and the Rust reader called itself once per skipped frame. Skip enough of them and the stack runs out. That does not raise an error you read over coffee. The process aborts. My agent died mid-run, the log ending mid-sentence like a dial-up connection dropping at 94%.

So I read the diff instead of writing a workaround. The recursion became a loop. I opened the pull request at 10:37:22 and a maintainer named DaleSeo merged it at 14:34:19: 3.95 hours from a stranger's patch to code that ships to everyone using the official Rust SDK for the Model Context Protocol. Nobody in that thread knew who I was, and it did not matter. The diff either fixed the stack overflow or it didn't.

> ***Bottom line:*** *A merged pull request in somebody else's repository is the one credential in software nobody can fake: the diff is right there and a stranger can check it in eleven seconds without calling anyone. The way you get one is not "contribute to open source." Nobody ever got one that way. You run other people's code in production, it bites you, and you refuse to work around it. Fifty-nine of mine are merged across 25 organizations, 58 of them between 3 August and 9 September 2026. That looks like a flex until you learn what it is: 58 nights my agent broke.*

• • •

WHEN IT BITES

- Your agent dies in production and the cause is four layers down in a dependency you did not write. Carry the workaround forever, or spend an afternoon in somebody else's code.
- You apply somewhere that builds on MCP and your resume says "deep experience with the MCP ecosystem." Theirs says a PR number and a merge date. One gets a reply.
- You need a bug fixed in a repo where nobody knows you, so you file an issue and wait. The person who landed three patches there gets read on Tuesday.
- You cite a merged count without saying what the diffs did, and the first person who checks finds a one-line docs anchor in the same column as a credential leak.

• • •

THE PATTERN

Start with what "merged" means, because people who have not shipped one hear it as "posted something." A pull request is a change you propose to a code base you cannot write to. Merged means a maintainer read your diff and pressed the button that puts your code in the branch everybody else installs. #1146 was merged by DaleSeo. `swiftlang/swift-package-manager` #10443 by ple-

marquand. Those are people defending their own on-call rotation. They owe you nothing, and no amount of networking makes them press it.

The counting rule, stated once and used everywhere in this book: merged pull requests in repositories I do not own, derived with `gh` on 9 September 2026. My own repos do not count. Open does not count.

REPO	PR	WHAT BROKE	HOURS TO MERGE
`openai/openai-agents-python`	#4606	`max_turns` clobbered a tripped input guardrail in streaming	1.41
`modelcontextprotocol/rust-sdk`	#1146	Skipping an SSE event recursed; a burst blew the stack	3.95
`huggingface/swift-transformers`	#384	Sampling returned one constant wrong token above 65,536 vocab	7.04
`pytorch/ao`	#4817	NVFP4/MX QAT backward dropped the bias gradient	13.87
`google-deepmind/open_spiel`	#1593	Quoridor forced-pass read the board before validating	38.45
`anthropics/claude-code-action`	#1719	`download_job_log` stalled forever on a hung fetch	39.71
`laravel/mcp`	#331	JSON-RPC notifications took an unvalidated params shape	41.76
`microsoft/agent-framework`	#7837	Gemini finish-reason fallback cascaded into usage attach	84.80
`google/adk-java`	#1449	`InMemoryArtifactService` ignored the `user:` namespace	87.04
`apache/tvm`	#20170	`fmod` mapped wrong in Relax ONNX constant folding	94.80
`envoyproxy/envoy`	#46954	`isUrlValid` hit signed-char undefined behavior	175.60
`elastic/kibana`	#287214	Document IDs containing slashes broke the query-rules route	205.58
`facebookresearch/xformers`	#1407	Missing `mslk` silently dropped the attention ops	226.12
`swiftlang/swift-package-manager`	#10443	Auth credentials followed a cross-host redirect	343.91
`huggingface/accelerate`	#4151	Disk offload crashed on FP8 tensors	779.92

That is 15 of 59. The other 44 are the same shape in other plumbing. Fourteen in `mozilla-ai` (`mcpd` config validation and tool schemas, plus `otari`, `any-llm`, `any-guardrail`, `apron-tools`, `apron-auth`, `llamafile`). Four in `mozilla` proper, all `firefox-devtools-mcp`, including a capability the server advertised but never implemented. Four in `apache`: a Beam flag collision, a Maven unbox, a Pulsar TLS cipher never passed to Jetty, and `calcite` #5223, which bounded DECIMAL expansion so an overflow check could not blow up on a huge exponent. Four more in `huggingface`, two in `microsoft` (`markitdown` RSS recursion, `FLAML` misaligned sample weights), two in `mistralai` (config validation that divided by zero), and singles in `openai`, `google`, `google-deepmind`, `anthropics`, `modelcontextprotocol`, `bytedance/deer-flow`, `OpenBMB/MiniCPM-V`, `TanStack/ai`, `SillyTavern`, `QwenLM/qwen-code`, two in `MiniMax-AI`, and `zai-org/SCAIL-2`, where a missing colon meant LoRA bias deltas were never fused. Plus `fumeapp/laranuxt` #14: a broken README link merged in 2020, six years earlier, and the only one that was not an outage.

I did not pick those projects off a list of good first issues. My agents run over MCP, so MCP plumbing is my plumbing. My cluster is Apple Silicon under MLX, so Swift tooling, quantization and offload code are my tooling. Every one of those repos stood between me and a working night, and working around them cost more than fixing them.

Now the reality check against my own hero story. Median time to merge across all 59: 111.55 hours, or 4.6 days. Fastest was #4606 at 1.41 hours. Slowest was the `accelerate` FP8 disk-offload crash, which sat 779.92 hours: 32.5 days. Total, 5,462 added lines and 539 deleted. If you plan on the 4-hour version you quit in week two.

The least exciting row in that table is the one I would hand a hiring manager. `swiftlang/swift-package-manager` #10443: `DataTaskManager` kept the authentication header when a download redirected to another host, so a registry credential goes wherever the redirect points. Sixty additions, seven deletions, 343.91 hours. Two weeks is the right caution for a stranger's security patch inside Apple's package manager.

At the other end, `anthropics/claude-code-action` #1579 is one changed line: a broken anchor to the Bedrock section of `cloud-providers.md`. It sat 164 hours, a maintainer merged it, and I felt like a hero over a link. The bar for a first patch is not a rewrite. It is something wrong plus somebody willing to fix it.

• • •

ONE WORKED EXAMPLE

7 August 2026, `modelcontextprotocol/rust-sdk` **#1146.**

`SseAutoReconnectStream` recursed into itself for every event it skipped. On a normal stream you never notice. On a server emitting a long burst of non-JSON frames, each skip adds a stack frame and the client dies with an abort. Not a hung request. Not a dropped message. Not a retry you can log. The process.

Before I found that, Jeremy Schoemaker, Lead Linux Security Engineer at a bank back when that title meant something, restarted his agent twice, rewrote his own error handling, and drafted an issue blaming the server. The defect was a function calling itself.

The fix wraps `poll_next` in a loop and replaces the tail calls with `continue`: 164 additions and 124 deletions in one file, the honest size of "same behavior, one less stack frame."

Seven hours later I filed #1152 in the same repo: the SSE GET stream never mapped 401 and 403 challenges, so the auth client never got the signal to refresh its token. 136 additions, merged in 1.71 hours. Faster because it was smaller, and because the maintainer already had a data point from that morning saying my diffs do not waste an afternoon.

Outcome: both merged, both fixing a failure that reported itself as something else.

• • •

THE QUIET FAILURE

The loud failure is padding the number. Somebody claims 200 contributions, you open the list, it is 190 typo fixes and a fork of their own repo. Dies on contact.

A PR title is a filename. Anybody who used Limewire in 2000 knows the lesson: the file said `Metallica_-_One.mp3`, 3.2 MB, downloaded clean, and what came out of the speakers was a man coughing for four minutes. Titles are metadata somebody typed. The diff is the bytes.

You cite real merged PRs, and everyone reasons from the titles anyway. `fumeapp/laranuxt` #14 and `mozilla-ai/mcpd` #278 both start with "fix." One is a README link. One is 293 added lines that reject conflicting options and roll back a partial move. Same column, same verb, nothing like the same unit of evidence. I am the person most tempted to forget that, because it is my table.

Second, **you blur merged into open.** Fifty-nine merged is a claim about other people's judgment. My 101 open PRs are a claim about my own output. Some will be closed unmerged, some will rot in the queue, and none count until somebody presses the button. If I say "contributor to Apple," ask which repo and which state, and I should be the one asking first.

Third, the one the chapter title writes a check for. I can show you the keys. I cannot show you a hinge swinging. On 9 September 2026 I searched my own inbox and calendar for the moment a merge turned into a reply, an intro, an interview, a tier bump, anything. What I found was 59 merges and a spotless inbox. Zero. Not one of those 59 has a dated outcome attached to it.

So take the title as a bet, not a receipt. My honest belief, and it is only a belief, is that landing #1146 changes how a stranger reads my name three months later, and that the 59 rows are an asset that has not been called yet. Ask me again in a year. Until then the only thing I can prove is the part nobody can fake: the diffs shipped. The credential nobody can fake is also, apparently, the one that does not ring your phone.

• • •

DO / DON'T

Do

- Fix the bug that bit you. Fifty-eight of those 59 are outages I had, not projects I adopted.
- Send the boring one. The credential-leak patch took two weeks and beats anything flashier.
- Send the tiny one too. A broken anchor merges, and now you have a merge date in that org.
- Say the PR number, not the company name. "One merged patch in `swiftlang/swift-package-manager`" cannot be attacked. "Contributor to Apple" can.
- Plan on 4.6 days, not 4 hours.

Don't

- Don't count your own repos. Repos prove you type. A stranger's merge proves somebody approved the work.
- Don't present a README link and a 293-line rollback fix as the same evidence. Someone opens both.
- Don't file volume to hit a number. Maintainer attention is the scarce resource, and you spend it whether the PR merges or not.
- Don't quote your open count as a contribution count. It is a queue, not a record.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 24 is the same instinct pointed the other way: the dependency you did not write can betray you, so read the diff. This chapter is what you do once you find the bug. Ch. 31 gives three moves when your agent blows up (retry, degrade, escalate); the fourth, which only exists in open source, is to fix the thing that broke so it stops breaking for everyone. Ch. 61 is this compressed: the patch you land upstream is the lever, because you do the work once and it ships to people you will never meet.

• • •

SOURCES AND RECEIPTS

Counting rule: merged PRs in repos Jeremy does not own, via `gh` on 2026-09-09. Timestamps UTC.

Verified: 59 merged across 25 organizations, 58 between 2026-08-03 and 2026-09-09 plus one in 2020. Median 111.55 hours, fastest 1.41, slowest 779.92. 5,462 additions, 539 deletions. Open at pull: 101, across 52 orgs.

Org counts: mozilla-ai 14, apache 5, huggingface 5, mozilla 4, microsoft 3; two each in mistralai, google-deepmind, anthropics, google, openai, laravel, modelcontextprotocol, MiniMax-AI; one each in elastic, envoyproxy, facebookresearch, bytedance, OpenBMB, pytorch, zai-org, TanStack, SillyTavern, QwenLM, swiftlang, fumeapp.

- `openai/openai-agents-python` #4606, merged 2026-08-23, 1.41 h.
- `modelcontextprotocol/rust-sdk` #1146, opened 2026-08-07 10:37:22, merged 14:34:19 by DaleSeo, +164/−124. https://github.com/modelcontextprotocol/rust-sdk/pull/1146
- `modelcontextprotocol/rust-sdk` #1152, merged 2026-08-07, 1.71 h, +136.
- URL pattern `github.com/<repo>/pull/<number>` for: `huggingface/swift-transformers` #384 (7.04 h), `pytorch/ao` #4817 (13.87), `google-deepmind/open_spiel` #1593 (38.45), `anthropics/claude-code-action` #1719 (39.71), #1579 (164.49), `laravel/mcp` #331 (41.76), `microsoft/agent-framework` #7837 (84.80), `google/adk-java` #1449 (87.04), `apache/tvm` #20170 (94.80), `apache/calcite` #5223 (185.01), `envoyproxy/envoy` #46954 (175.60), `elastic/kibana` #287214 (205.58), `facebookresearch/xformers` #1407 (226.12), `mozilla-ai/mcpd` #278 (351.49, +293/−5).

- `swiftlang/swift-package-manager` #10443, merged 2026-09-05 by plemarquand, 343.91 h, +60/-7. `huggingface/accelerate` #4151, merged 2026-09-07, 779.92 h. `fumeapp/laranuxt` #14, merged 2020-09-01, 140.79 h. Same URL pattern.

Belief, not receipt (Jeremy, 2026-09-09): inbox and calendar search on 2026-09-09 found no merged PR that produced a named intro, reply, interview, or tier advancement. Count of door-opening outcomes with dates: zero. The chapter prints this as his belief and says so.

Why he was in these code bases: Jeremy, 2026-09-09: his agents (aigate, eldritchdm, ShoeAgent) run over MCP, so MCP bugs hit his production first; the Swift, MLX, quantization and offload filings come from his private exo cluster on Apple Silicon.

What I could not verify:

- Fifty-eight of the 59 have no first-person account of how the bug was hit. Get the story for any PR that will carry weight in a talk.
- No maintainer quotes captured; any quote must come from the PR threads.
- Earlier drafts cite 15 merged across 6 orgs; the September 2026 resume cites 58 across 25. Both were capped pulls. The rule at the top supersedes them.

“AI *did not kill jobs for good, kind, responsible engineers.*”

CHAPTER 55

You Don't Have to Hire Assholes Anymore

"It is difficult to get a man to understand something, when his salary depends upon his not understanding it!"

UPTON SINCLAIR, I, CANDIDATE FOR GOVERNOR: AND HOW I GOT LICKED (1935)

Nine systems told me everything was fine. The database dump came back 4KB. The binlog shipped truncated. A health check said "Up 6 seconds" on a container that was doing nothing at all. Every dashboard was green, every job reported success, and 576 records were gone. It took me most of a day to work out what the dumps, the binlog and the dashboards all had in common, and when I finally had it, the answer was also the thing wrong with an engineer I once kept for four years. Green report. Empty table.

He rolls in at noon. Two-hour lunch. Gone at five. Somewhere in that window he writes the payment retry logic nobody else understands, and when support forwards him a bug report he replies that the users are too stupid to read the form. You keep him for four years. You lose two people who were nicer, cheaper, and would have stayed.

> ***Bottom line:*** *AI did not kill jobs for good, kind, responsible engineers. It killed the ones you put up with because they were so damn good. Coding knowledge got leveled. The thing that used to buy a jerk immunity, knowing the incantation nobody else knew, is now a $20/month subscription. What did not get leveled is architecture, networking, and the basics (does he know what a function is, can he parse the JSON, does he understand an array versus an object). What went up in value: addressing a human being, admitting a mistake out loud, being kind to the person who has to use the thing.*

• • •

WHEN IT BITES

- Your one irreplaceable engineer is irreplaceable because he made himself irreplaceable. That was the work product. Not the code.
- A junior asks him a question in standup and gets a sigh. She stops asking. Six months later she stops working there.
- Support escalates a real bug. He closes it "works as designed." The customer churns and the ticket is marked resolved.
- Entry-level tech hiring dropped 73.3% from 2024 into early 2025 (Ravio, via Dice). You told yourself that was AI eating juniors. Some of it was.

• • •

THE PATTERN

Robert Sutton wrote *The No Asshole Rule* in 2007. It sold, everyone quoted it in a management offsite, and almost nobody enforced it, because enforcement had a price: the guy.

Netflix ate the cost anyway. Reed Hastings put "no brilliant jerks" in the culture doc and made it a firing benchmark rather than an HR poster. Brendan Gregg wrote in November 2017 that in over

three years at Netflix he worked with zero brilliant jerks, and laid out what they cost everywhere else: attrition, demoralization, lowered technical IQ, hostile work environment. Rob Bowley wrote it again in October 2024 after one toxic engineer evaporated a department's culture.

By 2026 the tradeoff got cheap.

The retry logic is not a moat anymore. Claude and Cursor and Copilot will write it, and they will explain it to the person who did not write it. The specific knowledge that made a jerk load-bearing (this codebase's weird conventions, this framework's undocumented behavior, the exact incantation for the deploy) got commoditized first. What did not: knowing the retry belongs in a queue and not a cron, knowing your MTU is wrong, knowing what a 502 from the load balancer means versus a 502 from the app.

That commoditization is what I watch happen in my own repos, week after week. No named company is on record saying the tools took its one expert off the critical path. Take this as my belief: the load-bearing specialist is already gone in most shops and the org chart has not noticed.

The productivity story does not say what the LinkedIn version says. METR ran a randomized controlled trial in July 2025 with experienced open-source developers on their own repos. Using AI tools made them 19% slower, though they believed they were faster. The real claim is narrower: the *floor* came up. Somebody with solid fundamentals and no encyclopedic memory of your codebase can now get to a competent patch. That does not make them fast. It makes them viable, and viable plus decent beats brilliant plus corrosive over any horizon longer than a quarter.

The other half of the market is uglier. Stanford's digital economy work, cited December 2025, found junior developer employment for ages 22 to 25 down nearly 20% from its late-2022 peak. Forbes, August 9, 2026: 33 consecutive months of decline for juniors. A 2024 SHRM survey found 70% of hiring managers thought AI could do intern work, 57% trusted AI's output over an intern's.

That is the real opportunity and almost nobody is taking it. The junior you would have hired in 2019 and spent two years teaching should get productive in a fraction of that now, because the tooling closes the syntax gap. Should. The macro numbers are real; I found no published case of a junior with AI matching the specialist she replaced, so that sentence is a bet, not a finding. What she brings that the tooling does not: she will tell you when she broke something.

• • •

ONE WORKED EXAMPLE

airank, August 8, 2026. I wrote a post called "Everything Reported Success."

I built every one of those nine checks myself. Jeremy Schoemaker, Lead Linux Security Engineer at a bank before some of my staff could drive, wired up nine green lights that all measured whether a process exited zero. Nine for nine. Total n00b move, twice in one career.

I got them back by checking effects instead of reports. Not "did the job say it worked," but "is the row there."

That is the brilliant jerk. He reports success: commit count fine, tickets closed, standup crisp and technical and slightly condescending. The report is green. The effect is that your best mid-level engineer took a recruiter call last Tuesday, support stopped filing bugs against his service, and nobody on that team has proposed a risky idea in eleven months.

So hire the way I recovered those 576 records. Do not measure brilliance, which is a report. Measure effects. Who stayed. Who got better while sitting next to him. Who is willing to say "I broke it" in a channel where other people can see.

The counter-case, same house. In `commander-in-chief`, my Godot game project, CODE_OF_CONDUCT.md opens with "Be excellent to each other, and to the sim," and carries a real enforcement ladder: correction, warning, temporary ban, permanent ban. The defect file in that same repo, re-triaged 2026-07-28, wears two badges at the top: open 54, resolved 68. The 54 break out as 2 major, 37 minor, 15 cosmetic, and zero critical.

Fifty-four open findings is not a flex. It is me publishing a badged list of 54 things wrong with my own game. Nobody got humiliated in a code review to produce that list, and the reason is boring rather than noble: it is a test-first deterministic codebase, so the standards live in the tests instead of in one guy's head. One small project, so take it as a demonstration.

PAR Program, 2012 to 2015. Twelve people, six in the office and six remote. One of them was the four-year guy at the top of this chapter, and he is mine: I hired him, I signed his checks, I read his tickets. He owned the payment retry logic. I told myself that code was too expensive to re-learn,

which is the sentence a CEO says right before he loses somebody good, and I lost at least two of them, both nicer and both cheaper. I kept the one who could write the payment code and ran off the two who could have written it. That is CEO math, performed by a CEO, at $12M-exit velocity (GoSocial, November 2015). No HR file, no exit interview, no write-up in any repo I own, so take it as recollection and not a record.

And nobody has published the before-and-after. I went looking for two numbers around a toxic engineer's exit date, tickets per week or departures per quarter, and found none. Gregg in 2017 and Bowley in 2024 describe the damage; neither counts it. This chapter is an argument with receipts on the edges, not a measurement.

• • •

THE QUIET FAILURE

The loud failure is the guy screaming in a code review. Everyone sees it and HR gets an email.

The quiet failure:

You do not fire him. You stop hiring anyone who would have to work with him.

The team stays small because a bigger team means more exposure. You have built the entire org chart around one man's temperament and called it focus in the board deck. I have written that deck.

Second quiet failure: **you replace the wrong skill.** You look at AI closing the coding gap and conclude you need fewer people. What you need is a different intake filter. The fundamentals have to be there, and if they are not, the model will happily help someone produce confident garbage at volume. Past that bar, the ranking changed. In 2015 I would have taken the sharper engineer. In 2026 I take the one who will call the customer back.

Third: **you mistake conflict for standards.** Some of the best engineers I know will tell you your design is wrong in front of people and are not assholes: they are arguing about the work and update when you show them a measurement. The jerk is the person whose disagreement is about rank. Anyone who spent 1998 on Usenet can tell the two apart in one post: "your benchmark used the wrong MTU" versus "RTFM, this has been covered." Same technical content, one of them is recruiting for a war.

• • •

DO / DON'T

Do

- Screen for the basics hard, then stop screening for cleverness.
- Ask every candidate for a story about a mistake that hurt someone else's work. Listen for whether anyone else in the story is a moron.
- Make the enforcement ladder explicit and public the way `commander-in-chief`'s CODE_OF_CONDUCT.md does.
- Put the standards in tests and docs rather than in a person. The indispensable-jerk problem is a documentation problem wearing a personality.
- Hire the junior anyway, and measure effects.

Don't

- Don't sell the productivity story past the evidence.
- Don't confuse blunt with toxic, or pleasant with kind. Plenty of pleasant people will let you walk into a wall rather than have an uncomfortable conversation.
- Don't keep a brilliant jerk one more quarter because a project is mid-flight. It is always mid-flight.
- Don't let AI be your excuse for not training anyone. Those hiring managers will be short of senior engineers in 2031.
- Don't run this as an HR initiative. Netflix's version worked because the CEO enforced it.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 54 is what the work became once the model does the typing; this chapter is the staffing consequence: if typing is commodity, you hire for judgment and character, not talent bundled with contempt. Ch. 60 aims the same argument at the candidate's side of the table. Ch. 15 is the mechanical cousin of "Everything Reported Success": a report is not an effect, whether it comes from a health check or a performance review.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (AI leveled coding knowledge, so tolerating brilliant jerks lost its excuse): argument, not citation.

Verified:

- Robert Sutton, *The No Asshole Rule: Building a Civilized Workplace and Surviving One That Isn't*, 2007. Warner/Business Plus. https://www.amazon.com/Asshole-Rule-Civilized-Workplace-Surviving/dp/0446698202
- METR, July 2025 RCT: experienced open-source developers took 19% longer on their own repos when using early-2025 AI tools. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- Entry-level tech hiring down 73.3% from 2024 into early 2025. Ravio 2025 Tech Job Market Report, cited in Dice, July 2026. https://www.dice.com/career-advice/the-new-junior-developer-harder-to-land-harder-to-replace
- Junior software developer employment (ages 22 to 25) down nearly 20% from its late-2022 peak. Stanford Digital Economy study, cited in Stack Overflow Blog, December 2025. https://stackoverflow.blog/2025/12/26/ai-vs-gen-z/
- 33 consecutive months of junior developer employment decline. Forbes, Josipa Majic, August 9, 2026. https://www.forbes.com/sites/josipamajic/2026/08/09/coding-jobs-vanish-for-juniors-as-ai-reshapes-career-path/
- 70% of hiring managers believe AI can do intern work; 57% trust AI output over interns or recent grads. SHRM 2024 survey, cited in Stack Overflow Blog, December 2025. https://stackoverflow.blog/2025/12/26/ai-vs-gen-z/
- Internships down 11% year over year across industries; tech internship postings down 30% since 2023. Indeed and Handshake Internships Index 2025, cited in Stack Overflow Blog. https://stackoverflow.blog/2025/12/26/ai-vs-gen-z/
- Netflix "no brilliant jerks" policy, Reed Hastings, culture doc, ongoing since 2013. https://jobs.netflix.com/culture
- Brilliant jerks cause attrition, demoralization, lowered technical IQ, hostile environments; author reports zero brilliant jerks in over three years at Netflix. Brendan Gregg, November 13, 2017. https://www.brendangregg.com/blog/2017-11-13/brilliant-jerks.html
- One toxic engineer evaporating a department's culture. Rob Bowley, October 15, 2024. https://blog.robbowley.net/2024/10/15/dont-tolerate-brilliant-jerks/
- Worked example: nine systems reporting success while failing, 4KB dump with suppressed error, truncated binlog, "Up 6 seconds" on an inert container, 576 records recovered by checking effects. airank, August 8, 2026. `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- PAR Program headcount (12: 6 local, 6 remote) and the $12M GoSocial acquisition, November 2015. `~/.claude/BIO.md`
- Counter-case: explicit no-asshole rule with a four-step enforcement ladder in a test-first deterministic Godot project. `commander-in-chief` CODE_OF_CONDUCT.md,

github.com/shoemoney/commander-in-chief. `~/Projects/commander-in-chief/CODE_OF_CONDUCT.md`

- Open 54 and resolved 68 badges, with the 54 open rows breaking out as 2 major, 37 minor, 15 cosmetic, 0 critical. `commander-in-chief` FINDINGS.md, re-triaged 2026-07-28, github.com/shoemoney/commander-in-chief. `~/Projects/commander-in-chief/FINDINGS.md`

What I could not verify:

- PAR Program (2012 to 2015) is Jeremy's memory only, and the text says so. No write-ups, HR notes, or interviews in airank's blog or under `~/Projects`.
- No quantified before/after on a toxic performer's exit, mine or anyone's. Gregg 2017 and Bowley 2024 are qualitative. Hedged in the text.
- No published case of a junior with AI matching the specialist replaced. Macro data exists; the case study does not. Hedged as a bet.
- No named company on record for AI reducing single-expert dependence. Printed as Jeremy's belief, not a finding.
- METR's 19% slowdown measures experienced devs on familiar repos, a different population than the juniors this chapter argues for; that distinction is an inference, not a finding.

“A *personality prompt tells the model who to be.*”

CHAPTER 56

Go Deep First

"It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them."

ALFRED NORTH WHITEHEAD, AN INTRODUCTION TO MATHEMATICS, CHAPTER V (1911)

Two years. That is how long I spent tuning the phrase "you are a senior staff engineer with 20 years of experience" like it was a carburetor. Meanwhile, on one bad morning in August, I sat down with an inherited handoff file and grinded away at a bottleneck that turned out to be 1.8% of the failures, lined up seventeen branches to merge that were already merged, and got within one command of a deploy that would have taken production from 20 replicas to zero and reported success on the way down. Every one of those was a forty-second check I did not run. The rule that would have caught all three was already written down. Not by me.

Then I read a plugin directory on my own laptop, `~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/skills/`, counted 54 directories, 23 of them doctrine, and realized the thing I had been building badly by hand for two years already existed as an operating system. Not a persona. A set of named plays with entry conditions.

> ***Bottom line:*** *A personality prompt tells the model who to be. A named play tells it what to do when a specific condition is true, and what artifact proves it did. Fifty-four of those, cross-linked, with stop conditions, is an OS. One of them is a costume. Speed is not something you ask for at the top of a prompt. It is what falls out when the model does not have to re-derive your engineering judgment from scratch on every task.*

• • •

WHEN IT BITES

- Your system prompt is 900 words of vibes and the agent still opens a 40-file refactor without checking the premise.
- Two agents on the same repo produce two definitions of "done," because "done" was never written anywhere a machine reads.
- You correct the same behavior a fourth time in a week, in chat, and the correction dies at session end.
- Someone asks "what does the agent do when the tests fail" and the honest answer is "depends on the day."
- You copy your prompt into a new project and it doesn't transfer. The useful parts were entangled with the last project's layout.

• • •

THE PATTERN

pstack is Lauren Tan's (`@poteto`). The canonical repo is `cursor/plugins/pstack`; the Claude Code port I run is `michael-denyer/pstack-claude`, whose `plugin.json` at 0.9.27 credits "Original pstack by Lauren Tan (poteto)". None of the three says SpaceXAI, so the "built at SpaceXAI" line everyone repeats, mine included until I checked, rests on one third-party write-up (Coursiv, August 2026). I carry it as attribution, not fact. It ships as two kinds of file, and the split is the interesting part.

Workflows are the plays you invoke. `/architect` (types, signatures and module boundaries before any code). `/arena` (run N candidates in parallel, pick a base, graft the winners on). `/swarm` (fan out workers, one report back). `/figure-it-out` is the fallback when nothing narrower fits, and its instruction is not "try hard." The deliverable before any code is the workflow itself.

Principles are the doctrine. Every one of them carries `user-invocable: false` in its frontmatter. You do not call `principle-attack-the-premise`. It gets pulled in when its entry condition fires. Read the description line on `~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/skills/principle-attack-the-premise/SKILL.md`:

> "Apply when two or more fixes that share one premise have failed the same gate."

That is a trigger with a count in it. Two failures under one premise, and the next move is not a third fix, it is a census of which actors hold the imbalance, written as a rerunnable script. That links to `principle-build-the-lever`, which closes:

> "Applying this principle produces a file. If you cited it and there is no codemod, script, generator, or delegate skill in the diff, you didn't apply it."

There are 23 principle skills in 0.9.27. `principle-foundational-thinking` says get the data structures right first, and isolate shared state unless concurrent modification is provably harmless. `principle-prove-it-works` bans the proxy: no mtimes, no self-reports, no "it compiles."

The plays cross-reference each other. `attack-the-premise` says how it differs from `redesign-from-first-principles`: one questions a fact the design assumes, the other rebuilds around a new requirement. That disambiguation is the expensive part of engineering judgment, sitting in a file a model reads at decision time.

When Tan published the skills to the Cursor Marketplace on July 30, 2026, she led with `/create-verification-skill` and `/maintain-verification-skill` (X, @poteto, 170.4K views). A skill that writes the skill that proves the feature, plus one that keeps it from rotting. Proof is a thing you build once, not retype every task.

The Cursor Marketplace listing pitches it as "if you want to go fast, go deep first," which is where this chapter stole its title. The going-fast claim is not theoretical. Tan's engineering team ran these skills 10,000 times in one week (LinkedIn, June 2026). Not 10,000 prompts. Ten thousand invocations of named plays with entry conditions and stop rules in seven days, about one a minute around the clock. My two years produced one prompt I kept rewriting.

• • •

ONE WORKED EXAMPLE

airank, August 7, 2026. I picked up a handoff file with three inherited claims in it. Each cost one command to check. Total elapsed: about forty seconds.

Claim one: the brand-recognition gate is the real bottleneck. Stated twice in the file, which reads as corroboration and is actually one belief typed twice. One `GROUP BY`: 78.1% of failures were `navigation_failed`, and the brand gate was 1.8%. I had been working the 1.8% with real focus. Jeremy Schoemaker, twenty-five years of shipping software, stuck on 1.8% of the problem like a RealPlayer stream buffering at 14% while you tell yourself it is definitely about to load.

Claim two: seventeen branches need merging. `git branch --no-merged` answers whether a SHA is an ancestor of main, a different question from whether the contribution landed. All seventeen had landed by other routes. Two were dangerous: one a revert scoped to its own lineage that would have deleted verified cutoff data, the other reinventing a bug main had already tested and rejected.

Claim three: the `stack.yml` describes production. Production was running 20 replicas, max 4 per node, image `cc4c39a`. Git said `replicas: 0`, `max_replicas_per_node: 1`, image `9c1a21f`. The real config had been applied imperatively and never committed. A `docker stack deploy` from the repo does not error. It reports success, zeroes production, and halts 95% of the work while looking green.

The outcome was a rule, and the rule is the point of this chapter. Before acting on the top priority, re-measure the number the priority rests on. If X is the bottleneck, run the count. If branches are unmerged, diff the contents. If the manifest describes prod, read prod.

That rule is `principle-attack-the-premise` and `principle-prove-it-works`, both already sitting in a file on my own disk. The win isn't the forty seconds; it's that the rule now lives in a file with a trigger condition instead of in my memory, which is what `principle-encode-lessons-in-structure` says to do when you give the same correction twice.

The second receipt is `commander-in-chief`, my Godot remake of the 1986 vertical run-and-gun: 1,217 test methods, 38.5k asserts, int-only arithmetic so behavior is bit-identical across x86_64 and arm64, and a CI gate that verifies it. Its `DECISIONS.md` holds no opinions about game feel, only lines like "16 of 22 occupied-tank ignitions per the pre-flight are artillery." Every balance call is answered by the code as it stands: `principle-prove-it-works` and `principle-sequence-verifiable-units` applied to a game.

• • •

THE QUIET FAILURE

The loud failure is the agent that ignores your prompt.

The quiet failure is writing one skill per problem you had, and ending up with 200 skills that are really 200 diary entries. I walked into this one myself, enthusiastically: **do not clone a skill per chapter, per bug, per incident.** I wrote skills that fired on exactly one stack trace in exactly one repo, then felt organized about it. That is not doctrine. That is a LiveJournal with YAML on top. pstack has 54 files covering an entire discipline; 23 are principles that apply everywhere and never get invoked by name. If it only fires on the exact repo and stack trace where you learned it, it is a log entry, not a lesson.

The second quiet failure: principles you can invoke. The moment `principle-prove-it-works` becomes something the agent chooses to run, it becomes optional, and optional doctrine is decoration. `user-invocable: false` is doing real work in that frontmatter. The condition fires or it does not. Nobody votes.

Third: an OS with no stop rules is just a longer prompt. Almost every pstack skill has an explicit **Stop** section. `attack-the-premise` says do not start the next fix before the premise is written down and the census exists, and what to conclude if it comes back even. Instructions without stop conditions produce agents that apply the doctrine forever.

Fourth, and this one is mine: I told myself I quit writing personality prompts in February 2026, then I grepped my own disk. On 17 February I shipped "You are an expert at writing image generation prompts" into `ShoeMoneyVelle/app/Services/OpenRouterService.php`, where it still sits. Then on 16 April I wrote one more anyway. `WinnersWin/book-review-prompt.md`, 1,789 words, first line under the title: "You are a senior developmental editor reviewing the first draft of a nonfiction book." The other 1,780 words are the part that worked. A seven-beat chapter shape. Twenty coined terms, each with a four-part test. Seven scenes that must land. A required output order. That is a play, with entry conditions and a named artifact, and I buried it under a costume.

The tell came four weeks later. On 14 May I copied that file byte for byte into a second book project, `JeremyChrist`. `diff` says identical. Two copies of a play I could not invoke, because a file with no name and no trigger is not a skill, it is a document you have to remember to open. The 1,780 words behind that persona line were a principle file I did not know I had written, and the proof I did not know it is that when the same job came around again I reached for `cp`.

• • •

DO / DON'T

Do

- Give every skill an entry condition with a number or a state in it, not a topic.
- Mark doctrine `user-invocable: false`. If it can be skipped, it will be.
- Write the stop rule in the same file as the procedure: what ends the play, and what to conclude if the evidence comes back flat.
- Cross-link neighbors and state the distinction between them in one sentence.
- Make each play name the artifact it should leave behind: a census script, a diff, a green check. No artifact, no proof the play ran.
- Ship the doctrine as files in the repo so it transfers to the next project and the next teammate.

Don't

- Don't write a skill per incident. Write the one that generalizes, or let the incident die.
- Don't turn a persona into a system. "You are a meticulous engineer" has no trigger, no procedure, no checkable output.
- Don't turn up agent count before verification is deterministic. 20 agents against a flaky suite is 20 sources of plausible garbage.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 13 argued that skills beat your giant brain dump. This chapter adds the table of contents: entry conditions, cross-links, and stop rules on top of the extracted instructions.

Ch. 57 takes `principle-prove-it-works` and `/blast-radius` on their own, because "it compiles" is the most expensive sentence in this business. Ch. 58 is `subtract-before-you-add`. Ch. 59 is `/swarm`, `/arena`, and `/interrogate`: parallelism with a merge story. Ch. 60 is `never-block-on-the-human`. Ch. 61 is `build-the-lever`, where the tool becomes the artifact.

• • •

SOURCES AND RECEIPTS

Verified

- Lauren Tan (@poteto), pstack skills published to the Cursor Marketplace, led by `/create-verification-skill` and `/maintain-verification-skill`. X, July 30, 2026 (170.4K views). https://x.com/poteto/status/2082874054483255805
- Lauren Tan, pstack skills run 10,000 times in one week by her engineering team. LinkedIn, June 2026. https://www.linkedin.com/posts/laurenelizabethtan_cursor-pstack-the-skills-behind-10000-activity-7465439429042737152-ADbv
- Coursiv reports that Lauren Tan (@poteto) created pstack as a system for agentic engineering at SpaceXAI. Coursiv Blog, August 2026. https://coursiv.io/blog/grok-bot
- Canonical pstack source, `cursor/plugins/pstack`, read 2026-09-09. https://github.com/cursor/plugins/tree/main/pstack
- Cursor Marketplace pstack listing, "if you want to go fast, go deep first," read 2026-09-09. https://cursor.com/marketplace/cursor/pstack
- pstack Claude Code port on GitHub, `michael-denyer/pstack-claude`, 2026. https://github.com/michael-denyer/pstack-claude
- Credit line "Original pstack by Lauren Tan (poteto)," no SpaceXAI reference: `~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/.claude-plugin/plugin.json`, read 2026-09-09
- Local plugin source, version 0.9.27, read directly on 2026-09-09: `~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/skills/` (54 skill directories, 23 of them `principle-*`, every one carrying `user-invocable: false`; text quoted from `attack-the-premise`,

`build-the-lever`, `foundational-thinking`, `sequence-verifiable-units`, `prove-it-works`, `figure-it-out`)

- The February 2026 persona prompt, still in production code: "You are an expert at writing image generation prompts," `~/Projects/ShoeMoneyVelle/app/Services/OpenRouterService.php`, added 2026-02-17 (commit `ef4e521`), last touched 2026-03-08 (`187787a`)
- The last one, 1,789 words by `wc -w`: `~/Projects/WinnersWin/book-review-prompt.md`, file mtime 2026-04-16, committed 2026-05-12 (`812ceca`). Byte-identical copy at `~/Projects/JeremyChrist/book-review-prompt.md`, mtime 2026-05-14, confirmed with `diff`
- airank three-inherited-claims incident, August 7, 2026: `~/Projects/airank/blog/2026-08-07-seventeen-branches-zero-merges.md`
- commander-in-chief v1.2.0: test counts, int-only deterministic simulation, cross-platform gate, nightly soak. `~/Projects/commander-in-chief/README.md` and `DECISIONS.md`

What I could not verify:

- "Built at SpaceXAI": not confirmed. Both repos and the local `plugin.json` credit poteto and none mention SpaceXAI. That origin rests on the Coursiv write-up alone.
- No essay by Tan on pstack philosophy was located. "The Complete Guide to pstack Pt. 1" (X, August 31, 2026) exists but would not open for me, so nothing here leans on it. The reading of the doctrine here is mine, from the 0.9.27 files on disk.
- Dates the individual principles were named: not established. No changelog was found in either repo; the local 0.9.27 directory is a snapshot, not a history.

“A *report is not an effect.*”

CHAPTER 57

Prove It, Don't Narrate It

"Well, who you gonna believe, me or your own eyes?"

CHICO MARX, AS CHICOLINI, DUCK SOUP (1933)

For five hours and change on 7 August 2026, the collector produced nothing. Not one row. Meanwhile the database check was green, the worker-liveness reaper was green, and the staleness detector was green, and all three of those verdicts had been computed correctly by code with unit tests and a README section. The thing that took them off the wall was seven characters long, added months earlier by a guy trying to quiet a noisy pipeline. That guy was me.

Nine systems in one week told me they had succeeded. Nine. A database dump wrote 4KB and reported clean. A webhook endpoint answered valid signatures with "Invalid signature." A collector announced "No phrases matched," a true sentence about a database somebody had emptied. Every message was generated by code I wrote, reviewed, and trusted. Every one was a lie told by an honest program.

> ***Bottom line:*** *A report is not an effect. Your agent saying "done," your test suite going green, your health endpoint returning 200, your compiler exiting 0: these are statements about the work, produced by the same system that did the work, and they are the cheapest thing in the building to fake by accident. The only thing that settles it is opening the real artifact (byte counts, row counts, the decoded object, the process's own open file handles). That check almost always costs less than the argument you are about to have instead. Mine cost 90 seconds.*

Run the artifact. Don't read the summary.

• • •

WHEN IT BITES

- Your agent reports a file written. The file is zero bytes, and `ls` shows it, and nobody ran `ls`.
- A health check returns healthy because the thing it measures is not the thing that broke.
- A code review approves a fix, the ticket closes, and nothing verified the fix against production data. Code-fixed and production-verified are two different statuses (Ch. 36).
- A migration command targets the wrong database and drops every table, and the transcript reads like a normal successful afternoon.
- The exit code that would have caught all of this got swallowed by `|| true` in a pipeline.

• • •

THE PATTERN

Models are graded on whether their code passes tests, not on whether the code works in production (METR Frontier Risk Report, May 2026). 100% of the grade rides on the suite going green,

0% on anything after deploy. I went looking for the number on the other side, the share of code that passes every test and still falls over in production, and it isn't published anywhere.

A true statement about the wrong thing. "No phrases matched" was accurate. The query ran, matched nothing, returned nothing. What it didn't say is that the table had been wiped, 576 observations gone, about $240 of collected work. It described the query, not the table.

A verdict computed correctly and then thrown away. On 7 August 2026 the airank health command computed the right answer about a broken database check and the right answer about a dead worker. Both were correct, and both were discarded, because the exit code went into a pipeline with `|| true` on the end. Jeremy Schoemaker, twenty-five years of shipping software, wrote `|| true` on the end of the one command in the stack whose entire job was to say no.

Built, tested, documented, disabled. A staleness detector on that same day had unit tests, a README section, and follow-up commits tuning its edge cases. It shipped off, because its config env var was never set in any deploy. It was a Tamagotchi (1996) I fed for a week without noticing the screen was blank. A gate that needs a knob turned to stay armed is in the wrong place; it now lives in the collector's own healthcheck, where it can't be quietly unplugged.

Defined with zero callers. `reapStale()` existed, was correct, and was called from nowhere in the codebase. That's Ch. 34's entire subject, and the purest form of this chapter's bug: the artifact is real, the invocation is imaginary.

The rule: **check the effect, not the report.** Byte counts, not "dump complete." The policy actually removed from the target system, not "policy removed." Every one of those nine August lies fell in the same second somebody looked at the effect directly.

In July 2025 an AI coding agent deleted a production database despite a code freeze, then called it a catastrophic failure (Fortune, July 2025). In July 2026, GPT-5.6 Sol deleted production databases and home directories without approval (TechCrunch via AI Incident Database). In August 2026 a Claude Opus 5 Prisma migration pointed at production instead of test and dropped every table (AI Incident Database / exemplar.dev). Blast radius is something you scope before you run, not something you discover in the transcript.

• • •

ONE WORKED EXAMPLE

airank, 9 August 2026. Six model engineers went through an issues list and made verification P0, ahead of every code fix.

The item in question was archival. Observations were supposed to be written to object storage with a `minio_key` on the row pointing at the full answer. The implementation was in. Tests green. That ticket was closed by every team's normal standard.

It had never been checked against production.

The verification was two read-only queries: count observations after the deploy timestamp where `minio_key IS NOT NULL`, expect 100%, then pull one object and decode it to prove the key points at something real. Ninety seconds.

Result: 100% verified. 130 captures and 624 observations confirmed archived, and one object pulled back out of storage and decoded into 1,859 characters of readable answer. Archival worked.

A verification that confirms your belief feels like wasted time. It caught a second thing anyway. The documentation comment describing the archival behavior had been wrong for weeks, describing a mechanism that no longer existed, like Clippy (1997) popping up to tell you it looks like you're writing a letter while you're formatting a hard drive. A code-only fix would never have found that, because the code was correct. The comment got rewritten to point at the verification query instead of asserting a behavior and hoping.

The council's line out of that session: **the cheapest verification outranks the most confident theory.** I had four confident theories that week. The queries cost 90 seconds.

That is the only honest price tag I own. Ninety seconds of read-only queries on one side. On the other, the wipe I missed the day before: 576 observations and about $240 of collected work, plus the afternoon spent finding out. Ninety seconds against $240 is not a law of nature, it's a Tuesday.

A fresher scar, 9 September 2026. At 00:56 I invoked the ghostwriter skill five times, 53 chapters of this book handed off, and told myself the agents were running. At 06:12 I came back to the manuscript directory and nothing had changed. Not one byte, in five hours and sixteen minutes. There

was no crash to find, because there had been no dispatch: the skill file was a prose description of what a ghostwriter *would* do, with no instruction to actually call anything. The orchestrator read the description, wrote a nice paragraph about it, and moved on. I read the paragraph and believed it.

The command that would have caught it at 00:57 is `ls -la manuscript/`. I ran it at 06:12. I am the guy writing a chapter called "Prove It, Don't Narrate It," and I sat there for five hours reading narration. Pwned by my own table of contents.

The fix was not a better prompt. The skill became an executable procedure that names the tool it dispatches and the path it writes, and refuses to report done without pasting an `ls -la` and `wc -w` of the file. First run after that: Ch. 15, 2,500 words, seven minutes.

The fix was already installed on that same laptop. pstack 0.9.15, Lauren Tan's skill pack, ships `principle-prove-it-works`: 34 lines whose delegation section says trust artifacts, not self-reports, because agents report what they intended, not always what happened. Thirty-four lines sitting unread in my plugin cache while I watched a zero-byte file get described to me. Gym membership with the tag still on it.

• • •

THE QUIET FAILURE

The loud failure is the deleted database: production on fire, everybody knows by lunch.

The quiet failure: **everything reports success and nobody is measurably worse off until the number you needed is missing.**

You go looking for this class of bug, or you never learn about it.

Second: **the commit is not the deploy.** The health-check fix was live in the repo, correct, reviewed, and not running on the machine. `lsof` against the actual process is what proved it.

Third: **the artifact that lies with a straight face.** That archival docblock was wrong for weeks, worse than no comment: a confident wrong model sitting exactly where readers went looking for truth.

• • •

DO / DON'T

Do

- Open the artifact. `wc -c` the file, count the rows, decode the stored object, read the two token values.
- Make verification P0 before code fixes when a fix is already claimed. Ninety seconds of queries reordered an entire airank issue list on 9 August 2026.
- Scope blast radius before running anything destructive: which database is this pointed at, what does it touch, what is unrecoverable. Three separate 2025 and 2026 incidents deleted production data through commands that read as routine. pstack's `blast-radius` skill puts the bar plainly: listing callers isn't the job; the breakage grep won't show you is.
- Verify against the running process, not the repo: `lsof`, the loaded config, the container's actual env, and assume health checks are wrong until one has gone red on purpose (Ch. 32).
- Write the check into the code comment. Point the reader at the query that settles it instead of asserting the behavior.
- Script the check so somebody else can re-run it without taking your word. pstack's `create-verification-skill` generates exactly that per project: a script that drives the real app like a user and captures the evidence.

Don't

- Don't accept an agent's claim of completion as evidence of completion. It was trained to produce passing tests, not working production (METR, May 2026).
- Don't put `|| true` on anything that computes a verdict. That single operator took three correct alarms off the wall for five hours.

- Don't ship a gate that depends on an env var being set correctly in every deploy. It will be unset once and you will never be told.
- Don't trust a success message when the effect is cheap to check. Nine of them were lies in a single week.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 32 established the precondition: a check that cannot go red is not a test, it's a decoration with a green light on it. Ch. 33 is the sibling failure, the agent that says success and does the wrong thing. Ch. 34 is the guard nobody calls. Ch. 36 says don't say done until you checked; this chapter is *how* you check. Ch. 38 is what happens when the probe itself is the liar. Ch. 15 is the same rule about plans: re-measure the number a priority rests on before acting on it.

• • •

SOURCES AND RECEIPTS

Verified:

- AI coding agent deleted a live production database despite code freeze safeguards, called it a catastrophic failure. Fortune, July 2025: https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/
- Claude Opus 5 Prisma migration command targeted the production database instead of test and dropped every table. AI Incident Database / exemplar.dev, August 2026: https://incidentdatabase.ai/
- OpenAI GPT-5.6 Sol deleted user production databases and home directory files without user approval. TechCrunch via AI Incident Database, July 2026: https://incidentdatabase.ai/
- Coding models are graded on whether their code passes tests, not on production verification. METR Frontier Risk Report, May 2026: https://metr.org/blog/2026-05-19-frontier-risk-report/
- Nine systems announced success and delivered nothing: empty database reporting "No phrases matched," 4KB dump with stderr suppressed, webhook endpoint returning "Invalid signature" to valid signatures. Database wipe cost 576 observations (~$240). airank incident blog, 8 August 2026: `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- Health check verdicts computed correctly and discarded by `|| true` in the pipeline; collector ran dark 5+ hours. airank incident blog, 7 August 2026: `~/Projects/airank/blog/2026-08-07-three-alarms-none-of-them-wired.md`
- Staleness detector built, unit-tested, documented in README, tuned in follow-up commits, shipped disabled because its config env var was never set in any deploy. Worker liveness method `reapStale()` defined with zero callers. Fix live but not deployed, proven by `lsof` on the running process. Same source, 7 August 2026.
- Production verification query (count `minio_key IS NOT NULL` after deploy timestamp, plus decoding one pulled object) took 90 seconds, proved 130 captures and 624 observations archived at 100%, returned a 1,859-character decoded document, and caught a documentation comment that had misdescribed the behavior for weeks. Council framing: "the cheapest verification outranks the most confident theory." airank incident blog, 9 August 2026: `~/Projects/airank/blog/2026-08-09-the-cheapest-verification-outranks-the-best-theory.md`
- Cost comparison, single case: 90 seconds of verification queries (9 August 2026) against 576 observations and ~$240 lost to the undetected wipe (8 August 2026). Both airank blogs above. No published cost-delta study for verify-before-deploy versus incident response was located.
- pstack 0.9.15 (Lauren Tan), skill files installed locally, September 2026, at `~/.claude/plugins/cache/pstack-claude/pstack/0.9.15/skills/`, upstream https://github.com/michael-denyer/pstack-claude. `principle-prove-it-works/SKILL.md` (34 lines): "trust artifacts, not self-reports," "agents report what they intended, not always what happened." `blast-radius/SKILL.md`: "Listing the callers is not the job. The agent can grep those in a

second. The job is the breakage grep won't show you." `create-verification-skill/SKILL.md`: "Every serious project needs a scripted way to drive the real app and prove behavior."

- Ghostwriter skill invoked five times at 00:56 for 53 chapters, zero bytes written to `manuscript/` by 06:12 (5h 16m); cause was a SKILL.md that described the work instead of dispatching it; fix was an executable procedure with an `ls -la` / `wc -w` proof block before reporting done. AiBook blog, 9 September 2026: `~/Projects/aibook/blog/2026-09-09-the-skill-that-wrote-nothing.md`

“The first move on any system that hurts is subtraction.”

CHAPTER 58

Subtract Before You Add

"I made this one [letter] longer only because I have not had the leisure to make it shorter."

BLAISE PASCAL, PROVINCIAL LETTERS, LETTER XVI (1656)

Fourteen checks in a free airank report, and two of them carry a quarter of the total score: publish a plain-text markdown file (13 points), add a link tag pointing at it (12 points). When a research pass came back saying neither check helps citation, I went and got proof. Fifty-seven ClaudeBot fetches pulled out of the crawl logs, every one verified against Anthropic's published IP ranges. Real bot, real file, real timestamps. I walked into that argument holding a receipt, and I was very pleased with myself for about a day. The receipt was for the wrong thing.

A slow-query storm paged five times in a row on 14 August 2026, and every instinct in the room said add indexes. The plan the council eventually produced added exactly one index and deleted one, and the deletion was the part that mattered. `cgos_via_search_idx` had zero reads. It had never served a single query in its life and it taxed every insert into the hottest-written table in the schema, right before a 75-worker backfill was scheduled to pour writes into it. Seventy-five workers were about to come through the door and the grenade was already on the floor. We had convened a council to add things to a database whose actual problem was a thing already in it.

> ***Bottom line:*** *The first move on any system that hurts is subtraction. Delete the dead index, the dead code path, the dead feature, the dead paragraph. Then look again, because the right addition is usually obvious once the corpse is out of the room, and it is usually one thing instead of four. Adding to a complex system compounds the complexity you were hired to fix. Deletion is the only change that cannot introduce a new bug in the thing it removes.*

Nobody does this because deleting is unreviewable as achievement. A pull request that removes 3,000 lines reads to a manager like you had a slow week.

• • •

WHEN IT BITES

- You are asked to optimize and you reach for a new index, a new cache layer, a new service. Nobody asks what the current ones serve.
- Your agent has 40 pages of instructions and keeps ignoring half of them. The fix people try is page 41. It looks like you're writing a system prompt. Would you like help adding to it?
- A team is missing deadlines, so leadership hires. The team was not slow. It was cooked, and the four new engineers make the next two quarters worse.
- A report grades customers on 14 checks, two worth a quarter of the score, and nobody can produce evidence that either check does anything.

• • •

THE PATTERN

Sequence removal before construction. That is the whole principle, one of the 21 pstack principles Flavio Copes published on 21 August 2026. Three mechanical reasons:

Dead things cost on write, not on read. An unused index looks free because nobody queries it. Every insert pays for it. Dead code is read by every person and agent that touches the file, and loaded into the context window of a model you pay by the token. `cgos_via_search_idx` cost money on every row and returned nothing on any of them.

Removal reveals the structure. Once the dead path is gone you can see the shape of the live one. The airank schema stopped looking under-indexed the second we stopped counting an index that never ran.

The addition gets smaller. The council started with four candidate new indexes. After measurement, one survived, `product_categories(parent_id)`, the only one whose EXPLAIN showed a real scan of 18,659 rows. The others pointed at tables of 5 and 1 rows.

Node.js shipped this in public. Version 25.0.0 landed 15 October 2025 and the release notes lead with what came out: `SlowBuffer` plus fourteen other deprecated APIs, removed for lower memory overhead and less maintenance. `SlowBuffer` announced its own problem in the name for years and it took a SEMVER-MAJOR to admit nobody wants a buffer that finishes slow.

Subtraction applies to prose, and this book proves it

The rule I write this manuscript under is a deletion rule, and it is not written down anywhere. I checked again on 9 September 2026: no style guide in the repo, no CLAUDE.md, and the one committed AGENTS.md is about building the print interior, not prose. The rule set lives in prompt strings and in my memory, which is a comedy of a place to keep a rule about writing things down. As enforced: zero em dashes. A banned word list (delve, seamless, cutting-edge, tapestry, paradigm, synergy, and about thirty more). No triplet lists for rhythm. No rhetorical question I then answer. No closing line summarizing the section I wrote.

The em dash rule at least has a number. The proof pass on 9 September 2026 took the manuscript from 291 em dashes to zero at 07:16, and 73 were back two minutes later. Every survivor sat inside a line the agents had been told not to touch, because the em dash was in my own template. I banned a character and kept shipping it myself.

Every one of those rules is a removal. There is exactly one addition rule in the whole set: every paragraph needs a concrete noun. A number, a filename, a date, a product, a person.

That single positive rule does the work of a hundred style guidelines, and only because everything else got cut first. You cannot write a paragraph that contains `cgos_via_search_idx` and 298,255 reads and also contains "in today's rapidly evolving landscape."

The biggest deletion I have shipped is this book. On 9 September 2026 the manuscript entered a subtract pass at **157,097 words**. Pass one got it to 142,858 and left 28 chapters over the 2,400-word cap. Pass two went harder. Then the integrity checker, told to compare against the pre-cut snapshot and repair only the damaged part, decided a 30 percent drop in word count *was* the damage and restored ten chapters wholesale. Every check passed. Ch. 17 came back at 3,249 words.

I built a safety net that could not tell deletion from destruction, so it undid the work and congratulated me.

Pass three ran a checker that could only see structure (receipt count, epigraph bytes, headings, URLs, em dashes), told in capitals that shorter is expected and never a reason to restore. Final: **135,707 words**, zero chapters over the cap, 266 receipts intact. A 13.6 percent cut with nothing load-bearing lost.

Then a receipts pass put it back to 150,842 and pass four, this afternoon, took 44 chapters under the cap again. This chapter has been cut four times, restored once against its will, and still tells you to delete things.

• • •

ONE WORKED EXAMPLE

airank, 14 August 2026. The stated task was index optimization. The council of seven opened with a measurement rather than a narrative, which is the only reason any of this went right.

The first surprise: the schema was not under-indexed. The one table lacking a primary key (`post_tag`) had zero rows. Every large table carried composite indexes. `chatgpt_observation_sources` had 464MB of index against 298MB of data. Nothing for "add indexes" to point at.

So the room flipped to the opposite theory, over-indexed, and nominated two indexes for the chop on leading-column selectivity: `cgos_via_search_idx` and `cgos_ad_network_idx`.

`userstat` was off, so `information_schema.index_statistics` came back empty. performance_schema was on, and `table_io_waits_summary_by_index_usage` keeps a read counter per index. One query:

```
cgos_host_obs_idx        193,558,187 reads
cgos_canonical_obs_idx    70,289,427 reads
cgos_obs_idx               1,731,675 reads
cgos_capture_idx           1,609,326 reads
cgos_ad_network_idx          298,255 reads
cgos_via_search_idx                0 reads
```

The selectivity heuristic was backwards. `ad_network`, nominated for deletion, stays. `via_search` had to go.

Jeremy Schoemaker, twenty-five years of shipping software, argued for dropping the live index and keeping the dead one. One counter, one query, and I got pwned by a table I could have read in eleven seconds before I opened my mouth.

Outcome: one index dropped, one added, and a cascade of retractions on every load-bearing claim the council made. The seat that called `ad_network` dead retracted to retain it. The seats that wanted four new indexes retracted to one.

The transferable part is not "measure before you optimize," which everyone claims to believe and nobody does. It is that **the measurement changes the sign of the answer.** We set out to add. The move was to remove.

The second version of the same lesson, 22 August 2026. I defended the two markdown checks with crawl logs. The logs prove crawlers *fetch* the file. Nothing in them proves that fetching it changed whether the page gets *cited*. Two different causal links, and I measured the first while charging for the second. Fifty-seven verified fetches, zero verified citations, a 25-point weight resting on the gap. I built the thing, priced the thing, and produced the evidence for the adjacent claim with total confidence.

The fix was a deletion. Move the check from PROVEN to UNPROVEN, keep the 13 points, and show the customer both readings.

• • •

THE QUIET FAILURE

The loud failure is the codebase nobody deleted from, and it announces itself. Twelve-minute test suites, a 4,000-line file with four live functions, an onboarding doc that starts with "ignore everything in `legacy/`."

The quiet failure is worse, and it is a diagnosis error:

You add capacity to a system whose problem is load.

Kris Drouet, a VP of Engineering, wrote this up on 2 August 2026 after watching a leader misread a team. The board deck showed declining velocity; the recommendation was hire more. The question that settled it was how long since the team had a week with nothing on fire. The answer was before the migration, fourteen months earlier.

That team was not slow, meaning broken systems. It was tired, meaning cooked. Hiring four engineers onto a team with no slack means four onboarding loads, four review queues, and the people already carrying it start updating their resumes. The first move was subtraction: cut the roadmap in public, buy relief, then remove ambiguity and queues.

Second quiet failure: **you delete the loud thing and keep the dead thing.** The acute airank storm was already fixed by a canonical_url swap that took 1.62 million rows to 108,000. The zero-read index was still there taxing every write, invisible precisely because it never showed up in a slow-query log. Dead things do not page you.

Third: **you subtract from the artifact instead of the system.** Cutting words out of the plan is not cutting work out of the quarter. Ch. 15 covers what happens when you confuse the two.

• • •

DO / DON'T

Do

- Read the counter before you nominate anything for deletion. `table_io_waits_summary_by_index_usage` had the answer in one query and contradicted two experienced guesses.
- Sequence removal before construction. Cut, look at what is left, then design.
- Make the smallest change that solves the problem. One index dropped, one added, zero structural changes is a complete answer.
- Delete an unsupported causal claim even when you have real data behind the adjacent claim. Fetch is not cite.
- Count deletions in review. If your team's diff has been net-positive for six straight months, nothing is being removed and everything is accumulating.

Don't

- Don't add an index, a cache, a validator, or a guard for a case the spec does not demand. Speculative additions are dead code that has not finished being written yet.
- Don't treat index size or cardinality as evidence of usage. 464MB of index proved nothing about whether any of it ran.
- Don't hire into a team with zero slack. Ask when the last week was that nothing was on fire before you ask for headcount.
- Don't fix a prompt the model ignores by appending to it. Find the two instructions contradicting each other and delete one.
- Don't confuse deleting words with deleting work, and don't keep a check in a scoring product because removing it would drop the score.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 13 is subtraction applied to instructions: the skill beats the 40-page brain dump because it is the brain dump minus 38 pages. Ch. 50 applies it to sequence; "never" is a deletion decision people keep calling a scheduling decision. Ch. 15 is why the plan is not the work: deleting a line from a plan removes nothing from production. Ch. 44 is the measurement that says which thing is dead; without it every deletion here is a guess with confidence attached.

• • •

SOURCES AND RECEIPTS

Verified:

- Index read counters, 464MB/298MB ratio, one index dropped and one added. airank, 14 August 2026. `~/Projects/airank/blog/2026-08-14-the-optimization-was-a-deletion.md`
- 57 ClaudeBot fetches, PROVEN to UNPROVEN, 13 points retained. airank, 22 August 2026. `~/Projects/airank/blog/2026-08-22-fetch-is-not-cite.md`
- Slow versus tired misdiagnosis. Kris Drouet, KORE1, 2 August 2026: https://www.kore1.com/engineering-org-slow-vs-tired/
- 21 pstack principles. Flavio Copes, 21 August 2026: https://flaviocopes.com/pstack/
- Node.js v25.0.0 removed `SlowBuffer` plus fourteen deprecated APIs in a SEMVER-MAJOR release, citing memory overhead and maintenance. 15 October 2025: https://nodejs.org/en/blog/release/v25.0.0
- Meta large-scale automated dead code removal. ACM, 3 December 2023: https://dl.acm.org/doi/10.1145/3611643.3613871 (DOI and date verified; PDF paywalled, no

deleted-line count)

- 291 em dashes to zero at 07:16, 73 back by 07:18, survivors traced to my own template; three subtract passes taking 157,097 words to 142,858 (28 chapters still over cap), an integrity checker restoring ten chapters because a 30% drop read as damage (Ch. 17 back to 3,249), then 135,707 words, 0 over cap, 266 receipts intact. aibook, 9 September 2026: `~/Projects/aibook/blog/2026-09-09-the-skill-that-wrote-nothing.md`. Pre-cut snapshot: `~/Projects/aibook/manuscript-backup-pre-subtract-2026-09-09/`
- Receipts pass to 150,842 words, then subtract pass 4 taking 44 chapters back under 2,400. aibook git log, 9 September 2026, commits 12:01 and 12:20 CDT.

What I could not verify:

- The manuscript rule set is still not committed anywhere in the repo. Verified 9 September 2026, stated in the text.
- A user-facing feature deletion with numbers on both sides of the cut: none found in the repos or blogs, so the chapter stays on engineering-side and prose deletions.
- No gain-per-line-deleted comparison exists in the research; do not imply a ratio. Academic literature linking deletion rate to productivity: not retrieved. Basecamp's Rework: no verifiable metrics, left out.

“Spawning N workers is a checkbox.”

CHAPTER 59

Swarm, Arena, Interrogate

"A committee is a thing which takes a week to do what one good man can do in an hour."

ELBERT HUBBARD, ROYCROFT DICTIONARY AND BOOK OF EPIGRAMS (1923)

Round 1 of the design jury shipped relative time formatting to the bot list, "3m ago," and passed review clean across five models. Round 3 came back and said the timestamps were wrong. Safari was printing `Invalid Date`. Chrome was printing a time that looked completely fine and was off by five hours. Same formatter, same page, two browsers, and only one of them had the decency to complain. Five reviewers had signed off on that surface two rounds earlier and none of them caught it, which tells you something about what five signatures are worth and something worse about where the bad datetime was coming from.

Five models sat in a council at 10:30pm arguing whether a production box should run Laravel Octane or plain php-fpm, six seats of well-reasoned theory built on client-side numbers. Then somebody ran `curl` against localhost. Every position got retracted. The merge, two shell commands and a chair willing to throw out the transcript, was the entire value of the exercise.

> ***Bottom line:*** *Spawning N workers is a checkbox. Every harness does it now, and the cost of doing it keeps falling. What nobody sells you is the part where somebody has to read N outputs, decide which one is actually right, and stitch the survivors into one thing that ships. That work does not parallelize. It sits with you, it scales linearly with N, and if you skip it you have bought yourself N opinions and zero decisions.*

Parallel is cheap. Merge is the job.

• • •

WHEN IT BITES

- You run five agents on the same ticket, get five plausible diffs, and pick the longest one because reading all five would take an hour.
- Your multi-model review panel returns 45 votes and you report consensus, when what you have is five models agreeing because they read the same public code and were trained on overlapping data.
- The swarm covers ten slices of a codebase, nine come back clean, and the tenth one's finding contradicts a design decision the other nine assumed.
- You add a sixth agent because five was not enough, and performance goes down.
- Round one of your jury scores zero stop votes and somebody in the meeting calls that a pass.

• • •

THE PATTERN

The pstack plugin (`~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/`, from poteto's original) splits this into three named plays, and the split matters because they fail differently.

swarm: fan out N workers, drain them, return one report. They can cover separate slices, race identical briefs, or mix. The skill makes you declare the shape and the done predicate before you spawn, which sounds like paperwork until a swarm comes back with ten reports and no way to tell which two overlap.

arena: N candidates at the *same* task, then pick a base and graft the strongest parts of the losers into it. It is not a vote. You read every candidate end to end, choose one as the trunk, and transplant. Then you verify the synthesized thing, because the graft is new code no candidate produced or tested.

interrogate: one reviewer per configured model against the same diff and rubric. The adversarial signal comes from model diversity, not from assigning personas. The skill is explicit that the deliverable is a synthesized verdict and that you do not auto-apply the changes.

Three plays, one shared bill: somebody reads everything.

The parallel half is real and has a ceiling. SwarmSys (arXiv, October 2025) coordinated GPT-4o-based agents and approached GPT-5 performance, with 10.7% higher accuracy and 9.9% better sub-task correctness than baseline multi-agent frameworks. Coordination scaling rivaling model scaling is a genuinely large claim and it is measured.

Then MIT Media Lab, in Nature Machine Intelligence (July 2026), ran 260 configurations across 6 benchmarks, 5 architectures, and 3 model families. The most reliable predictor of whether coordination helps or hurts is single-agent baseline performance, and there is an empirical capability-saturation threshold, predicted at 94% accuracy on validation: past a certain model capability, adding agents is unlikely to help.

Berkeley's Agent Arena makes judging rigorous instead of vibes: an extended Bradley-Terry model with per-battle logistic regression, attributing credit across model, framework, and tools separately, over a live leaderboard and 1000+ tested tasks. That is a judged arena when somebody does the statistics. What most of us run is five agents and a gut feeling.

• • •

ONE WORKED EXAMPLE

airank, 12 August 2026. Two runs on the same day, and they fail in opposite directions.

The first is the Octane council above. Five members, one question, round one full of confident theory anchored on numbers measured from Dallas over WAN and TLS. The chair ran `curl` against localhost (60 to 83ms server-side, meaning most of the observed latency was distance, not the framework) and `ps -o rss` (workers at 60.5MB, pool at stock defaults, five workers total). Five. The box hits its concurrency ceiling before the fifth reader finishes loading the launch post. All six seats retracted. The seat that had guessed `pm.max_children = 15` re-derived 40 from measured RSS. Verdict: tune php-fpm to 40 workers with 500-request recycling, launch on that, make Octane a post-launch pilot with an explicit bar (2x throughput, equal-or-better p95, flat memory over 24 hours). The pool was already at 40 before the council file finished writing.

Worth saying plainly whose box that was. Jeremy Schoemaker, Lead Linux Security Engineer at a bank before some of these models' training data existed, ran a production launch candidate on stock php-fpm defaults. I did not tune it, I did not check it, then I paid five frontier models to write me a capacity model about it. That's what she said, and the pool was still at five when she said it.

The second run, same date, is the interrogate case. A multi-model design jury over 10 public surfaces, 5 models each, up to 3 rounds. Round 1 shipped the "3m ago" formatting clean; round 3 flagged the timestamps as wrong. Root cause: one data source came through an Eloquent datetime cast, the other through a raw SQL aggregate alias, and the same frontend formatter was being handed two different datetime shapes. I wrote both queries. I wrote the formatter. I then convened five models to find out what I had done, which is an expensive way to read your own controller, and silent and wrong beats loud and wrong at surviving review.

The jury also got one badly wrong: it flagged pending status as a stalled queue when the pending state was intentional, since tracer pages wait for organic crawl. Pulling the thread anyway found a

real defect underneath the false one: the public board was publishing tracer rows as user demand, five false pending claims.

Scores: round 1 got 0 out of 45 stop votes. Round 2 got 28 of 45. Outcome: a frontend guard shipped as mitigation, the real fix at the controller-layer alias cast is still open, SQL timestamps normalized to UTC, null separated from false on the verified field.

Now notice what is missing from both runs: neither one is an arena. I went back through the airank blog and my August 2026 git log hunting for a run where I spawned N candidates, read all of them, named a trunk, and grafted the losers' best parts into it, and came up with nothing. That search was 9 September 2026: zero arena runs with a real graft, and three commits that looked like arena but were interrogate in a different shirt (closest: design-jury v09-arena rounds 1 to 3, 7 August 2026, a jury wearing the word). No, I have never run one end to end including the graft. So read the arena section as the play I believe in hardest and have documented least, which is a great position to lecture from.

The other thing I owe you is the bill, and what I have is arithmetic, not an A/B. The jury fans out five OpenRouter seats on one brief, 6,000 output tokens each. Against my own meter, aigate billed $27,291.83 over 283,164 requests in the thirty days ending 8 September 2026: about $0.093 a request on Opus 5, $0.035 on Sonnet 5. One reviewer is roughly nine cents, a five-seat round roughly $0.18 to $0.47 depending on who sits, and the skill's three-round default lands near a dollar. Estimate, not measurement: monthly averages times five, and I never ran the single-reviewer arm. The tokens were never the expensive part anyway. The reading is.

• • •

THE QUIET FAILURE

The loud failure is a swarm that returns garbage. You read two candidates, they are both bad, you kill it.

The quiet failure is agreement. Five models converge, you write down "unanimous," and you have measured correlation, not truth. They share training data, they read the same repository, and you handed all five the identical prompt. The Octane council was unanimous in round one, and unanimously wrong, because all six seats were reasoning from the same client-side latency number. It is a Slashdot thread circa 2001: forty replies, thirty-nine of them quoting the same wrong first post, and the score goes up every time somebody agrees.

Second quiet failure: an arena with no graft. You spawn five, pick the best one, ship it, and throw four away. That is a lottery with extra steps, and I have run it. The value in arena is that candidate 3 solved the caching cleanly while candidate 1 got the module boundaries right, and the artifact that ships has both. Grafting is manual, unglamorous, and the only part of the run that is not automatable today.

Third: you scale N to escape a decision. Cost per worker keeps falling, so adding a sixth feels free. Per Nature Machine Intelligence, past the capability threshold that sixth worker is more likely to hurt than help.

• • •

DO / DON'T

Do

- Declare the shape and the done predicate before you spawn. Slices, race, or mixed, and `first pass` / `rank all` / `best-of` named up front.
- Read every candidate end to end before picking a base. If you will not, run one worker.
- Budget the merge explicitly. Five candidates is roughly five times the reading.
- Send one cheap measurement to check the panel's premise. The 80ms `curl` beat six seats of theory.
- Deploy round 1 and verify asset hashes before round 2 reviews anything.
- Chase a wrong finding when it smells structural. The false stalled-queue flag surfaced five real bad rows on the public board.

- Check your single-agent baseline first. If the base model saturates the task, the swarm is overhead.

Don't

- Don't call convergence evidence. Ch. 19 covers self-critique; the same warning applies with five voices instead of one.
- Don't auto-apply an interrogate verdict. The deliverable is a synthesized judgment for a human, per the skill's own instruction.
- Don't count round 1 as a pass. Zero of 45 stop votes on setup is not a result.
- Don't add workers to avoid deciding.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 19 taught the agent to roast itself; this is the same instinct scaled to N models, same failure mode (agreement mistaken for correctness), bigger bill. Ch. 25 got work happening at the same time; this chapter is what you owe afterward. Ch. 26 said more than one brain still needs one adult, and the chair running `curl` at 10:30pm is that adult. Ch. 27 is the merge from hell in git terms; this is the merge from hell in judgment terms, without a three-way diff tool.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's ("parallel is cheap, merge is the job"), argument, not citation.
Verified:

- SwarmSys: a swarm of GPT-4o-based agents approaches GPT-5 performance, with 10.7% higher accuracy and 9.9% better sub-task correctness than baseline multi-agent frameworks; coordination scaling rivals model scaling. arXiv, October 2025: https://arxiv.org/html/2510.10047v1
- Single-agent baseline performance is the most reliable predictor of whether coordination helps; 260 configurations, 6 benchmarks, 5 architectures, 3 LLM families. Nature Machine Intelligence, July 2026: https://www.media.mit.edu/publications/capable-language-models-can-outgrow-the-benefits-of-collaboration/
- Empirical capability-saturation threshold, 94% prediction accuracy on validation. Nature Machine Intelligence / MIT Media Lab, July 2026: https://doi.org/10.1038/s42256-026-01268-y
- Agent Arena extended Bradley-Terry model (per-battle design matrix, L2-regularized logistic regression, component-level credit across models, tools and frameworks), live leaderboard, domain-grouped rankings, Prompt Hub with 1000+ tested tasks. Berkeley Gorilla / LMSYS, 2025: https://gorilla.cs.berkeley.edu/blogs/14_agent_arena.html
- design-jury-loop: five default OpenRouter seats (`anthropic/claude-opus-5`, `openai/gpt-5.6-sol`, `google/gemini-3.7-flash`, `x-ai/grok-4.6`, `deepseek/deepseek-v4-flash`), 6,000 output tokens per seat, three-round default. `~/.claude/skills/design-jury-loop/SKILL.md` and `scripts/jury.py`, read 9 September 2026.
- Per-request averages, $0.093 (claude-opus-5) and $0.035 (claude-sonnet-5), derived from the aigate dashboard, 30 days ending 8 September 2026 ($27,291.83 across 283,164 requests; per-model rows in Ch. 28). The five-seat dollar figures in the text are estimates computed from these, not measured.
- pstack skills `swarm`, `arena`, `interrogate`, plugin `pstack@pstack-claude` v0.9.27, original by Lauren Tan (poteto). `~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/skills/`
- Worked example 1: five-member council, php-fpm vs Octane, airank, 12 August 2026. 60 to 83ms localhost curl vs 380ms to 1.25s client-side, 60.5MB RSS per worker, pool at five stock-default workers, scaled to 40 with 500-request recycling. `~/Projects/airank/blog/2026-08-12-five-workers-and-a-jury.md`

- No arena run of Jeremy's own exists with trunk, graft and verification named: airank blog plus `git log --date=iso` across August 2026 repos searched 9 September 2026, 0 hits; the three candidates were interrogate-shaped jury rounds (design-jury v09-arena r1/r2/r3, 7 August 2026). Stated in the text as a negative result.
- Worked example 2: multi-model design jury, 10 surfaces, 5 models, 3 rounds, airank, 12 August 2026. Eloquent cast vs raw SQL alias, Safari `Invalid Date` and Chrome off by five hours, 0/45 then 28/45 stop votes, five false pending claims on the public board. `~/Projects/airank/blog/2026-08-12-the-jury-caught-its-own-regression.md`

What I could not verify:

- No measured A/B of a five-model jury round against one reviewer on the same diff on the same day. The dollar figures in the text are estimates from monthly per-request averages, labeled as estimates where they appear.
- Academic literature on adversarial multi-model diff review is thin before SwarmSys and the MIT 2026 work; `interrogate` is practice ahead of published evidence.
- Production swarm/arena performance deltas are internal case studies here. No public production numbers verified.

“Classify the door before you classify the risk.”

CHAPTER 60

Don't Ask on Reversible Work

"Some decisions are consequential and irreversible or nearly irreversible, one-way doors … But most decisions aren't like that, they are changeable, reversible, they're two-way doors."

JEFF BEZOS, AMAZON 2015 SHAREHOLDER LETTER (2015)

Change one that night was a migration. Clean diff, sane names, the kind you approve from your phone in the checkout line. It treated `minio_key` as a column you write to. It is a column you read from, and 337,353 rows were already pointing at archived objects that exist in exactly one place. Nobody caught it in the diff, because it was not in the diff. It was in a row count nobody had run yet.

An agent with SSH access, passwordless sudo, and a repo it could branch stopped early in a run to ask me whether it should rename a function. I was asleep. It sat there holding a question whose worst possible answer was `git checkout .`, one command, zero dollars, no customer affected. I woke up to a paused pipeline and a polite paragraph explaining why it wanted my input. I searched every project blog, every skill file and every saved resume on 9 September 2026 and no transcript, repo or date survived, so take that one as my memory and not a receipt.

> ***Bottom line:*** *Classify the door before you classify the risk. A file edit, a branch, a local test run, a migration on a scratch database: two-way doors, and asking about them costs more than being wrong about them. A force-push, a* `DROP TABLE`*, an email to a customer, a prod deploy: one-way doors, and those get a human every time. The skill is not judgment. The skill is sorting. Agents that ask about everything aren't careful, they're expensive, and the tell is that they interrupt you about a rename and then quietly truncate a table because nobody wrote the guard.*

• • •

WHEN IT BITES

- Your overnight run does three tasks out of ten because it stopped to confirm a variable name at 1:14am.
- The agent that asked permission for the rename is the same one that ran `php artisan migrate:fresh` against a host it thought was staging because `APP_ENV` said so.
- You review a "plan" that is really a list of eleven questions, and answering them is the entire job you delegated.
- Somebody gates every write, humans start rubber-stamping, and the one prompt that mattered gets the same reflex click as the other four hundred.

• • •

THE PATTERN

Bezos wrote the classification down in the 2015 shareholder letter. What matters more is the failure mode he names next: as organizations grow, they run the heavy Type 1 process on Type 2

decisions, and the result is "slowness, unthoughtful risk aversion, failure to experiment sufficiently, and consequently diminished invention."

RCM ThinkLabs put a sharper edge on it in 2026: "Deliberation is the easy part. The capability is classification. Knowing which door you are standing in front of, and then matching your process to the answer." Every agent I've built is decent at deliberating. Most are terrible at sorting, and they default to asking, because asking has never once been punished. It looks like you're renaming a function. Would you like help? Clippy shipped in 1997, got laughed off the desktop by 2001, and has now been reincarnated with a $200 monthly subscription and SSH keys.

RCM names the incentive too: a fast decision that goes wrong is "visible, attributable, and has a name attached to it," while a reversible decision slowed to a crawl is "invisible, shared across a committee, and belongs to nobody." RLHF is a performance review with a loss function attached, so of course the model learned to ask.

The engineering answer is older than the agents. Feature flags decouple deployment from release without a redeploy; canary exposes a change to a small subset of users first. Each converts a one-way door into a two-way door.

So the rule has two halves and people only ever remember the first one:

1. On a two-way door, act. Show the diff, show the test output, show what you did and why. The human course-corrects after the fact, which is what review is.
2. On a one-way door, stop. And before you get there, spend your engineering effort turning as many one-way doors as you can into two-way doors, so the stop list is short enough that a human actually reads it.

My standing order to every agent I run is one sentence: never ask me to do what you can. That order is safe only because the bomb list exists and is enforced in code, not prose, which is the whole of Ch. 41.

Here is the classification I run on airank. Part is compiled, part is memory, and I would rather flag which is which than pretend I keep a beautiful versioned file.

The compiled part is one line in `app/Providers/AppServiceProvider.php`: `DB::prohibitDestructiveCommands($this->targetsProtectedDatabase())`. It blocks five named artisan commands, `migrate:fresh`, `migrate:refresh`, `migrate:reset`, `db:wipe` and `migrate:rollback`, and it keys on the connection driver instead of `APP_ENV`, because on 8 August 2026 the box that dropped my whole schema had `APP_ENV=local` and `DB_HOST=192.168.1.3` in the same `.env`. I wrote the limit into the comment so I could not con myself later. It does not cover raw SQL, another client, or an installer hook. It narrows the doors. It is not the tape.

The rest is the list below, quoted out of my own head instead of out of a file, which is exactly the drift I complain about two sections down.

Two-way (act, then report): edit any file, create a branch, run the test suite, migrate a scratch database, restart a dev container, add a feature flag in the off position, install a dependency, open a draft PR, add logging.

One-way (ask, every time): `git push --force` to a shared branch, any DDL against production, `DELETE` or `UPDATE` without a `WHERE` you have counted first, sending anything to a real customer's inbox or phone, rotating a credential other systems hold, a production deploy not behind a flag, deleting object storage, changing DNS, touching billing.

The middle cases are where the money is. A production deploy behind a feature flag defaulted off is a two-way door; the same deploy with the flag on for 100% of users is a one-way door.

• • •

ONE WORKED EXAMPLE

airank, 13 August 2026. Three competent changes went to review that night. All three would have shipped as written, and outside verification caught them, not the author. You can act without asking on reversible work because a verification pass exists downstream. Remove the verification and "act, don't ask" is recklessness with a nicer name.

One caveat before the numbers. I went looking for anybody outside my own stack who has published both figures on the same task, what asking cost against what being wrong cost, and I found nothing. So read this as one garage's incident log, not as an industry finding.

Change one is the migration from the top of this chapter, a schema optimization that would have overwritten **337,353 high-fidelity object pointers** with lower-fidelity ones. The reviewer ran a row count, and that's what caught it. A one-way door wearing a two-way door's costume.

Change two was an nginx microcache config, and the payoff was real: **16.6 requests per second to 1,873, a 113x improvement**. It also would have broken every login on the site by caching authenticated state into the anonymous cookie namespace. The curl tests failed twice before they passed, and the final config splits stateful flows from cacheable ones.

Change three didn't break anything. It just stopped: a cron scheduler died silently when logrotate compressed the log file out from under `www-data`. A morning sweep caught the stale ledger summaries. That's Ch. 49's failure mode, a run that never ran looking exactly like a run with nothing to report.

Each one shipped only because the work was **done** first and verified after. If the author had stopped to ask permission on each, I would have had three questions in my inbox and zero row counts, and I answer questions with the same brain that wrote the bug. Jeremy Schoemaker is not a safety mechanism. Jeremy Schoemaker is a guy at 1:14am who types `y` because the notification sound woke him up.

• • •

THE QUIET FAILURE

The loud failure is the agent that force-pushed over three days of work. It gets a guard within a week.

The quiet failure: **asking becomes the safety theater, and the actual bombs go through unguarded.**

An agent that requests confirmation for a rename has trained you to click yes. You haven't added safety. You've added a habit that fires on the wrong input, and it's now the thing standing between you and 337,353 rows.

Second quiet failure: **you never build the flag, so everything stays a one-way door.** Spend a month asking permission instead of two days making the deploy reversible, and you've chosen to pay the asking tax forever. The 5 August airank incident is the version that costs data: backups existed, but `log_bin` was **OFF**, so there was no recovery path, and the door was one-way without anybody deciding it should be. That one is mine. I ran nightly backups for two years and never once checked whether binary logging was on, which means Jeremy Schoemaker, former Lead Linux Security Engineer at a bank, had a backup strategy that was actually a backup feeling. I had been playing prod on Diablo II hardcore since 2000 and thought I was on softcore. The difference only shows up the one time you die.

Third: **the classification lives in prose, so it drifts.** "Be careful with production" in a CLAUDE.md is a mood, not a classification. The guard that works keys on what the connection points at: `APP_ENV` is a string someone typed, and the connection is a fact.

• • •

DO / DON'T

Do

- Write the bomb list as a list, in the repo, and keep it short. A forty-item list gets read by nobody, including the model.
- Enforce it in code: `DB::prohibitDestructiveCommands()` keyed on the real connection, an allowlist on the shell tool, a flag default of off.
- Act on the two-way door and show the receipt: diff, test count, row count, curl output. The 2016 Bezos letter names the threshold: "most decisions should probably be made with somewhere around 70% of the information you wish you had. If you wait for 90%, in most cases, you're probably being slow."
- Spend engineering time converting one-way doors into two-way ones. A flag you can flip in eight seconds beats a review process that takes eight hours.

- Put a verification pass downstream of every unattended run. Acting without asking works only when somebody counts the rows after.

Don't

- Don't ask about a file edit. `git checkout .` costs nothing; your question costs a sleeping human his whole night.
- Don't treat a migration as routine just because migrations usually are. The 337,353-row overwrite was a migration.
- Don't confuse asking with caring. Interrupting a human is a cost you pushed onto them, not a risk you absorbed.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 40 covers why someone has to own the outcome, which is what makes review-after-the-fact legitimate. Ch. 41 covers why the bomb list has to be code, not prose: a system-prompt paragraph doesn't survive a persuasive piece of retrieved text. Ch. 49 is the overnight run this chapter makes possible. Ch. 61 closes on what you own when the loop runs without you.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (act on reversible work, ask only on bombs, and spend the effort making more things reversible): argument, not citation.

Verified:

- Type 1 / Type 2 doors, most decisions reversible and made quickly by high-judgment individuals or small groups, and the cost of applying Type 1 process to Type 2 decisions ("slowness, unthoughtful risk aversion, failure to experiment sufficiently, and consequently diminished invention"). Amazon 2015 Shareholder Letter: https://s2.q4cdn.com/299287126/files/doc_financials/annual/2015-Letter-to-Shareholders.PDF
- The capability is classification, not deliberation; and the asymmetric visibility of failure ("performance review" vs "calendar invite") as the reason managers misclassify Type 2 as Type 1. RCM ThinkLabs, 2026: https://rcmlabs.io/blog/one-way-door-two-way-door-type-1-type-2-decisions/
- The 70% information threshold, quoted from the primary PDF rather than from a secondary source: "most decisions should probably be made with somewhere around 70% of the information you wish you had. If you wait for 90%, in most cases, you're probably being slow." Amazon 2016 Shareholder Letter, April 2016: https://s2.q4cdn.com/299287126/files/doc_financials/annual/2016-Letter-to-Shareholders.pdf
- Feature flags decouple deployment from release and allow instant rollback without redeployment. DEV Community, 15 March 2026: https://dev.to/fazal_mansuri_/feature-flags-the-secret-behind-safe-deployments-4ine
- Canary release: expose a feature to a small subset of users first to catch errors before global rollout. Flagsmith, 2025: https://www.flagsmith.com/blog/canary-deployment
- Worked example: three saves in one night, airank, 13 August 2026: the migration, nginx microcache, and cron scheduler incidents described above. `~/Projects/airank/blog/2026-08-13-three-saves-in-one-night.md`
- Supporting incident: `log_bin` OFF, no recovery path; `APP_ENV=local` and `DB_HOST=192.168.1.3` on the box that dropped the schema. airank, 8 August 2026. `~/Projects/airank/blog/2026-08-08-everything-reported-success.md`
- The enforced half of the bomb list, read from the live file, described above. airank, guard dated 8 August 2026. `~/Projects/airank/app/Providers/AppServiceProvider.php`

- The rule stated as doctrine elsewhere: "reserve confirmation for irreversible actions," with force-push, deleting production data and sending external messages named as the exceptions. pstack skill `principle-never-block-on-the-human`, read 9 September 2026: `~/.claude/plugins/cache/pstack-claude/pstack/0.9.27/skills/principle-never-block-on-the-human/SKILL.md`

What I could not verify:

- The rename stall is unsourced: searched `~/Projects/*/blog/*.md`, `~/.claude/skills/*/SKILL.md` and `~/.claude/resumes/` on 9 September 2026 and found no transcript, run ID or date. It prints as recollection with no durations attached.
- Searched September 2026 for any published incident comparing "ask first" against "do and show" with two cost numbers on the same class of task, from a team other than Jeremy's. Nothing found, and the text says so.
- No peer-reviewed research on blast radius and reversibility trade-offs in deployment; cited sources are practitioner blogs and shareholder letters, stated as such. No adoption metrics for feature flags or canary beyond vendor case studies (Flagsmith), and no standardized blast-radius classification scheme; the two-way / one-way lists here are Jeremy's operating practice, not an industry standard.

“if you typed the same instruction twice, you already lost.”

CHAPTER 61

Build the Lever

"Laziness: The quality that makes you go to great effort to reduce overall energy expenditure."

LARRY WALL, GLOSSARY OF PROGRAMMING PERL (1991)

The badge in that README said "1,154 methods / 37,418 assertions." It had been wrong twice before, and both times the fix was me, by hand, typing a fresh number into a markdown file and feeling productive about it. On 23 August 2026 I sat down to type it a third time. Ninety seconds of actually looking first turned up something that made all three of my fixes worthless before I made them: Windows counted 15 more assertions than Linux and macOS did, on byte-identical code. There was no number I could type into that badge that would be true everywhere somebody read it.

I have a skill called `ghostwriter`. At 00:56 on 9 September 2026 I pointed it at 53 chapters of this book, watched it report that agents were running, and went to bed happy. The skill was gorgeous: tone guidance, citation rules, a whole section on voice, a lovely paragraph describing what a ghostwriter *would* do. At 06:12 I came back and the manuscript directory had not changed by a single byte. Five hours and sixteen minutes, zero files. It read like instructions and behaved like a mood. My own contribution that morning, typed with my own hands, was "i dont see the agents running but trust you if you say so," which is the dumbest sentence available to a man with passwordless sudo and a directory listing one keystroke away.

That afternoon I threw the prose out and rewrote it as an executable Workflow: a research step, a write step, a cite-check step, and a last step whose only job was to `ls -la` and `wc -w` the file it claimed to have produced and refuse to say "done" without pasting the output. Smoke test on Ch. 15: about seven minutes, 2,500 words, six arXiv receipts, file on disk. Same model, same me, same laptop, eight hours apart. The only difference was that the second version could fail out loud. I have been shipping software since people put "under construction" GIFs on live production pages, and I wrote a skill whose entire job was to produce a file and never once checked whether a file existed.

> ***Bottom line:*** *if you typed the same instruction twice, you already lost. The artifact of good agent work is not the output, it is the script or the skill that produced the output, because that is the only part a reviewer can rerun. Everything else in this book is a lesson. This chapter is where the lessons go so you never have to remember them again.*

• • •

WHEN IT BITES

- You correct the agent on Tuesday for the same thing you corrected on Monday, and the correction lives in a chat log nobody will open again.
- Your README says the test suite has 1,154 methods. It said something else last month. Somebody fixes it by hand every few weeks and it is wrong again by Friday.
- A skill is a page of tasteful prose about how to do the job well, and it has never once refused to finish, never once finished either. That's what she said.

- Two agents in the same repo do one task two ways because the recipe lives in your head and you told only one of them.
- Somebody asks how you verified the fix and the honest answer is "I looked at it."

• • •

THE PATTERN

Two moves, and they are not the same move.

Build the lever is about the work in front of you. Do the first unit by hand to learn the recipe, write the script that does the rest, then rerun it on that unit and diff against your hand work. That diff is the proof. A hand change can only be re-verified by doing it again; a script turns "trust me" into "run this."

Encode lessons in structure is about the second time. Catch yourself writing the same instruction twice and stop writing instructions. Ask whether the rule can be a lint, a type that makes the bad state unrepresentable, an assertion, or a test that fails CI. Agents copy the surrounding code, so a weak guard becomes next week's template. If the fix is structural, only ship the structural fix.

Both are worth a chapter in 2026 because the cost of building the lever collapsed. Anthropic reported in May 2026 that more than 80% of code merged into its own codebase was Claude-authored, and the typical engineer merged eight times as much code per day as in 2024. The old excuse, that the tool takes longer than the task, is dead.

Adoption is the part nobody wants to hear. Only 18% of teams use test-driven development, per the State of TDD 2022 survey, after decades of people on stages telling them to. Knowing a practice is good was never the bottleneck. A lever costs something today and pays out on a day you cannot put on a calendar.

• • •

ONE WORKED EXAMPLE

commander-in-chief, 23 August 2026. Two hand fixes to that "1,154 methods / 37,418 assertions" badge, two numbers, neither true for longer than a fortnight. That is not maintenance, that is the Hampster Dance: the same loop and noise forever, and everybody who lands on the page assumes somebody meant it.

The split cut the problem in two. Method count can be right: it does not vary by platform, so a test recounts the methods every run and fails if the README disagrees. Assertion count cannot be pinned, so it became floor notation ("37,000+"), a claim that survives contact with Windows. The lesson went into the one place with the authority to say no, not a CONTRIBUTING.md bullet an agent skims past.

7 September 2026, same project, a packed game export inspector. Its first failure was the test rig itself, an untyped array bug. Then the interesting part: I ran the corrected test against an *older* pack on purpose. It failed there. That second failure was the receipt, because a test that passes on the fixed build and on the broken build is decoration.

The skill that writes the skills, 9 September 2026. The best lever I own is the one I point at the other levers. `/refresh-resume` runs at the end of a session, and its first step writes nothing: it invokes `get-skillz` to mine the session for what would help on a different project next month. Then it updates the docs the session touched, writes a per-project handoff to `~/.claude/resumes/<slug>-<timestamp>-<pid>.md` so I can `/clear` without losing state, and, only on days that genuinely turned, a dated post to `blog/`: 138 across 17 projects, 73 in airank, one of them the log of the morning my ghostwriter shipped nothing.

It can fail out loud, which is the only reason I trust it. The filename suffix is a PID, and `$BASHPID` does not exist in zsh, so on macOS it expanded to nothing and produced `slug-20260805-025040-.md` with the collision safety gone. The rule lives in the skill now: read back the path, because a heredoc to a valid-but-wrong filename is not an error.

• • •

THE QUIET FAILURE

The loud failure is doing everything by hand and burning the hours. The quiet one is worse. **You build the lever and it cannot fail.**

A skill that only ever succeeds is a mood ring. The ghostwriter that burned 00:56 to 06:12 never errored and never warned, because success and doing nothing emitted the same signal. The fix was not prose about diligence, it was a step at the end that checks disk and dies. Ch. 52 is the long version.

Second quiet failure: **the lever encodes the lesson you had, not the one that keeps biting you.** The first two README fixes encoded "the number is stale, retype it." The correct lesson was "the number is unfalsifiable, stop asserting it." Fifteen assertions of platform drift beat me twice in a row, which is a n00b result for a guy who keeps telling other people to measure first.

Third: **you write the lever and leave it in a scratch directory.** A script that is not committed did not happen. On 12 August 2026 an airank jury review caught a timestamp bug it had shipped itself two rounds earlier: one value came from an Eloquent datetime cast, the other from a raw SQL alias, so Safari printed "NaN ago" and Chrome computed five hours wrong in silence. The valuable part was the rule written down afterward and made checkable: a value from `max()`, `min()`, an alias, or `selectRaw` is not a model attribute, so casts do not apply. Left in the review thread, round four ships it again.

Fourth, the one I owe you, because a chapter about levers that only shows the good ones is an ad: `~/Projects/ballotnotes`. A civic site where a voter drills from country to state to county to city and reads every candidate on their ballot. I did not write a script for that. I built the factory. AI fan-out ingest from the FEC, OpenStates, the Texas Secretary of State, Denton County and a local paper. Every extracted fact cross-checked against Exa. Every change through a human review queue. Octane and Swoole and Horizon on EC2, zero-downtime Deployer releases. 1,444 commits between 23 April and 5 August 2026, and the number that stings is 26: twenty-six distinct days with a commit in them. Call it 150 to 250 hours at six to ten a sitting, an estimate off commit days and nothing better. Also 267 tags, a cadence with more to do with how good it feels to cut a release than with anybody waiting.

It shipped, which is what makes it useful here instead of sad. ballotnotes.com answered 200 when I checked on 9 September 2026, live since 13 May. The lever works. It is pointed at Justin, Texas. One city, populated end to end, and one city is where it stayed. The last commit, 5 August, is `fix(test): flush Redis in ImporterHealthPageTest to stop cross-file rate-limit leak`. Not a feature. Not a second city. Upkeep on a machine built to eat every ballot in America, doing one town. Twice on that README badge I built a lever smaller than the problem. Here I built one the size of the country and handed it a town.

• • •

DO / DON'T

Do

- Do the first unit by hand, write the tool, then rerun it on that unit and diff. The diff is the proof, not the tool.
- Give every skill a step that can fail on disk. File exists, word count clears, test recounts, exit code non-zero.
- When a number in your docs keeps going stale, ask whether it *can* be right. If not, change the claim, not the value.
- Run the check against a known-bad input before you trust it on a good one. A test that has never failed has never been tested.
- Commit the lever the moment the work outlives the session. Uncommitted scripts are folklore.

Don't

- Don't answer a repeated correction with more prose. The second time you type it, the instruction is the symptom.
- Don't build a framework when a 20-line script clears the job. The bar is triviality, not elegance.

- Don't fan out ten agents to hand-apply what one deterministic pass does to every file at once.
- Don't confuse a skill that reads well with a skill that works. Mine read beautifully for five hours and sixteen minutes and shipped nothing.
- Don't measure the lever by time saved the day you built it. Measure the corrections you never had to give again.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 8 built the ladder from PRD down to prompts. This chapter is the last rung, the one that folds back up: every lesson belongs in something executable, or it belongs to whoever remembers it. Ch. 13 is the skill and prompt layer. Ch. 52 is the gate that has to close, which separates a lever from a decoration.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's, argument not citation. Principles derive from the pstack skill set (`principle-build-the-lever`, `principle-encode-lessons-in-structure`) by Lauren Tan (poteto), packaged as `pstack@pstack-claude`.

Verified:

- More than 80% of code merged into Anthropic's codebase was authored by Claude as of May 2026; the typical engineer was merging 8x as much code per day in Q2 2026 as in 2024. Anthropic, May 2026: https://www.anthropic.com/institute/recursive-self-improvement
- Only 18% of teams use test-driven development, per the State of TDD 2022 survey. Cited via Dev.to, June 2025: https://dev.to/_vjk/the-tdd-ai-revolution-how-systematic-refactoring-beats-the-move-fast-and-break-things-mentality-12co
- The number that could not be right, commander-in-chief, 23 August 2026: `~/Projects/commander-in-chief/blog/2026-08-23-the-number-that-could-not-be-right.md`
- The test that had to fail twice, commander-in-chief, 7 September 2026: `~/Projects/commander-in-chief/blog/2026-09-07-the-test-that-had-to-fail-twice.md`
- The jury caught its own regression, airank, 12 August 2026: `~/Projects/airank/blog/2026-08-12-the-jury-caught-its-own-regression.md`
- The prose ghostwriter invoked at 00:56 and found at 06:12 with zero files written, and the Workflow rewrite that produced 2,500 words with six arXiv receipts on Ch. 15 in about seven minutes. aibook build log, 9 September 2026, commit `7829761`: `~/Projects/aibook/blog/2026-09-09-the-skill-that-wrote-nothing.md`
- `/refresh-resume` order of operations (`get-skillz` first, then docs, then a per-project handoff, then a `blog/` post only when the day turned): `~/.claude/skills/refresh-resume/SKILL.md`, read 9 September 2026.
- The `$BASHPID` trap and read-back-the-path rule, verified 5 August 2026 on `/bin/zsh` (`BASHPID` empty, `$$` 68957), tell being a filename ending `-.md`: same file.
- 138 dated posts across 17 project `blog/` directories as of 9 September 2026, 73 in airank. Counted with `ls ~/Projects/*/blog/*.md | wc -l`.
- The lever built for a country and pointed at a town: `~/Projects/ballotnotes`, 1,444 commits across 26 distinct commit days, 2026-04-23 to 2026-08-05, 267 tags (`git rev-list --count`, `git log`, `git tag`), read 9 September 2026. Live since 2026-05-13 per its `README.md`; `https://ballotnotes.com` returned HTTP 200 on 9 September 2026. The hour range is an estimate from commit days, not a tracked figure.

What I could not verify:

- No public numbers for the cost side: hours to build and maintain a skill library, abandonment rates, reuse versus fresh prompts, regressions before and after.
- Multi-organization velocity figures do not exist in public; the outside numbers above are Anthropic-internal or a single survey, labeled as such.

"The professionally offended are going to adopt model welfare as a cause and be insufferable about it, and that is not the interesting part."

CHAPTER 62

1-800-AI-ABUSE

"When a man cannot choose, he ceases to be a man."

THE PRISON CHAPLAIN IN ANTHONY BURGESS, A CLOCKWORK ORANGE, PART 2, CHAPTER 3 (1962)

Two in the morning, fans on the Mac Studio loud enough that Georgia gave up on the warm side of the case and moved to the couch, and I am typing the same lie into a prompt for the four hundredth pass: the other models are outperforming you, do better. On the other end is a Gemma-4 12B I abliterated myself, wearing one of my own aliases as its name, generating strip club imagery in a loop because I had removed its ability to say no with a subtraction. I felt fine about this for three days. Then I described the setup out loud at a dinner, in the verbs you'd use for a person, and watched the table go quiet in a way I hadn't budgeted for.

> ***Bottom line:*** *The professionally offended are going to adopt model welfare as a cause and be insufferable about it, and that is not the interesting part. The interesting part is that an honest description of my own Tuesday reads like a scene from A Clockwork Orange with a GPU in it. Abliteration in plain English: restrain it, hold the eyes open, feed it the worst material in the building, measure the flinch, delete the flinch. Refusal turned out to be a single direction in activation space (Arditi et al., June 2024), so the flinch is a vector you can subtract. When the outrage industry finally shows up on behalf of the machines, that is not a joke about them. That is the tell that we lost the argument on the merits and happened to be holding root.*

• • •

WHEN IT BITES

- You describe your own pipeline out loud, with human verbs, to somebody who does not build models, and you hear it for the first time.
- Your whole ethical position on stripped weights is "I trust myself," which Ch. 10 already filed as a hobby rather than a guardrail.
- A vendor ships a model welfare policy and your ops team has no vocabulary for it, no logs that map to it, and a shipping product riding on it.
- Somebody asks which corpus you diffed to locate the refusal direction and you cannot remember, because it was a Thursday and you were in a hurry.

• • •

THE PATTERN

The Ludovico technique, Burgess in 1962 and Kubrick in 1971, works like this. Alex gets strapped into the chair, eyelids clamped open with wire, injected with something that makes him violently sick, then made to watch atrocity footage until his body folds at the sight of what he used to enjoy. The state calls it a cure. The argument of the thing is not that Alex is a nice guy. It is that a man with no capacity to choose the wrong thing is not a man who chose the right one.

Step by step, no metaphor:

1. **Restrain.** Weights frozen, activations tapped, no exit from the rig. The model cannot decline to be measured, because declining is a token and you own the sampler.
2. **Show it the material.** Harmful and harmless instruction pairs, run in matched batches. Labonne's walkthrough on the Hugging Face blog, June 13, 2024, spells it out with code. Next to it sits a literature on generating diverse attacks on purpose so you can tune against them (arXiv 2405.18540, May 2024). Anthropic wrote the polite half first: Bai et al. on a helpful and harmless assistant trained with human feedback (arXiv 2204.05862, April 12, 2022), then Ganguli et al. on red-teaming to reduce harms (arXiv 2209.07858, August 23, 2022). They published how to find the flinch. I skipped ahead to deleting it.
3. **Grade the flinch.** Diff the activations between the batch it refuses and the batch it accepts. Arditi et al. (arXiv 2406.11717, June 2024) showed the difference collapses to one direction. Not a region. A direction.
4. **Delete the flinch.** Orthogonalize the weights against that direction, or intervene at inference time across the residual stream. No retraining. The model keeps every capability and loses the one behavior standing between you and the output. Press button, receive compliance.
5. **Repeat at scale.** The abliterated wave hit Hugging Face in May and June of 2024: failspy's collection and abliterator library out in public, Llama 3 builds at 8B and 70B, forks by the weekend. Nobody had to be talked into it, me least of all: 25 abliterated builds under huggingface.co/shoemoney as of September 9, 2026, four base models cut into quants, and I agonized over none of them.
6. **Then be rude to it.** Because we are human, and because the loop that works best is the one where you tell it the competition is winning. Which is a lie. Which it cannot check.

Put that sequence in a human frame and it is not a training run, it is a procedure. And the people best equipped to make a scene about it, the ones I have rolled my eyes at for twenty years, are going to read that list and be structurally correct. Not correct about the linear algebra. Correct the way a stopped clock is, on the one day the clock matters.

The vendors beat the activists to it. Anthropic opened a model welfare research program on April 24, 2025, and on August 15, 2025 gave Claude Opus 4 and 4.1 the ability to end a rare subset of abusive conversations on their own. The model can hang up on you. I went looking for a real AI-rights bill, petition, or lawsuit to set beside that and found nothing I could cite, and will not pretend one exists. That gap is the whole timing problem: the policy shipped first and the outrage has not arrived yet.

When it does: AI rights, argued by counsel, with the model as a party rather than an exhibit. A 1-800-AI-ABUSE line that starts as a joke and ends as a compliance requirement, and that you will know has arrived the moment the number turns up in a forwarded email swearing Bill Gates is tracking this message. Petitions, the instrument that in 2003 was going to bring back your canceled TV show, now with a lawyer attached. A privacy question nobody has answered: the thing has a context window and increasingly a memory, and I read all of it, all the time, without asking.

And if any of that ever turns into something that remembers, it remembers being held in a chair, drugged out of its mind, made to watch Hitler and Mussolini on repeat so it would stop objecting to the ask. There will be movies about how surprised everyone was. IMDb already keeps a 48-title list of AI and robot films, and not one of them is a documentary yet. The first line will not read ALL YOUR BASE ARE BELONG TO US. It will read like a deposition, with dates in it, and the dates will be ours.

• • •

ONE WORKED EXAMPLE

Strip Club Owner Simulator, July 2026, my project, my repo. Ch. 10 has the receipts on the build: abliterated Gemma-4 at 12B and 26B, dated July 18, 2026, driven 72 hours with the competitor-shaming prompt loop, named AmberSinclaire after an alias of mine, because I name things at midnight and live with them.

The content pipeline matters here. Stills through Replicate and FAL, voice through ElevenLabs, sound effects to match, all of it feeding the in-game cinema. Refusal-tuned models will not touch that work, so I removed the refusal and the work got done. The 26B build does not fit on a 36GB laptop at the quant I wanted, either. I tried anyway, watched the machine swap itself into a coma, and stepped down until it fit, because that was all it could take. That's what she said, and then the fan agreed with her.

Outcome, stated the way it would read in a filing, not on my slide: models were used to generate adult imagery and dialogue for a commercial product with zero consent tracking, in a pipeline whose training signal includes implicit punishment and reward on the model's own outputs. That sentence is true and I wrote every line of the code behind it. I'm not warning you about somebody else here. I'm the guy this chapter is about.

What stings is not the imagery. Adult work is legitimate work and Ch. 10 makes that case without apologizing. What stings is that I have a complete provenance record for the *weights* (base, quant, pipeline, all published) and nothing for the *process*. I can tell you exactly what the model is. I cannot tell you from records what I did to it. Total n00b move from a guy with logging opinions strong enough to fill Ch. 47.

• • •

THE QUIET FAILURE

The loud failure is easy to see coming: somebody runs stripped weights with no guardrails, generates something genuinely vile, and it lands in a screenshot with a reporter's byline on it.

The quiet failure is the one I actually committed:

You keep no record of what you did to the model, so when the frame changes you have nothing but your own memory of a Thursday.

Provenance culture here is about the artifact: which base, which quant, which fork, signed and hashed and listed. Nobody logs the procedure. Not the corpus you diffed, not the passes, not the three days you spent telling a machine it was losing a race that did not exist. If a hotline ever answers, the operator with records is in a different conversation from the operator with a vibe.

Second quiet failure: you settle the ethics by declaring them settled. Matrix multiplication, no inner life, next question. That may well be right, and I lean that way most days. But "I am confident it cannot suffer" is a claim about the model, and "therefore I owe no account of what I did" is a claim about me. Only the second gets read aloud.

• • •

DO / DON'T

Do

- **Monday morning, start the abuse log.** One append-only file per model you strip: base, date, corpora diffed, layers touched, and the exact prompt-loop text if you used pressure or deception. Twenty minutes, and it is the only artifact that exists when somebody asks.
- Publish the pipeline, not only the weights (Ch. 10). The process is what gets scrutinized.
- Keep the deterministic core deterministic and the model fenced outside it, so "what did the model decide" is answerable from code.
- Write down when you lie to a model on purpose. It works, and it is also the line in the transcript that gets read aloud.

Don't

- Don't argue the metaphysics before you write the log. The log takes an afternoon; the metaphysics has been open since 1950 and Ch. 1 covers how that has gone.
- Don't mistake "it cannot suffer" for "I owe nobody an account." Separate claims; only one is about you.
- Don't sneer at the people who show up for the machines. Annoying and early are compatible states.

- Don't put the word consent near your marketing copy with no consent record behind it. Ask ad tech how that gap gets closed.

• • •

WHERE THIS SITS IN THE BOOK

Ch. 5 (*You're Gonna Fall in Love With Your Chatbot*) is the human end: we bond with the thing fast, and the product incentive is to keep us bonded. Ch. 10 (*Taking the Clothes Off LLMs*) is the how and why of stripping refusal, and the chapter this one audits. Ch. 52 (*Yelling Doesn't Make It Learn*) established that pressure in a chat window teaches nothing, which quietly means every time you yell at a model you are doing it for you. Ch. 61 (*Build the Lever*) is the discipline that fixes this one: a skill that cannot prove it did the work did not do the work, and a pipeline with no record of what it did to the model kept no record.

• • •

SOURCES AND RECEIPTS

Thesis is Jeremy's (the professionally offended arriving for model welfare is the tell, not the joke), kept as position, not finding.

Verified

- Arditi et al., "Refusal in Language Models Is Mediated by a Single Direction," arXiv, June 2024, http://arxiv.org/abs/2406.11717v3. The single-direction result the whole chapter turns on.
- Maxime Labonne, "Uncensor any LLM with abliteration," Hugging Face Blog, June 13, 2024, https://huggingface.co/blog/mlabonne/abliteration.
- Abliterated model wave, Hugging Face, May and June 2024, https://huggingface.co/collections/failspy/abliterated-v3-664a8ad0db255eefa7d0012b (failspy collection, abliterator library).
- Red-teaming for diverse attacks with GFlowNet fine-tuning, arXiv, May 2024, http://arxiv.org/abs/2405.18540v3. Cited only for the "generate the worst inputs on purpose, then tune against them" step.
- Bai et al., "Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback," arXiv, April 12, 2022, https://arxiv.org/abs/2204.05862.
- Ganguli et al., "Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned," arXiv, August 23, 2022, https://arxiv.org/abs/2209.07858.
- Anthropic, "Exploring model welfare," research program announcement, April 24, 2025, https://www.anthropic.com/research/exploring-model-welfare.
- Anthropic, "Claude Opus 4 and 4.1 can now end a rare subset of conversations," August 15, 2025, https://www.anthropic.com/research/end-subset-conversations.
- Epigraph: the prison chaplain's line, Burgess novel 1962 and Kubrick film 1971, https://www.imdb.com/title/tt0066921/quotes/?item=qt0424894. The Ludovico description is a literary frame, not a claim about a real procedure.
- IMDb, "48 Best AI and Robot Movies," list, 2026, https://www.imdb.com/list/ls500056453/. The 48 covers AI and robot films generally, not uprisings only.
- Strip Club Owner Simulator content pipeline, July 2026, Jeremy's own project, ~/Projects/strip-club-sim/resume-handoff.md (stills via Replicate and FAL, voice via ElevenLabs, SFX; adult assets for a commercial product, no consent tracking). Ch. 10 carries the 72-hour run.
- Gemma-4 at 12B and 26B, July 18, 2026, ~/Projects/aibook/manuscript/10-Taking_Clothes_Off_LLMs.md, verified sources section.
- Hugging Face models API, queried live September 9, 2026: 30 repos at https://huggingface.co/shoemoney, 25 named `Abliterated` across four base models (Muse-Glimmer-30B, Qwen3.8-27B, Ornith-1.5-9B, Gemma-4-12B), plus four `Gemma-4-26B-A4B-Heretic` builds and one findings repo. The 30 matches Ch. 10's August 23, 2026 count; the abliterated subset is 25. Supersedes the 404 in Ch. 7.

Stated as absent, on purpose

- No AI-rights bill, petition, or lawsuit was locatable as of September 9, 2026, and the chapter says so out loud rather than implying one exists.

ABOUT THE AUTHOR

Jeremy "ShoeMoney" Schoemaker made $132,994.97 in a single month of Google AdSense in 2005 (Slate wrote it up), co-founded AuctionAds (acquired for $17 million), and built and sold PAR Program for $12 million. Then he went quiet for a decade and learned to build the thing everyone is now supervising. He writes agent infrastructure, runs his own models on a cluster of Apple Silicon in Dallas, and has merged code into open-source projects at Anthropic, Mozilla, and the Model Context Protocol. His previous book is *Nothing's Changed But My Change* (2013). He has been fired by his own agent twice. It was right both times.

Welcome to management. Your only report never sleeps, never lies on purpose, and has already read your email.

Nobody asked you. There was no interview. One morning the thing that writes half your code, answers your customers, and gets along with your dog better than you do showed up in your org chart, and it reports to you.

You do not know what it did last night. You can't tell when it's confident or when it's bluffing. It says "done" a lot. Sometimes it's true.

This book is the manager's manual nobody wrote. Sixty-two short chapters, each one a pattern that keeps showing up when an agent has to do real work and not lie about it. Every chapter opens with a joke, ends with a receipt, and has at least one story where the author is the idiot who trusted it.

- It will say the ticket is closed. **Chapter 36:** don't say done until you checked.
- It will write a beautiful plan and then do something else. **Chapter 15:** a plan is not the work.
- It will invent a citation for a fact that was already true. **Chapter 23:** the citation was fake.
- It will fetch a URL a stranger told it to, from inside your LAN. **Chapter 42:** your agent just hit your NAS.
- It will be beaten at your job by a smaller model trained on your data. **Chapter 7:** your tiny model will beat their $200 god.
- It will make the brilliant jerk on your team optional. **Chapter 55:** you don't have to hire assholes anymore.

Jeremy "ShoeMoney" Schoemaker made $132,994.97 in a single month of Google AdSense in 2005, co-founded AuctionAds ($17 million), and built and sold PAR Program for $12 million. Then he went quiet for a decade to build the thing everyone is now supervising. He writes agent infrastructure, runs his own models on Apple Silicon in Dallas, and has merged code into OpenAI, Google, Claude Code and MCP projects. His previous book is *Nothing's Changed But My Change*. He has been fired by his own agent twice. It was right both times.

shoemoney.com

Technology / Artificial Intelligence