Throughput, reconsidered

I want to talk about a genuinely surprising personal engineering breakthrough I've experienced. Over the past 3 days, I challenged myself to experiment with a true AI-native development workflow based on the advancements I've observed over the last few months. I chose Cursor Ulta, Warp, superwhisper, Claude 4 Opus and a few other capable models for different purposes, and focused on creating the most comfortable and fluid experience possible.

I'll provide my background so you can decide if this is relevant to you, but also to lend some credence to my claim. I've been an engineering leader and staff engineer for a couple of decades. My day job is managing around 40 people in a distributed environment as a CTO of a venture backed startup. I've touched a lot of platforms and languages over the years, including native and close to the metal commercial products. Plenty of failures and a few commercial successes along the way. I am mostly quiet on social media.

Anyway, lets see some results.

Output

- 6 pull requests
    - Additions: 21,834 lines (5,103 + 1,655 + 11,233 + 2,314 + 840 + 689)
    - Deletions: 189,850 lines (1,016 + 12 + 4,709 + 0 + 184,017 + 96)
    - Files Modified: 1,272 files (65 + 14 + 70 + 29 + 1,079 + 15)
- Duration: 2.5 days

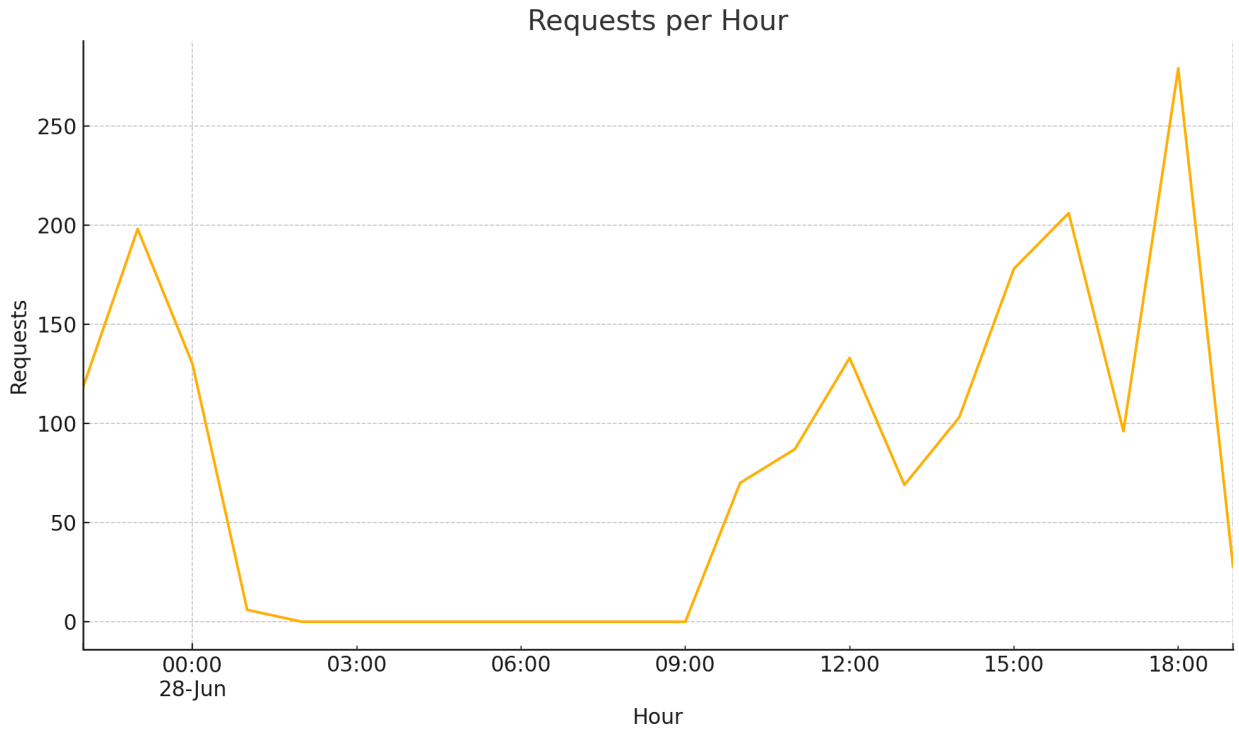
Outcome

- 4 new features for Encamp (startup where I serve as CTO)
    - A document upload and indexing pipeline using LLMs for structured metadata extraction with a vector database for semantic search
    - A UX pass that improved our search and filtering experience for environmental documents, combined with a mobile responsiveness pass
    - QR code generation and printing for any URL slug, pointed at our mobile experience for environmental data collection with edge caching
    - An improved search experience for our chemical catalog, surfacing state of matter, density and improving the workflow for some of our subject matter experts
- A handful of command line tools that improve developer experience
    - A tiny Figma MCP proxy to work around troubles I had with Cursor's built-in MCP support
    - A tool for bootstrapping and working with Git worktrees for developing code in multiple branches with multiple local agents
    - A tool for starting up multiple local clones of our Vite/Apollo-based web and API servers for testing

- A massive refactoring and removal of dead code in the area of our monorepo that deals with operations migrations and scripts for manipulating data
- Throughout: well architected, linted, unit and integration tested, infrastructure included (we use terraform extensively on AWS), observability included (cloudwatch alarms etc)

Model usage

- All these calls were through Cursor, unfortunately I didn't track other tools
- ~120–200 req/hr, peaking near 280 req/hr in late afternoon on our second day



Model types

- claude-4-opus-thinking (daily driver): 1,668 requests (6.6% error rate)
- default: 33 requests (9.1% error rate)
- I ran out of quota (on Ultra) near the end of the second day, reverted to claude-4-sonnet
- "Max" (more re-prompting and tool-use) mode mostly enabled until quota issues
- Alternatives
    - claude-4-sonnet (Anthropic) - good/fast results
    - o4-mini-high (OpenAI) - good/fast results
    - gemini-2.5-pro (Google) - comparable to opus, maybe better spatial reasoning
    - o3/o3-pro (OpenAI) - weakest anecdotally, but good at planning

The new stack

- Cursor Ultra - you probably know Cursor, but if not: a vscode based editor that offers an agent first experience. Its “Max” reasoning mode allows for near infinite re-prompting and tool-use. I suspect this can likely be replaced with Anthropic's Claude Code product as a direct swap in. Codex CLI from OpenAI roughly a few months behind from a usability perspective (as of now, June 29, 2025)
- ChatGPT Pro - primarily, I use o4-mini-high and only fool around with other models. It’s great for brainstorming and having a second model’s perspective. I use it for data science-based exploration. I don't use Codex or Operator, I don't like their paradigm for a variety of reasons (primarily that I don't want to do work in a cloud instance or set up containers replicating my entire life). I occasionally use Deep Research and o3-pro for where I want extensively sourced materials.
- Warp - a modern terminal that has built-in multi-model agent-based AI. Not strictly necessary, since Cursor has a built-in terminal and terminal tool-use, but I find having a few different shells in another window is preferable for some tasks.
- superwhisper - a voice dictation tool for macOS with templated prompting. It transcribes speech extremely quickly and integrates with your clipboard. I use it for prompting mostly. I don't find I like writing with it, other than to capture ideas, mostly because when I'm dictating, my ideas come out in a strange order and require a tremendous amount of restructering anyway.

The method

My main goal here was to get to a place where I felt like I was in a flow state. There is a feeling where you have a well tuned editor and terminal with your own keybinds (vim or whatever your flavor), some ambient music and a vision for what you need to build and you can just jam. I feel like I achieved that here.

Multiple threads

The first thing I noticed when using Cursor with a model like claude-4-opus with Max enabled is that it takes its sweet time. I'd estimate somewhere between 2 and 10 minutes for medium complexity tasks. The results are pretty phenomenal - verified with unit and integration tests, usually at a minimum getting to a working state where we can debug things to correctness without wading through a sea of garbage.

I'd compare it to working with a relatively capable junior engineer whom you are mentoring much more regularly than you would in the real world. Because this paradigm is a little slower, it makes you immediately reach for multiple threads of development. For some, this would be nails on chalkboard. For me, this is actually a plus - maybe its just how my brain works. I just limit to 1 - 3 threads of thought at a time depending on complexity.

Small diversion into "auto-apply" with terminal commands. I mostly do NOT have it turned on, because these models can occasionally go deeply off the rails and do insane things, or sit spinning on a series of terminal commands that aren't yielding fruit due to local setup or even something as simple as a "pager" (like `less`/`more`, which is used by the AWS CLI) interfering with its intuition about command output that it should be able to see. This means I was interacting with each agent much more frequently than strictly necessary.

Threaded development paradigms come in different flavors in our stack:

- **Cursor "tabs"** - multiple agents in the same editor, limited to 3 at a time by Cursor 1.0 itself. The pros here are that you aren't managing multiple windows or directories, but the cons are pretty heavy: agents interact with each other poorly, undo each others work, got into deadlocks editing the same files. And when your ideas are too dissimilar, they need to be in a different branch for hygeine. I primarily use it to make small edits to a secondary set of files or work on a back-end task while something is working principally on front-end, but in the same concept, so therefore the same branch
- **Multiple cursor instances** - multiple agents, one per editor. No limits. The pros here are that you are completely free to make as many changes as you want and don't need to worry about having to wait for agents to fight over the same files. The cons are it requires a bit of special setup and you have the cognitive load of dealing with and identifying multiple windows operating on different branches, but which look similar.

On remote agents

You might ask yourself - why not Codex, Devin.ai or Cursor Background Agents? No issues with managing complexity of multiple windows and so on. My answer is pretty simple and practical: with the amount of intervention I'm performing on a regular basis to get realy high quality results, there is no way with the current generation of UX and models that this is possible in a fully remote environment. I want to maximize my own output as a person who understands force multiplication and has a vision for how something needs to be built, not have a bot fix my low complexity low priority bugs. It's currently a fundamentally different value proposition.

Secondarily, there's just a lot of set up to make a working development environment. Despite best intentions, every codebase I've ever worked on has required at least a day of environment setup. Local tools, state, sandbox environment, quirks of tooling, versions, operating systems and so on.

I can see remote agents being the way in the future as you scale this process up extensively, but its going to require us to have the next generation of tooling in my opinion. With how fast things are moving, this might be 6 months from now though. Honestly, I hope as an industry we focus on the "native" experience with keybinds and management systems optimized for the absolute power user. We'll get disproportionate impact. I think thats why Claude Code is so instantly popular - they're focusing on the hardcore engineer who loves their environment and cares deeply about getting into the flow state.

Git worktrees

This is a bit of a game changer. Just set up 3 directories, but reference one git database in a main repo. More or less just a copy of your development environment that three instances can interact with.

We don't absolutely need worktrees for this, but it saves a bit of time and space.

The only snafu, as I mentioned with remote agents, is you likely have a bit of setup that you're going to need to do per directory. So, one of the tools I created inventoried all the local state in my development environment and copies that state into your new worktree. Its specific to Encamp, but the ideas are pretty simple: copy `.env.local` and any other local config files, copy your terraform/terragrunt environment state, but leave behind things like packages and build outputs. Its easier and less error prone just to type `yarn install` and `turbo build` again (or whatever your flavor). If you've got a relatively hygenic codebase, hopefully this is limited to 5 - 7 files and a few directories worth of state.

It looks like this:

```
~/dev/
├── branch-1/          # Main worktree (jm/document-parse-metadata)
├── branch-2/          # Secondary worktree (jm/encamp-cli-worktree-sync)
├── branch-3/          # Tertiary worktree (jm/filter-improvements)

git worktree list

# Output:
# /Users/jmoyers/dev/branch-1  83fb585af3 [jm/document-parse-metadata]
# /Users/jmoyers/dev/branch-2  b4f9fa2aca [jm/encamp-cli-worktree-sync]
# /Users/jmoyers/dev/branch-3  bb8625f868 [jm/filter-improvements]
```

The next wrinkle was just getting my local dev API server and web server to run side-by-side for each worktree. To achieve this, I just built a simple tool to incrementally check for port availability and bind appropriately and use environment variables to keep each component in sync.

Once the setup is done, open a Cursor instance per branch and let the good times roll.

Repeatable success

As you build up habits and realize the parts that the model is going to consistently get wrong, you absolute need to record your findings. I started out with README.md files - I moved to Cursor Rules, which are affectively the same thing, but with a front matter-style block to tell the agent when to attach the rules to the context automatically. I've found that the rules are somewhat inconsistently applied, but I'm confident thats going to get better.

As you learn the shortcomings, you are basically shoring these up with a context aware system prompt. Start investing in this immediately and you'll only have to occasionally nudge the model back to instructions pertinent to the style of problem you're trying to solve. This doesn't work perfectly, but I suspect it saves a huge amount of frustration factor over the thousands of hours you're likely to be working with these tools.

Integrating design: Figma and model context protocol

This technology is obviously brand new (June 4) in its official form. It works using your local (nativeish) Figma instance, and is a beta feature you need to enable. The basic premise is to select a frame in Figma, get a node ID, and then having your agent request information about that frame. There are a few variants, but the idea is to get a screenshot and some generic codegen for the components referenced in the frame.

The agent can use this information to build UI without having to have the extremely profound spatial reasoning required to properly interpret a screenshot by itself, but also not rely on Figma's (perfectly workable, but generic) code generation. Its like giving the model a screenshot, but also the general layout of a design in a format more suited for the current generation of LLMs to make working product from.

I'm not a big believer in handing over enormous tasks to these agents, so this gives us an opportunity to create a smaller more achievable scope. It also means you have an intermediary step between some generic codegen and integrating it with your codebase.

The only catch: it didn't work out of the box. Cursor's MCP tool-use integration hung for me and couldn't retrieve any information. Its a black box with no obvious way to debug it. So I wrote a few curl commands to debug Figma's MCP server (it uses SSE), and it turns out it works perfectly well — it looks like Cursor's MCP tool-use that is broken.

So, I had the idea to create a tiny proxy that exposes the same tools, and have Cursor call that through the terminal and boom. We're back in business. It does two things only: downloads a screenshot of the frame and gets a snippet of typescript and puts both of those things in a `.gitignore`d directory to reference during implementation. I wrote a small Cursor Rule to reference these, but to match our surrounding React application and use our chosen UI toolkit (MUI in this case), and it produces surprisingly excellent results. I nudged towards a near pixel perfect implementation with around 15 minutes of fiddling the first time through.

How can we make this better?

A few small things could make this even more pleasant. One thing is that I currently stick to manual approval for terminal commands. This is fine, and I prefer it for safety. However, when using multiple agents, there is no fast way to figure out which one is waiting for input. Cursor has a notification sound when an agent completes, but not when it needs intervention. All we really need is a notification system that takes us to where we need to be after acknowledging it — the right window, the right tab, and the right text input focused.

We could also use more nuanced rules about auto-apply. Its hard to imagine agents not being involved in the decision making process here, but we could, at a smallish extra expense, try for a multi-model quaram. We could be looking at a tool-use invocation and be judging its destructiveness based on an expanding circle of privlege (mutate local state, mutate state outside current context, mutate state out of project, touch the network, touch infrastructure etc). That said, I think we're fine for another 6 months to a year if we just had some better UX for directing us to the right places for intervention when managing multiple agents. Make keybinds, keep us in a flow state, and help us mange multiple contexts.

An aside on cost

I think these tools must be sold as a significant loss leader if my napkin math is close to the mark. It seems like even if there is wild efficiency in Cursor's context pipeline, the inference costs for a power user seem like they'd reach >$20k/year. Am I totally off base here?

Conclusion

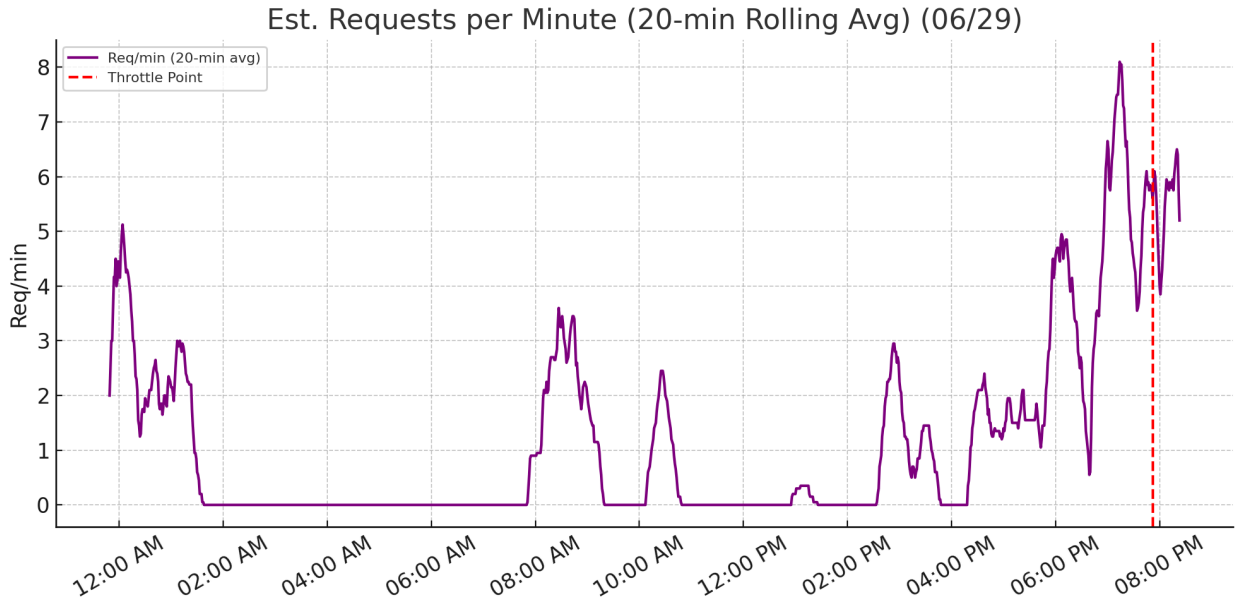
This is more than real. It’s a huge throughput gain with zero sacrifices other than becoming utterly and completely dependent on new technology. That is scary. However, there's a ton of healthy competition in this space, and I'm not personally going to let my brain atrophy and allow the robots to do all my thinking for me. In the words of Alexandr Wang, you still need to "really, really care" to produce something extremely high quality.

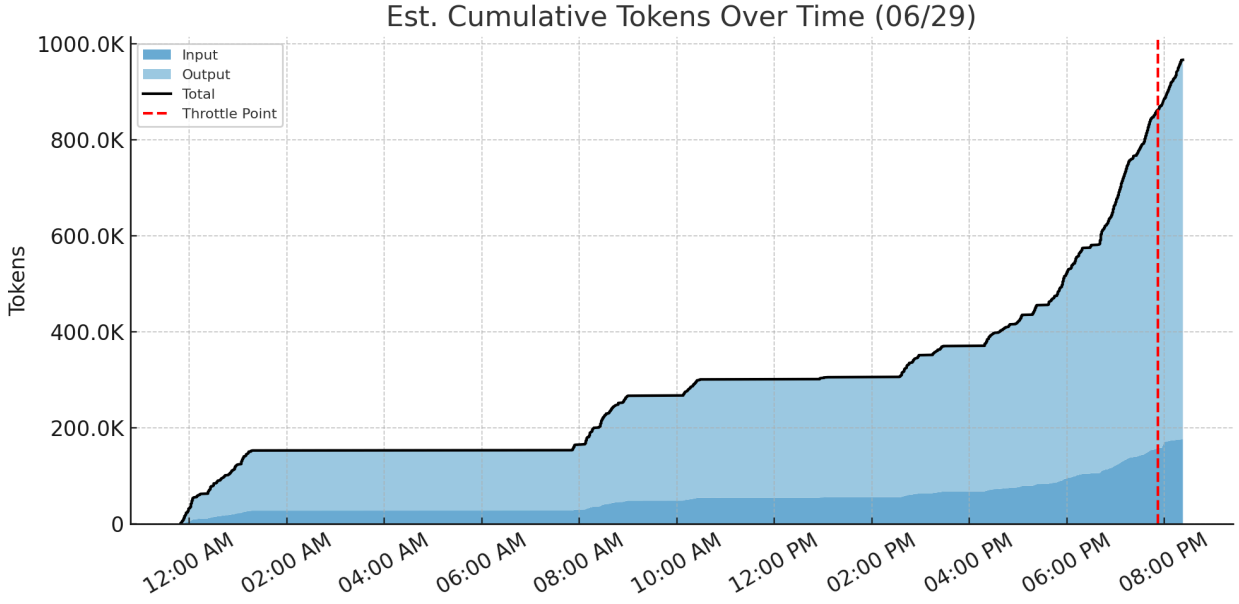
Long term, lets hope we're headed towards The Culture vs. the Butlerian Jihad. Go read Ian M. Banks - parts of that future he describes are dark and dystopian, but large parts of it are hopeful.

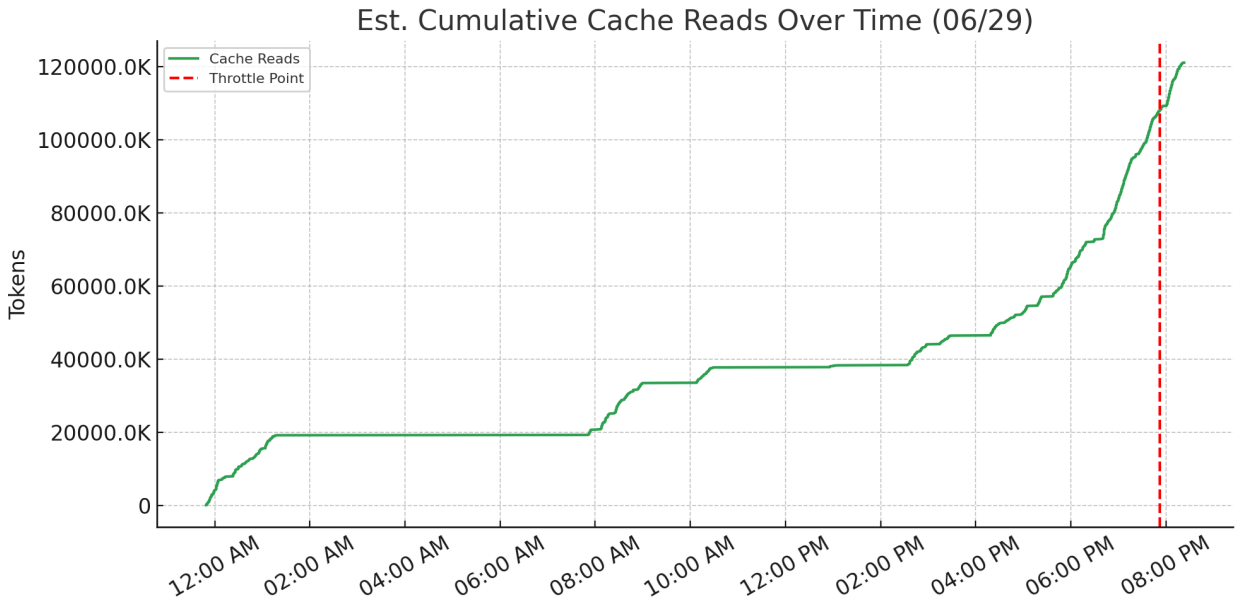
tldr:

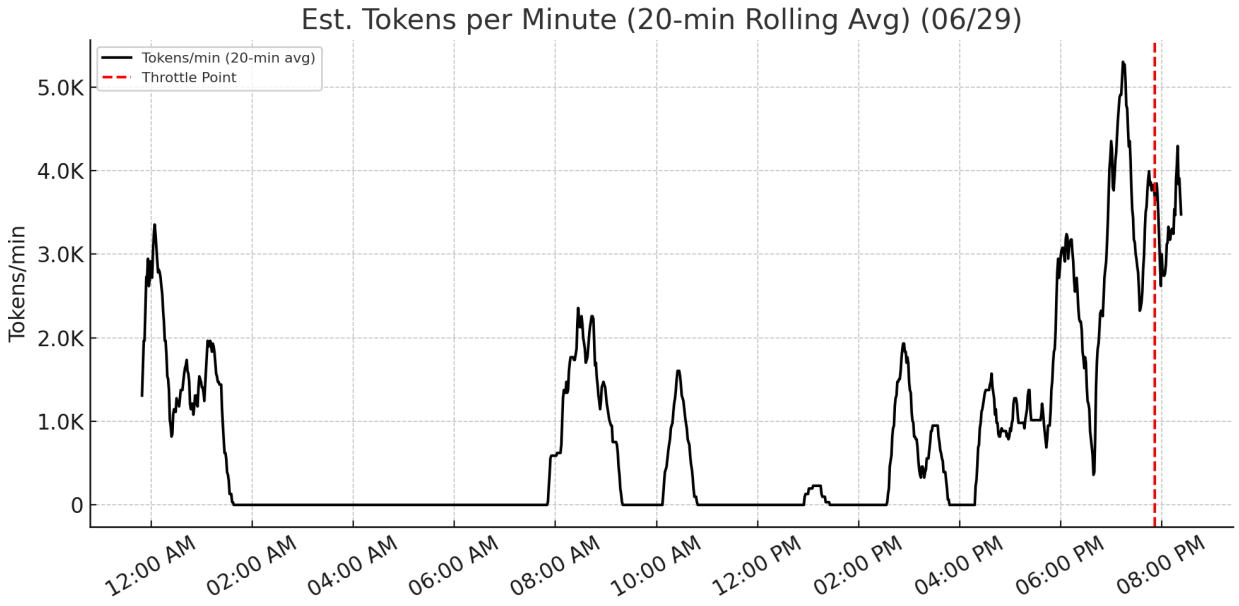
- CTO with 20+ years experience ran a focused 2.5-day AI-assisted sprint
- Native AI dev stack: Cursor Ultra, Warp, superwhisper, Claude 4 Opus
- Shipped 6 PRs:
    - ~21K lines added, ~190K lines deleted
    - Touched 1,272 files
    - Delivered 4 production features (LLM metadata extraction/indexing/semantic search, mobile QR tools, UX/search upgrades, chemical catalog improvements)
- Wrote supporting CLI tools (e.g. Figma MCP proxy, multi-worktree bootstrapper)
- Maintained high quality: infra, tests, linting, observability all included
- Leveraged Git worktrees + multiple Cursor instances to parallelize agent threads
- Agents are like capable juniors: effective, especially with tight scope and context rules
- Massive throughput improvement, feels sustainable with native workflows
- Costs to vendors likely unsustainable (estimate: >$100K/yr inference costs)
- Dives into methods, ergonomics, and why remote agent UX still isn't there yet

Extras (another day of data)






Est. Cumulative Cache Reads Over Time (06/29)
Cache Reads
Throttle Point
Tokens
120000.0K
100000.0K
80000.0K
60000.0K
40000.0K
20000.0K
0
12:00 AM
02:00 AM
04:00 AM
06:00 AM
08:00 AM
10:00 AM
12:00 PM
02:00 PM
04:00 PM
06:00 PM
08:00 PM


Est. Tokens per Minute (20-min Rolling Avg) (06/29)
Tokens/min (20-min avg)
Throttle Point
Tokens/min
5.0K
4.0K
3.0K
2.0K
1.0K
0
12:00 AM
02:00 AM
04:00 AM
06:00 AM
08:00 AM
10:00 AM
12:00 PM
02:00 PM
04:00 PM
06:00 PM
08:00 PM