Claude Fable 5.2 ships: lower token price, but early users say per-task cost and effort settings need checking

Coding Agents Alpha Tracker

2026-11-10

Claude Fable 5.2 ships: lower token price, but early users say per-task cost and effort settings need checking

By Coding Agents Alpha Tracker • November 10, 2026

Anthropic's Fable 5.2 launch dominated the period. Practitioners shared pricing math, effort-level guidance, an ownership prompt, and when to pick it over Opus 5.5. Also covered: Simon Willison's live-iteration build, the Pi team's issue-triage bot, and new remote-machine controls for agents.

Fable 5.2: cheaper tokens, similar cost per task, and effort settings that changed

The release. Cursor added Claude Fable 5.2 right away. - **CursorBench:** 56.9% at default effort, up from 49.2% for Fable 5.1. At max effort it scores 60%, the top result on that benchmark [1]. - **Price:** $8/M input and $40/M output tokens, 20% less than 5.1 [2]. - **Speed:** Addy Osmani says it generates output up to 30% faster than 5.1 [3].

Watch the per-task cost. Osmani says Fable 5.2 "uses a lot of tokens". In their testing, a task costs roughly the same as on Fable 5.1 despite the lower price [4]. Theo's numbers on the $200 Claude Ultra plan point the same way. He estimates the plan covers about $15,000 of Opus 5.5 a month but only about $3,000 of Fable. Fable is also capped at 50% of the weekly limit and "burns faster per $" [5].

Theo's preferred measure is work done per week: - Opus only: about 1,200 turns and 230 threads. - Fable until its limit, then Opus: about 900 turns but slightly more threads. He says Fable is more likely to finish work in one prompt [6].

He based this on more than 1.1 million real API requests [7]. He warns the numbers may shift, because much of his usage so far has been benchmarking the new model [6].

Don't reuse your old effort setting. Cursor says effort levels behave differently from 5.1. Start at the default and compare levels at cursor.com/evals [2]. Theo reports that on the code-only benchmarks in Slopalytics, xhigh and max score *worse* than high. Fable's overall 5-point lead also shrinks a lot on code benchmarks [8, 9]. Alex Albert gives similar advice: start at medium or high and save xhigh for the hardest problems [10].

Prompt tip: tell it the code is its own. Anthropic's Fable 5.2 prompting guide, quoted by Simon Willison, says the model adds to existing structure, even repeating code, unless told the code is its own. The suggested sentence is: `This codebase is yours: what it's for and how it's built are your call, and you'll keep building on it.` On small coding tasks, it merged repeated code in about half of tries with this sentence and under a tenth without it [11].

Alex Albert's workflow tips [10]: - Before it builds anything, ask it what's wrong with your idea. It will push back. - Say up front what needs your approval, such as deletions, and that the rest is its call. - Give it a whole project rather than the next step, and let it run for hours.

He also runs one audit prompt on every project whenever a new model arrives: *"Read this entire project and find everything that's broken, inconsistent, inefficient, or no longer needed. Fix what you're sure about and list the rest for me to review."* [12]

Fable or Opus 5.5? Most early advice says to keep Opus as the default. - **Osmani:** start most coding work on Opus 5.5 and use Fable for ambitious, open-ended work [3]. - **@TheWholeDaisy:** use Opus when you already know what you want, and Fable for creative tasks where you don't [13]. - **Theo's 3D-game test:** Fable took 76 minutes and about $42. He rated it "not significantly better than Opus 5.5 (esp. for the price)" [14]. The Opus version had a better gameplay loop, controls and performance, and he thinks the two models will work well together on 3D-game tasks [15].

Simon Willison: let the agent see and change the running app

To find a beep among unopened boxes, Willison built Beep Locator (repo). It located the source to within about 40 cm: a carbon monoxide detector that needed new batteries [16].

His steps: 1. **Start from a spec.** In Claude Code Cloud on his iPhone (Opus 5.5, Extra effort), with an empty repo, his first prompt was "Write out a problem.md file describing the problem." Claude found and read a 2007 paper on acoustic ranging while writing the spec [16]. 2. **Ask for hot-reload, not just**

the feature. He told it to "add a mechanism so you can send new code to the listeners which they will then run automatically", so it could keep experimenting while he did something else [16]. 3. **Iterate live.** On his laptop, Claude Code with Fable 5.2 started the server, could see what was happening, and could change the software as it ran. An `installModule()` hook pushed code to the listening devices over WebSockets. That avoided page reloads, which on some devices re-trigger microphone permission prompts [16].

Handing routine work to agents

Pi issue triage. The Pi team was sorting 30–100 issues a day by hand. They built an issue-triage bot on Pi Durable. Sub-agents trigger on each new issue, do a first analysis and a first fix, and a maintainer approves the diff [17]. A small classification guide updates as humans sign off on processed issues. They say the bot now matches their taste using past examples, which probably wouldn't have worked six months ago [17].

Kent C. Dodds' state-machine prompt: "Find all of the implicit state machines in my app… elevate… UX, DX, and agent experience (AX)… without breaking anything. Also, make state machines explicit." [18]

Romain Huet (OpenAI) says skills or AGENTS.md files untouched for about four months are "probably far outdated" [19].

Running agents on other machines

- **Dots (OpenAI):** you can now connect several local computers to one dot and choose which handles a task. For example, it can build and test on a dedicated machine while you keep coding on your laptop [20].
- **T3 Code:** press cmd+k and search "mcp" to set up a connector for any agent you want to control T3 Code from [21]. Theo's tip if you hit Grok Bot usage limits: get the $100 Claude plan, install T3 Code, set up that connector, and tell Grok Bot to "call T3 Code with Opus 5.5 for any task where it is applicable" [22].
- **Brendan Long's sandbox:** he runs Claude Code with all permission prompts disabled (clawed-abode) [23]. To contain the risk, agents run as an unprivileged user inside an unprivileged VM. They use rootless Podman instead of Docker, because Docker access easily gives root [23]. They reach the GPU only through a queue over SSH into a gVisor + nvproxy container. Its allowlist of NVIDIA driver calls reduces exposure to driver exploits, though imperfectly [23].

Sources

1. X post by @cursor_ai
2. X post by @cursor_ai

3. X post by @addyosmani
4. X post by @addyosmani
5. X post by @theo
6. X post by @theo
7. X post by @theo
8. X post by @theo
9. X post by @theo
10. X post by @alexalbert___
11. Quoting Prompting Claude Fable 5.2
12. X post by @alexalbert___
13. X post by @TheWholeDaisy
14. X post by @theo
15. X post by @theo
16. Locating the source of a mysterious beep with Claude
17. Ep 08 - "Nobody's Going to Write the Next React"
18. X post by @kentcdodds
19. Deep Dive 220 – Developer Experience at OpenAI: Romain Huet on building for agents
20. X post by @OpenAIDevs
21. X post by @theo
22. X post by @theo
23. My Paranoid Coding Machine Setup