The models were fine. I was the bug.

Varun Choraria · 2026-08-02 · https://www.varunchoraria.com/the-models-were-fine-i-was-the-bug/

I typed a command and nothing happened. No output, no error, no prompt coming back, just a cursor sitting there while my laptop got warm.

What was actually happening is that my own tool had spawned about 280 copies of itself, each one launching the next.

Everybody writes the post about what their agents did well. Nobody writes this one, which is a shame, because this is where the lessons live. I wrote up what a month of this costs already. Here's the other column.

280 of me

The cause is almost funny. When you build a program, the build tool writes out a file. That file needs a little flag on it that says "you're allowed to run this." Mine wasn't setting it.

Here's the bit worth remembering. A file without that flag isn't treated as broken. It's treated as **invisible**. The shell goes looking for a command by that name, finds a file it isn't allowed to run, and carries on looking as if the file simply weren't there.

So it kept looking, found a small backup script further down the list, and ran that instead. That script's whole job was to say "can't find the real thing, let me go fetch it." Fetching it meant looking up the same name again. Which found the same backup script, which fetched it again, and so on without end.

2 things made this horrible to catch. It never happened to anyone else. When you install software properly, the installer sets that flag for you. It only bit me, on my machine, testing my own build. So it looked random, and random is where you lose days.

And there was nothing to search for. It just hung. If a command ever hangs on you with no output, count the processes before you sit there waiting:

```
ps aux | grep -c "[r]ainmaker"
```

A number that climbs is a loop. That one line would have saved me an afternoon.

What I took from it: an intermittent bug is usually shaped like your machine, not like your code. When something breaks only for you, stop rereading the logic and start listing what's different about you.

The agent that fired me mid-task

10 minutes into a session, I'm walking an agent through an interview about a business. Halfway through, it decides the tool it's running has a flaw, and goes off to rewrite the tool.

25,000 tokens and 10 minutes, and nobody asked it to. The annoying part is that the flaw was real. It wasn't wrong about the code. It was wrong about whose call it was.

I'd written the rule down. It was in the README, in plain English, saying don't do this.

The model doesn't read the README. It reads the file that defines the skill.

What I took from it: a rule written where the model doesn't look isn't a rule. It's a note to yourself.

I moved 5 boundaries into the file it actually loads, and wrote a test for each so they can't quietly rot:

- Don't modify yourself
- Don't edit the user's site while you're diagnosing it
- Don't ship anything on their behalf
- Don't hand-write data a program is supposed to generate
- Don't estimate a number the tool can just compute

That last one is my favourite. Guessing at a number that's one command away is the most human thing on the list.

The bug that made no sound

This is the worst of the 4, and it produced no error at all.

The tool is an orchestrator with 26 smaller skills underneath it. The orchestrator runs the conversation and produces the context. Each small skill refuses to run without that context, deliberately, because a skill that guesses at missing context produces confident nonsense.

I shipped 0.3.1 with the orchestrator in it. The installed plugin was pinned at 0.2.1, which had the 26 skills and no orchestrator.

So every skill did exactly what it was built to do. It checked for context, found none, and politely declined. 26 correct refusals adding up to a product that did nothing.

Nothing crashed and nothing logged. It just sat there doing nothing, very politely.

Once I went looking I found 3 more of the same shape. A setting written into every config file and read by nothing, because the code that picked it only ever looked at the command-line flag. A startup hook that told the assistant to run a shell command, so it ran the command and stopped instead of loading the skill. A pointer file written to one filename when the assistant reads a different one.

Same bug, 4 hats. The design lived in the prose. The plumbing quietly went around it.

What I took from it: the design isn't what you wrote in the doc. It's whatever the delivery layer actually does. Check the hook text, the filenames, the version pins, the config keys. That's usually where the gap is.

My own gate, in my face

This is the smallest one, and the one I'd defend hardest.

I published a post with a new tag on it. The deploy failed.

My publishing pipeline checks every post against an approved list of tags before it'll build. A tag that isn't on the list stops the deploy. I'd forgotten I wrote that check, so for about a minute I was annoyed at myself.

Then I remembered why it's there. Tags with no page behind them. Tags that are near-duplicates of each other. A taxonomy that rots because adding a tag is easier than thinking about one. The check turns a slow invisible mess into a fast obvious failure.

What I took from it: the guardrail you wrote will eventually block you, and that's it working. Deleting it in the moment trades 2 minutes of annoyance today for a mess in 6 months.

Same point applies to agents generally. They'll do exactly what the guardrails allow. Every rule you don't write is a rule you've decided not to have.

So what actually broke

Read the 4 back and it's uncomfortable. None of them were the model being stupid: a permission flag, a boundary in the wrong file, a packaging mistake, and a check working perfectly.

The models were fine, and my plumbing wasn't.

I think that's most of what people mean when they say agents are unreliable. They've built something where the design lives in a prompt and the behaviour lives everywhere else.

So the habit that fixed things wasn't better prompting. It was writing failures down where the machine reads them again.

For that tool it's a file called `FEEDBACK.md`. 15 entries under "Fixed", each titled as the symptom instead of the fix. Next to it sits `SPEC.md`, which says in 5 clauses what "done" means, and every open item gets sorted against it. Anything that doesn't break a clause gets queued instead of argued about. That one rule took my open list from 6 items to 3, not because things got fixed but because things were finally allowed to leave.

On my machine there's a memory folder. One file per fact, summary line at the top, cross-linked. 21 of them. Why a build failed. Why a name is frozen. What a constraint actually was, 6 weeks after everyone stopped remembering.

None of it is clever. It's a diary the machine reads before it starts work.

And then I did it again

While fixing the version pin above, I wrote the setup instructions for my agent into a file called `AGENTS.md`, which is a sensible name that reads well.

The assistant I actually use loads a file called `CLAUDE.md` on startup. It has never opened `AGENTS.md` in its life.

So I sat down, wrote careful instructions, saved them, and shipped a tool that had never once read them.

That is the same mistake as the 2nd one up there. The one I’d already found. Already fixed. Already written a rule about.

Months in, with the rule sitting in my own notes, I did it again with a different filename.

Which is the honest ending. None of this stops you being wrong. It just makes the second time cheaper than the first, and there is always a second time.