

The Claude bill said \$1,492. I paid \$20. Here's how.

Varun Choria · 2026-07-30 · <https://www.varunchoria.com/how-to-get-1492-out-of-a-20-claude-subscription/>

I pay \$20 a month for Claude. Last week I went digging through my own logs to find out what that actually buys, because I wanted to write an honest post about it and I did not want to guess.

The logs are sitting on everyone's machine. Every session your coding agent runs gets written to a file, with the token counts attached. Mine covers one month, from the 2nd of July to the 2nd of August. 72 sessions. Older ones had already rotated out.

Here is what came back, by model.

Model	Written	Cache written	Cache read	If billed per use
Opus 5	3.59M	9.69M	746M	\$523
Sonnet 5	2.34M	10.7M	721M	\$292
Opus 4.8	2.40M	7.70M	313M	\$265
Fable 5	1.81M	4.81M	260M	\$412
Total	10.1M	32.8M	2.04B	\$1,492

One month. \$1,492 of inference at list price, on a \$20 plan.

I had been telling people I get about \$100 of work out of a \$20 subscription. I was off by more than an order of magnitude.

The number that surprised me wasn't the money

Look at the last 2 columns again. 10 million tokens written. 2 billion read.

That ratio is 200 to 1, and it changes what you think you're paying for.

Almost none of this is a model producing text. Almost all of it is a model reading. Every time an agent picks up a task, it re-reads the project: the spec, the design file, the last 20 commits, and the rules I wrote for it 6 weeks ago and forgot about. 2 billion tokens of re-reading against 10 million tokens of output.

Which means the thing you're actually buying is not writing. It's context. And the way to get more out of the same \$20 is to make the context worth re-reading, not to write cleverer prompts.

The whole bill

Since the point of this post is receipts, here is all of it.

- Claude Pro: \$20 a month
- ChatGPT: free tier
- Codex: rides on what I already have. 1.05 billion input tokens since February, 797 million of them in July alone

- Firecrawl: free tier, for bulk scraping
- context.dev: 10,000 credits, gifted by the founder after I told him I was between roles and couldn't afford a plan
- Hosting: GitHub Pages, free
- Domain: about \$10 a year
- Everything else: nothing

That's it. There is no agency, no freelancer, no seat licence, no analytics tool with a sales call attached.

How \$20 stretches

3 things, and none of them are clever.

Plan with the expensive model, execute with cheap ones. I use a frontier model to think through what needs to happen, then hand the actual execution to a swarm of smaller agents. Thinking is where the money is worth spending. Typing is not.

Compress the output. I run [caveman](#), a skill by Julius Brussee that makes every response drop articles, filler, and hedging while keeping every number, error string, and line of code exact. Julius measures it at about 65% fewer output tokens. I went looking for that number in my own logs and couldn't produce an honest version of it, because token counts include code and tool calls, which the skill deliberately leaves alone. So take his figure, not mine. What I can tell you is the effect I actually feel: more work fits inside one session before the context fills up. I didn't write it, I just use it every day, and it's the cheapest change on this list.

Let the repo do the remembering. This is the big one, and it gets its own section below.

The machine

I should mention this because it undercuts the usual excuse.

All of the above runs on a personal M2 MacBook Air. 8GB of memory. A 256GB drive. In 2026 most people would call that machine finished.

Nothing runs locally. The models run somewhere else, the site builds in CI, the daily jobs run on GitHub's runners. The laptop is a terminal. Hardware stopped being the constraint a while ago and most people haven't noticed.

What my own site actually does now

The concrete version of all this is the site you're reading.

Every push to main runs 10 gates before anything ships. It tests the changelog pipeline, then runs it. A model reads the commit history and writes the visitor changelog, in my voice, because the voice is pinned in a file in the repo rather than re-prompted each time. A bot commits that back. Then

dependency audit, 2 syntax linters, post validation, 413 lines of SEO, AEO, and design checks, the build, and a post-build pass over the HTML, schema, images, and links. Only then does it deploy.

A separate job runs every morning at 09:47, pulls the latest release from every side quest repo, and updates the changelogs and the contribution graph.

One line in that config is doing more work than it looks like it is:

```
on:
  push:
    branches: ["main"]
    paths-ignore:
      - "_data/site_updates.json"
```

The bot commits that file. Without `paths-ignore`, its own commit retriggers the deploy that made it, which commits the file again, forever. One line stands between a nice piece of automation and an infinite loop billed by the minute.

I did not know that in advance. I found out the way everyone finds out.

The part that isn't a flex

I'm learning all of this for the first time. So the honest column of this ledger looks like this.

Models hallucinate at me. My instructions are unclear more often than I'd like to admit, and an unclear instruction burns real tokens producing something confidently wrong. I've spent whole sessions on the wrong problem because I never stated what "done" looked like. I've had an agent go off and rewrite a tool in the middle of a task I'd asked it to do something else entirely.

None of that shows up in a \$1,492 figure. It's real, and if you're starting out you should expect it.

I wrote up [the 4 worst ones separately](#), because the failures turned out to be more useful than the wins.

The thing that compounds

Here's what I'd keep if I could keep one habit.

Every mistake gets written down somewhere the agent will read again.

For my SEO agent, that's a file called `FEEDBACK.md`. It has 15 entries under "Fixed", and each one is titled as the symptom rather than the fix:

- A non-executable bin turned every call into a fork bomb
- The agent rewrote the tool during a user's audit
- Provider keys implied consent
- A connected analytics property that measured nothing looked healthy

Alongside it sits `SPEC.md`, which defines what “done” means in 5 clauses. Every open item gets sorted against that definition. Anything that doesn’t break a clause gets queued rather than argued about. That single rule took my open list from 6 items to 3, not because things got fixed but because things were finally allowed to leave the list.

And on my own machine there’s a memory directory: one file per fact, each with a summary line at the top, each linked to the others. 21 files right now. Why a build failed, why a name is frozen, what a client’s constraint actually is.

None of this is sophisticated. It’s a diary that the machine reads.

But it’s the difference between paying for the same mistake twice and paying for it once. When 99.5% of your spend goes on re-reading, what gets re-read matters more than anything else.

The misconception

The most common thing I hear is that people want to take a course first.

I can’t do that. I can’t sit through long videos and absorb theory. And I think the premise is wrong anyway. People treat working with these models as a science, and to some extent it is. It’s also an art, and the art is a very old one: conversing with clarity.

There’s no hack underneath it. Use the model as a thought partner. Pick the right model for the job. State the goal properly. And don’t let it be a yes man, because a yes man will happily help you build the wrong thing at great speed.

This post is the demonstration. I didn’t write it and hand it to a model to polish. I asked one to interview me, to keep asking follow-ups, to push back when I dodged a question, and then to publish it here in my voice, structured so both search engines and AI agents can read it. It pushed back twice. Both times it was right.

Do it once. Automate it the next time. That’s the entire method, and it costs \$20.