Solving the Halting Problem

Steven Zimmerman · October 26, 2026 · effortlesssteven.com

Shift left. Shift way further left. And I don't mean "*test early*". Shift left into your issues.

You're not shifting left because it's "*better for quality*". You're shifting left because you're out of compute and engineering time.

Token costs are falling off a cliff. It costs fractions of a cent to get enough intelligence to complete real work. So how does that change development?

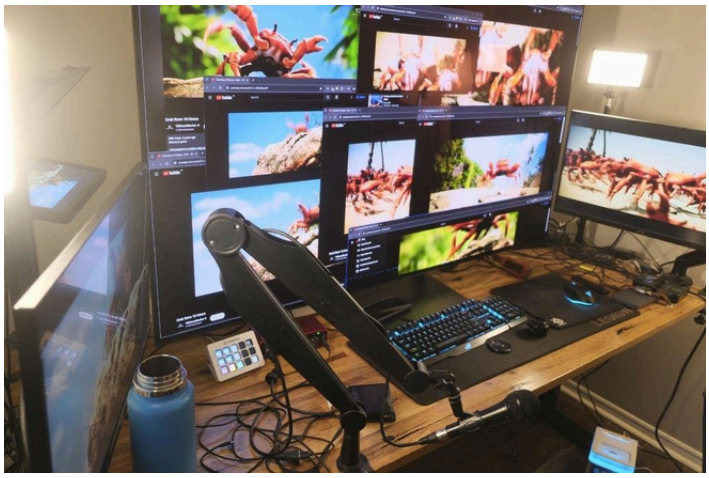
How does it change work when you have no choice? I got dropped into a 20-something person finance team that lost 30 people and had collapsed to 7 coming out of bankruptcy. A finance team that relied on regulatory-critical Jupyter notebooks, maintained by a five-man dev team that had also left. Notebooks that were the only thing standing between us and 20 to 30 million dollars in fines. And I hadn't read Python in five years. But Cursor could read it. Cursor got me through enough of those notebooks to keep the company running, to keep reporting going, to find and fix errors that already existed in the notebooks, and to help the company transition to a more automated future and survive post-bankruptcy.

But LLMs hallucinated hard at the time. Code looking right doesn't mean it was right, and I didn't have time to read it anyway. I needed a way to let Cursor figure out if the code was real; if the test actually tested. Mutation testing was the answer. Cursor could run for an extra hour or two and solve all the problems that would have been wasting my time.

But mutation testing is slow. I needed something faster, something that could lint well and test well, give me great feedback, give my agents great feedback, early and without wasting my time. And Rust was right there.

LLMs Have Nine Lives

mergecode was my shiny new tool. An AST context packer to shrink my code down into tiny context windows. You didn't have a million tokens to work with. You had 4,000. And you had to fit enough context in to get a usable answer back. Constantly copying back and forth between ChatGPT and AI Studio and Cursor, trying desperately to hammer it into shape.



Then Claude Code came out.

It was a step change. You could do new things with your codebase. You can move faster. You can go further. You can review better.

But there was duplication. Claude hallucinated APIs. We built features from scratch and kept starting new implementations instead of finishing what we had, over and over and over again. It didn't matter how good your design was; Claude was hallucinating a new one.

But that's okay. There are multiple versions. There are multiple attempts. It's an evolution. We can compare them. We can contrast. We can use it to help specify the right design. We can get closer and closer. We just needed to feature flag stuff properly, and it's fine. It's all good. We can learn from it and get the repo into better shape. Gradient descent. Yada yada yada.

Then F1 weekend arrived. I hadn't set up SSH yet, and I couldn't let my account sit idle for four or five days in a row. I was paying money for this. I needed to lean in. So I set up Termux on my phone on the drive to Montreal. I hammered away at it all weekend from the track, keeping my thumb on the display so I didn't lose the wakelock. Charging everywhere I could to keep Claude running.



Claude and I were ironing out the design, while little did I know, Claude was creating millions of lines of dead code in the background. Great ideas and concepts of how these platforms should function, but not much working implementation.

It was a point of no return. Those repos were not going back to how they were before. Too much design work had gone into them, and they had seen a glimpse of the path to controlling this firehose. To building something.

So what do we do? How do we stop those hallucinations? We modularize, we feature flag, we mutation test and unit test and end-to-end test and lock in everything we can. We consolidate branches. I worked through it day and night, and it wasn't enough.

Moving beyond commits?

I needed a better way to track my changes.

And this is where you're laughing. But you have to understand, I'm not coming from a traditional software development environment. I was a Finance FDE doing ERP deployments. I hadn't worked with other devs. PRs were not standard practice for me. They were unneeded ceremony. I'm the only dev in the repo. Why do I need PRs? What do I need to deal with all this for?

They were exactly what I needed to track agent history, to track the candidates, to give a GitHub-native surface for review, to encode the state closer to the work.

We went from no PRs to thousands of PRs.

We decomposed the work into a series of GitHub-native steps and encoded it into agents and skills that an orchestrator session could drive through the PR process.

But it was heavy. It was hundreds of files, and they were tuned to each specific repo in order to minimize context reacquisition costs. So I pulled together a stack to distribute my skill and agent sets across my repos.

All of my repos had swarm orchestration, and I could run 10 repos wide, 15 repos wide, for 9 to 12 hours straight per request, building massive 70,000-line well-tested features.

And that's too much code to read, even for the agents. So we needed a dashboard. We needed a way to reduce the amount of time required to understand a PR. We built out a glass cockpit explaining what changed in the PR, making it easier to understand.

But it wasn't for me. I have never read an internal PR. I built it to guide the agents through their analysis and make it easier for the agents to

review. We made it so that the state that gets encoded in GitHub better directs them on what to do next.

Layered review and analyses compressed into a series of GitHub comments that future agents could read and reason over.

Building layers and layers of verification and testing to protect against the fact that anything could be wrong. Any claim can be mistaken, hallucinated, or just plain inaccurate.

But we can't re-verify everything every time. It's too inefficient.

So we gave the agents the analysis history. How the prior agents reasoned over the work. We make it possible for agents to reason over what might still be unreliable. Make it cheap and fast for them to figure out the next steps.

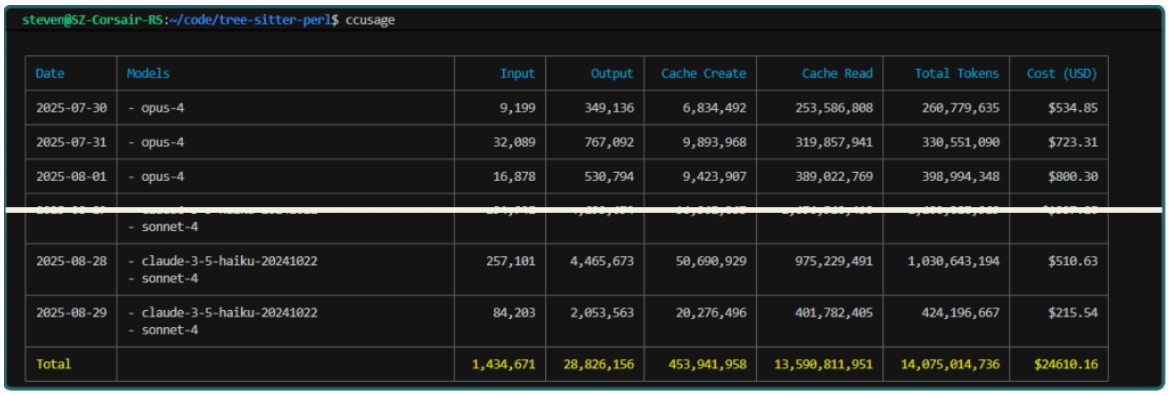
DemoSwarm

But they delved too greedily and too deep. I was running Claude all waking hours to make the most of the five-hour buckets. But people were using cron jobs (now known as loops) to keep Claude sessions running through the nights too. Then Anthropic announced weekly limits.

I thought the weekly limits were only for Opus. I thought Sonnet was still just the five-hour limits. So I redesigned everything to work around Sonnet, to use Sonnet all day long, to run continuously in Sonnet, agent after agent after agent, having the main thread act as an orchestrator, calling role after role after role to work through the traditional SDLC, to shape work from issue through to merge, step by step by step.

On the recommendation of Anthropic and Cursor staff, I had been exclusively using Opus for a while. But Sonnet was good enough and efficient. If you gave it small pieces of work and guardrails, it was able to get a lot done.

I'd launch a set when I woke up, and another before I went to sleep. I spent the rest of my day working on things and planning. I didn't have to babysit the code because the code was getting hammered by tests and review and mutation and oppositional review and formal verification and questioning if the PR should even exist.



```
steven@SZ-Corsair-RS:~/code/tree-sitter-perl$ ccusage
```

Date	Models	Input	Output	Cache Create	Cache Read	Total Tokens	Cost (USD)
2025-07-30	- opus-4	9,199	349,136	6,834,492	253,586,808	260,779,635	$534.85
2025-07-31	- opus-4	32,089	767,092	9,893,968	319,857,941	330,551,090	$723.31
2025-08-01	- opus-4	16,878	530,794	9,423,907	389,022,769	398,994,348	$800.30
	- sonnet-4						
2025-08-28	- claude-3-5-haiku-20241022 - sonnet-4	257,101	4,465,673	50,690,929	975,229,491	1,030,643,194	$510.63
2025-08-29	- claude-3-5-haiku-20241022 - sonnet-4	84,203	2,053,563	20,276,496	401,782,405	424,196,667	$215.54
Total		1,434,671	28,826,156	453,941,958	13,590,811,951	14,075,014,736	$24610.16

I got back 16 hours of the day that I could spend on other things. I became more productive while having better code with less attention.

Mistakes were made

I was getting mergecode ready for public release and building out language support. Its Perl dependency was out of date, unavailable on crates.io, and breaking compilation.

The smart thing to do would be to drop Perl. There's no reason to keep Perl. I don't use Perl. No one I knew used Perl. For the next nine or ten months, anyone I mentioned Perl to had one question: "Why Perl?"

I couldn't leave a significant language like Perl unsupported by mergecode. FFI was already a pain in the ass. I didn't want to vendor and maintain C code alongside it. I was coming fresh off of designing a reverse engineering flow, and was freshly thinking about API integration targets and testing. Why don't we just have the agents build a Rust parser targeting the tree-sitter output? It can't be that hard. It's a well-defined API.

A year later, more than half a trillion tokens in, I learned a couple things, namely that I should never have touched Perl. But we're a little too late for that now.

And to my great misfortune, Perl was a great testing ground for agents. Perl gave us reason to try microcrates for their clean APIs, modularity and small agentic surfaces. And Perl gave us reason to stop using them.

We had a mostly working parser within days, but it wasn't enough. It wasn't fast enough. It wasn't reliable enough. It wasn't accurate enough. It wasn't sufficient for what mergecode needed, and it wasn't sufficient for static analysis tooling.

Meat proxy

Then the swarm died. Not from the rate limits. It was from Sonnet 4.5 getting range anxiety. Sonnet 4.5 wasn't running 50-plus agents in a

row. It was running 20 and then stopping at the end of the context window, which, with a five-minute cache, meant I was never hitting the cache, and it was sitting with a couple hundred thousand tokens in memory that I'd have to clear regardless.

But I can see the finish. We are close to publication. If I just put my head down and grind through it with ChatGPT and Claude, I can do high-quality work at volume and get us there.

I can finish off the rough edges to get it launch-ready and get it out there and give talks on it, and explain how agentic development can create complex tooling.

So I mapped out my browser windows and terminals to maintain muscle memory across virtual desktops for different projects. I burned through session after session after session, trying to get there. I turned off compaction to get the extra 20,000 tokens from Claude. And every two hours (at the end of Claude's run) I'd dump the context back into ChatGPT, discuss the next steps, and send it back to Claude before moving to the next virtual desktop.

And unfortunately, Claude's output was not very descriptive in agent mode at the time. So if you wanted to give good information to ChatGPT, you needed it to give you the line-by-line streaming log that came from working directly in the orchestrator. But once you did that, you could give ChatGPT a full history of what's going on, let it reason for a couple minutes because the cache had expired anyway, and give good guidance to the next Claude session after you `/clear`.

Suddenly, I was orchestratorless and agentless.

But I was still making progress. I was making real progress, even without that orchestration.

And release felt right around the corner. It was always just a couple weeks away. Just a couple weeks away.

And we had a deadline. We had to be ready for the GDG talk. We had to ship in time.

For four months after operating a swarm, I kept manually routing between Claude and ChatGPT. I kept driving towards release rather than building compounding automation. Rather than fixing the swarm.

I could have gotten the swarm updated and working in October and had my days back four months earlier. But the drive to release stopped that.

I was giving talks about the very automation that I myself was not using. Because I was still producing more output at higher quality than most teams had.

But I could have been even more effective.

Stateful Work

But I can't be constantly giving the agents direction. I need them to know how to do the basics. I need to be able to discuss things at a high level and have them work through the specs and the ADRs and their skills and get it done without me having to watch them. I need to get my time back.

I need to get to the point where I'm vaguely discussing the direction with ChatGPT and having the agents orchestrate it without me having to also route them. Or at least not as much. I need them to survive crashes and to make small changes that are easy to track.

I need it to decompose the direction into issues for implementers to track, into states within a PR, into bits and pieces that move the direction out of the agent roles and into just-in-time notes that the agents come across that keep them on the right course. Make them able to interact continuously without me directing them by giving them clear work state in GitHub that they can work through. A clean issue queue with well decomposed issues.

And the ChatGPT connector makes that easy. ChatGPT can write directly to GitHub and take the issues that you are vaguely gesturing at and turn them into real implementation plans that have researched your codebase. It can build out the work queue that Claude on a loop can work through.

And Claude's cache extends too. It's not just a five-minute cache. It's a one-hour cache. So you have time to find something, reason over it with ChatGPT, and drive the next session without clearing.

And then agents got better too. Agent teams can communicate with each other without having to leave filesystem notes. Suddenly the agents weren't blind runs, but active sessions that orchestrators could drive.

And the old swarm was getting in the way, because the work-defined roles were not fitting with the role-shaped agent team design. We had to move much of the step instructions into skills and separate them from the roles, because the orchestrator wouldn't know that those agents were designed for a dedicated flow and would call them anyway.

We had to take the work state out of the agents and put it into GitHub state and skills to guide agent teams through it. Have them work over the issue and over the repo and take over issues and file whatever they see that's out of scope and carry things through end to end.

You're taking the graph of the work and encoding it close to the work to minimize the lookup and minimize the context needed. Make the work help guide the agents through. Have each skill point to the next options in the flow.

Having agents encode their findings directly on GitHub to make future agents be able to reason over it more readily.

Yeah, we're reducing the amount of reasoning that has to be done each time, because each agent says what they think and why, and that way each new agent starts with that context.

Making it easy for agents to make reasonable documented decisions by documenting reasonable decisions.

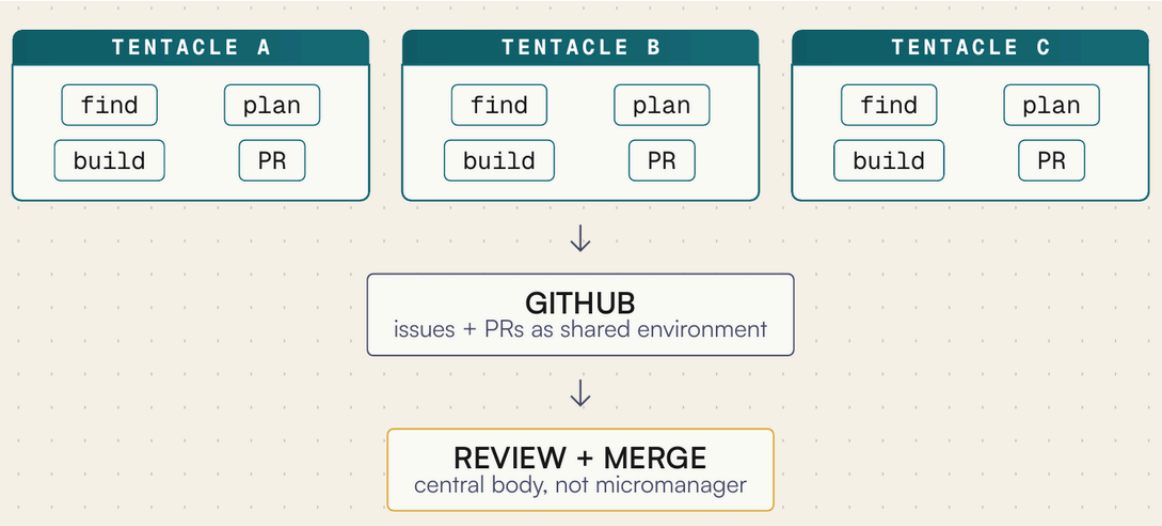
Octopus Cluster

Then Copilot launched Fleet Mode. And it worked.

Suddenly, my swarm was working again. Suddenly, things just ran. Suddenly, I didn't have to watch the agents anymore.

For a week, it saturated my compute. My desktop and a half dozen other computers as well.

I was queuing up multiple swarms of Opus agents that couldn't talk to each other. Each swarm could only interact with each other via GitHub. So I got them working together via GitHub.



Some swarms were building candidates. Some swarms were reviewing. Some swarms were testing. Some swarms were merging.

I had my time back. My computer was busy, but I could do other things. *The code quality was not that big of a drop-off*, and I was no longer involved.

I was nervously watching, but there was nothing for me to do. So I got other work done. I cleared out things that were on the backlog.

And then GitHub changed their pricing.

And banned me from Copilot.

And it's impossible to reach a human at GitHub Support. So what do we do? How do we get unbanned? We ship.

We do everything we can to get perllsp ready and turn it into critical developer tooling in the hopes that that might help us get the attention of a human at GitHub to get unbanned.

Let's ship, let's ship, let's ship.

And so for another two months, we are in a sprint to ship again, spending all day and all night trying to get perllsp good enough to properly announce it, good enough to ship the public alpha at Toronto Perl Mongers on April 30th.

Two back-to-back release sprints, putting off compounding orchestration gains. And all the while, I'm looking at my CI numbers and trying to figure out how to get them down without losing all verification. Our CI cost per PR is not bad, but that changes at agentic volume.

I upgraded to an Enterprise account for the 50,000 minutes, and that still wasn't enough.

The CI wall

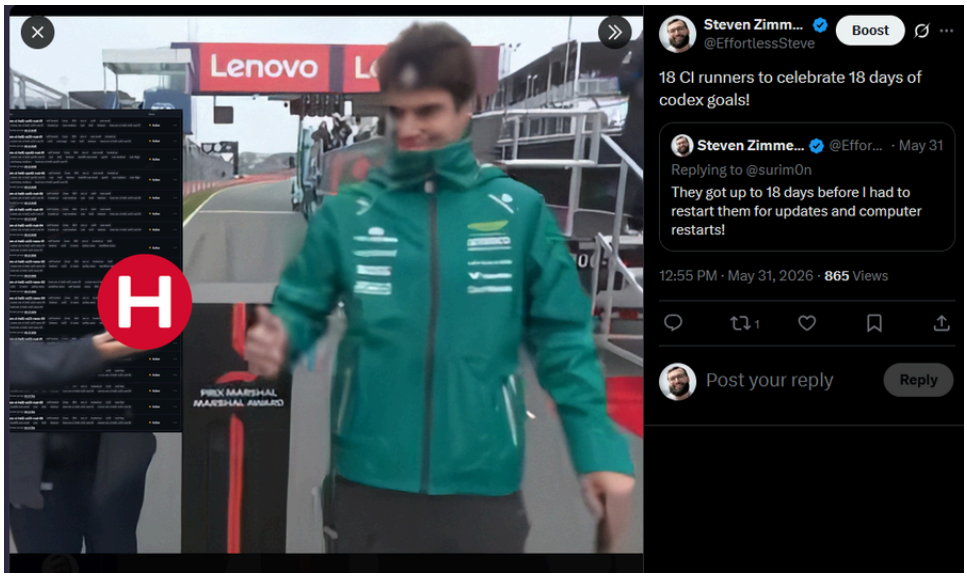
Then OpenAI gave out 200x months to everyone that applied for the 5.5 launch party.

And I panicked.

How the hell do I use 200 billion tokens without completely redesigning my SDLC? Do I just turn CI off?

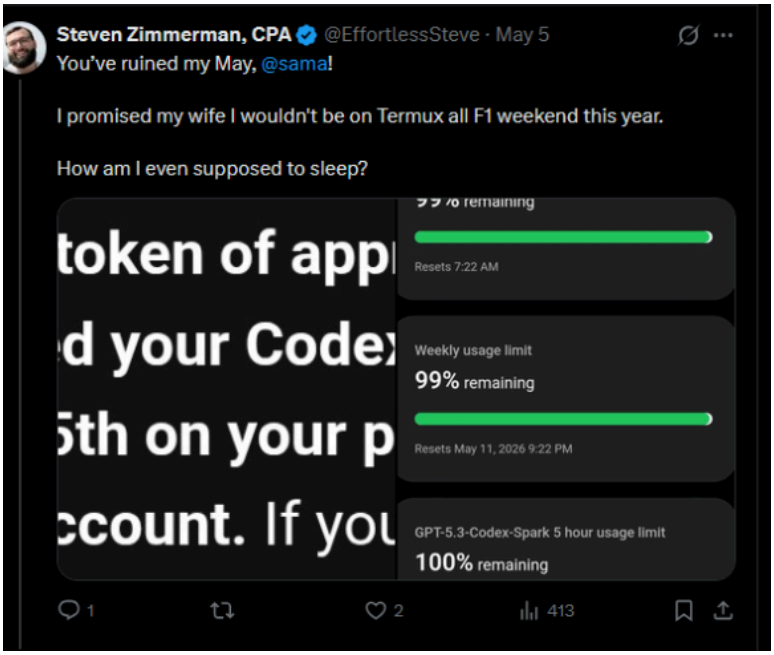
There was no way GitHub was going to sponsor 200K in CI.

So I grabbed servers. I got eight runners. It's not enough. 13 runners. Not enough. 18 runners. Not enough. **25 runners.**



$500 in Hetzner VPSes. And it was barely enough.

F2



But there was a problem. May was the Grand Prix. I was going on vacation for a week. I couldn't steer a 200x plan. I had to set it up on the new Goal Loops and let it run.

It wasn't like 2025 when it was on my phone in termux. These were on my desktop terminal back home, plugging away.

But it was fine. They had good instructions. They were doing the right things. They were just working through the issue backlog. They had good goals. And you know what? If they finished a little early and

stopped working partway through, okay, fine. At least I got, like, three, four useful days out of them.

But then IBM reached out. They were looking for an inline completion target for LSP 3.18 support. And ours didn't quite work. And I had never tested it. I had built it for spec completeness, but Claude said it was close to actually working.

And the swarm was still running. The swarm was fighting me. It was blocking my PRs because they weren't authorized and were out of lane. But it was only 3 or 4 small fixes needed. It mostly worked already.



So I sat there, watching cars race around the hairpin, and then spend a minute debugging with Claude while waiting for the next lap. And that was enough to get it pulled together and get a release out to get distributed in LSP4IJ.

And then the 200x month ended. And suddenly I was in a release sprint again, because TPRC 2026 was coming up, and I had to get perllsp 0.17 ready.

Mostly Working

Finally.

The conference was over.

I could take a moment to catch my breath. Try and catch up on some publishing and video editing and etc.

And then 5.6 Luna came out.

30 billion tokens a week of near GPT-5.4 intelligence. Of enough intelligence to run the factory. And not just for a single bonus month. Suddenly I had the capabilities of the 200x month every month.

I couldn't retire the Hetzner runners.

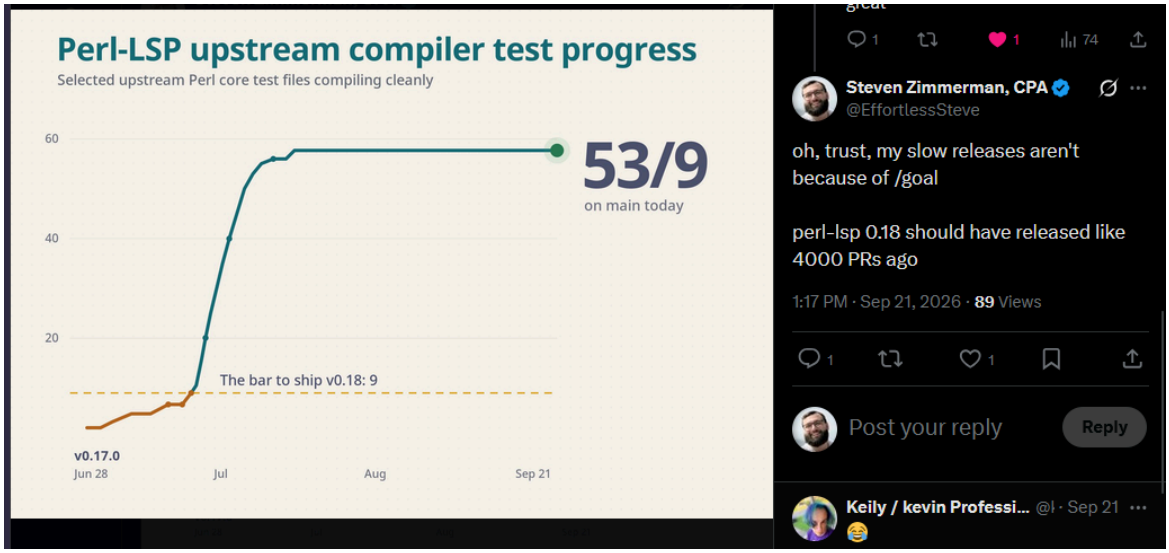
I needed every MB I could get.

Those plans to relax went out the window because I needed to lay down the issues and roadmaps for this massive flow of tokens. I needed to give them direction.

Once again, rather than compounding my automation and fixing my swarm, I had so much capacity that my time all went into directing my capacity.

But it was good. perllsp was advancing. We were getting close to a very good 0.18 release.

And then we just drove past 0.18 and didn't publish it. Because we had too much capacity. And we kept finding more bugs and more things to handle and more edge cases to cover and more compiler facts to test and etc., etc., etc.



perllsp 0.18 was almost releasable. We just needed to keep burning down the edge cases and the issues. Just a couple more issues.

I could just ship.

I could just ship and it would be beyond my release bar. It would be a better experience than the bugs people are currently sitting with.

But there's still more bugs that I need to run down before it's good enough to ship, because the bar keeps getting higher, because I keep finding new edge cases, new corner cases, new depths required to properly parse. And solving them is always just a couple weeks away.

The shape of the work

I'm constrained by CI and hardware and compute, not tokens or planning.

I have months of issues decomposed. I have hundreds of billions of tokens available.

It feels wasteful to burn more tokens for the same output, but I need to do everything I can to address these growing bottlenecks. Because at 2,000 PRs a week, even ten dollars per PR in CI is too expensive.

So I'm actively doing everything I can to make my CI more efficient, because I can see more throughput coming. Tokens are still getting cheaper. Quality is getting high enough that we are reaching pure scaling. And at that point, all the tooling designed for a single seat fails. I don't have a single review bot that keeps up with me. I have the tokens to do it cheaply, but most solutions were built for a different scale. I am having to build custom tooling to try to reduce those verification costs.

But the tooling isn't ready fast enough. I need to do more. I need to redesign my SDLC.

I need to shift CI right.

But how far?

CI on drafts is right out. CI on ready? No, the agents will flip it to run remotely.

CI on merge? No. We don't want an agent to improperly turn that on and have GitHub auto-merge. We want a merge agent to review and handle the merging after checks are green.

CI on label? It's not ideal. But it keeps the check on every agent, and as long as we set everything up properly, it runs smoothly. But there's a step further you can go.

No CI on PRs.

CI on main only.
CI run once an hour or every 10 PRs, whichever takes longer. And tooling to help discover where breakages are within main, and then tag on freeze. You can still use the PRs to track state and movement and candidates. But there is no external merge gate. There is no external CI gate. Your production is fully decoupled from your integration testing.

Run the CI checks in the session. Run reviewers in the session. Get answers live as you're going. Don't rebuild context from scratch. You've already compiled. Use that same session. That same coding box. Use it to provide the information and review environment needed to get the PR into the right shape. You don't need to spin up another runner. You can have the session actually review its own work and combine it with a static reviewer that benefits from the information the session uploads to provide useful review cheaply and at scale.

And I needed to, because I really needed to scale down that Hetzner fleet. 500 bucks a month adds up fast.



I've got power. I've got internet. Let's just stack some mini PCs in the corner and run it myself. I don't need to be running always-online VPSs. I need cheap, efficient workload. I need boxes that can compile Rust and run my CI with a safely partitioned build directory.

Rust build cruft stops being a problem if you can just blow away the build directory.

And storage is huge because these are Rust. And memory is huge as well because this is Rust.

And meanwhile, I just keep thinking I should have stuck with Python.

Python works in ChatGPT. ChatGPT Pro can build fantastic applications with Python. It can't build Rust. It doesn't have it installed in its environment. My CI issues are because of Rust build size. Python isn't that slow.

I'm out here moving away from Rust for ecosystem reasons.

I've had weeks where I had billions in excess tokens because I didn't have the compute to support it.

But just because you can doesn't mean you have to.

You can have your agents pre-research issues without giving them full merge authority. You can have your agents kill mutants while you're in a meeting. You can train your agents to walk through a GitHub native flow to reduce the amount that you have to remember to prompt each time. To make the work easy for the agents.

You can put on the strictest lints you want, as long as it's making the code easier for the LLMs to verify the shape of. Because it's not for coding. It's for maintaining.

More done with less attention

Another vacation, but something was different.

I wasn't stressed. Well, not from code.

I wasn't checking my phone. Not for code.

I'm at a rodeo and I didn't check my phone once. I walked the whole state fair.



Carol M. Highsmith / Library of Congress (2013)

The work was well enough decomposed into issues and planned out well enough that there weren't any decisions I had to make live. Everything just ran. I checked on things once a night, but I wasn't directing agents all day.

There was more work than capacity. There was more defined work than the capacity we had.

I was able to focus on other things.

The shape of the tools

Give me dashboards. Give me metrics. Give me KPIs. Agents maintain them just fine.

Show me a good summary and dashboard and actual understanding of where it's at, what's happening, what needs to be done, and what questions are actually outstanding.

A task tracker is a task tracker. A Gantt chart is a Gantt chart. It's shaped that way because the work is partially shaped that way.

The shape of the work is shaped in part by the work.

Our work tools are the way they are in part because they are fit to the tasks that we use them with. Not just as a human interface. They help keep the agents on track as well.

And not to discount the value of improving understandability for humans.

Let me dial into each of the tasks on the Kanban and see what is coming up. Help guide the agents by giving them the right information for their upcoming tasks for when they need it. Work with Claude and ChatGPT to maintain the information they need to track projects effectively, and keep themselves on track.

Even basic templates generated from JSON are enough to improve understanding of the work, and the progress through the work. And Claude and ChatGPT are both far better at visualization than just that.

Why are we looking at a streaming log?

The agents are good at creating useful summaries and tracking changes in state.

And that goes beyond just the coding interface. You don't need all the interfaces to your agents to be chat logs. OpenAI and Anthropic are supposed to be Future of Work companies. They're supposed to be thinking about how to use LLMs beyond current uses. And yet not one of their PMs has released their Kanban. Not one of their PMs has released their ChatGPT-maintained Kanban.

You don't need to have only one interface. Google has many products. Apple has many products. Microsoft has many products. You can stick a designer on any interface style you want. People will gravitate to the useful ones. And you can try out all the different shapes that workplace assistants can take, all built on the same APIs.

You can have ChatGPT maintaining your project tool state, keeping a Kanban that your ChatGPT conversations can access, and that your dot can keep up to date. Your dot can keep track of the projects in your life and make it easy for you to see where things are at and what needs to be done and what next steps are. It can proactively build useful context packs if the interfaces are built.

Persistent workspaces with work-shaped interfaces rather than just doc-shaped Spaces. You have all the tools and information needed to make organization easy.

Speaking of the dots.

They have too many approvals. They're still just a chat interface. They lose track of tasks constantly. They don't have a current state graph that's sufficient, and they use an OptMem-like memory system that works pretty well, but can get a bit hung up on confusions that it runs into. But it's the new thing. It's a built-in orchestrator. It should still be useful. Let's set it up.

"Hey, Kestrel, use my computer and investigate my OpenClaw and see what's up. See why it's not working. See if you can fix it."

Boop, boop, boop. It's done.

Oh shit.

It worked.

It got my OpenClaw working again. The OpenClaw that I spent months debugging with ChatGPT Pro that I was planning on just blowing away.

And I didn't need to do anything. It just searched through memory, did a little search, used the SSH connection I set up for it, and figured it out.

It doesn't have a Kanban, but it had a similar oh shit moment to the original OpenClaw launch. Because it just worked.

Between the bots and the session history search improvements, we're finishing tasks in the time we used to spend building initial drafts.

But there's always more.