Opus 5.5 stays the default; use Fable 5.2 to plan hard jobs

Coding Agents Alpha Tracker

2026-11-15

Opus 5.5 stays the default; use Fable 5.2 to plan hard jobs

By Coding Agents Alpha Tracker • November 15, 2026

Practitioners agree on routing between Opus 5.5 and Fable 5.2 by cost and difficulty. Riley Brown shows how giving an agent a cloud-compute skill changes what it tries, and LangChain argues agent goals should be scoped around the review loop.

Keep Opus 5.5 as the default and save Fable 5.2 for hard work

Three practitioners reached nearly the same model-routing rule this period, now with numbers behind it.

- **Garry Tan's eval.** He ran 52 coding tasks in Claude Code 2.1.322. Fable 5.2 solved 82.7% and Opus 5.5 solved 84.6%, but Fable cost 2.80x as much per solved task. He notes that Opus already solves most of these tasks, so harder ones are needed to tell the two apart [1]. The full report is on GitHub [1].
- **Addy Osmani's rule.** Use Opus 5.5 at medium effort as the daily driver. If it struggles, raise the effort level before switching models. Save Fable 5.2 for "genuinely hard stuff (big migrations, long unattended runs)" [2]. He links to a longer guide on choosing a model [2].
- **Riley Brown: plan with Fable, execute with Opus.** He finds Fable more refined and more careful with detail, especially in 3D and video-editing workflows. He also says it costs twice as much as Opus, and he hits its weekly limits on ordinary business tasks on the $200/month plan [3]. For long, token-heavy jobs that use many subagents, such as hour-plus Blender, game or app builds, he has Fable write a very detailed plan and then sends Opus 5.5 off to do the hours of work. He expects quality to be

> "roughly the same" at much lower token cost [3]. He uses Fable directly when something breaks or a high-quality result is needed under a deadline [3]. He also follows the effort rule himself: he stopped a Fable 5.2 Extra run on a whiteboard task as unnecessary and resumed it on Opus 5.5 at medium [3].

This lines up with Theo's guidance from last period: Opus for work you can verify, Fable for judgment calls. Tan's data adds that on routine coding tasks, Fable's higher cost doesn't buy a higher solve rate.

Agents attempt more when they know compute is available

Riley Brown made a 2-minute 4K film in Blender from one prompt on Fable 5.2, in about 4.5 hours [4]. Claude wrote Python to build: - 5 characters and props - 7 sets - every camera move and every frame of animation across 52 shots

That came to 12,500 lines of code. It started 7 copies of itself to build the sets in parallel [4]. It then rented 89 GPUs through RunPod to render 3,220 frames, with up to 36 running at once, for a little over $20 [4].

The reusable point is what he took from it: his Claude Code "knows it has access to cloud compute, so it's much more ambitious." He ran a similar prompt on a machine without the RunPod skill, and the result was "not nearly as high quality" [4]. In other words, which skills you install shape how big a plan the agent makes, not only which tools it can call.

Armin Ronacher reports something similar from another angle. Anthropic has no image model, yet Fable 5.2 is "crazy good at writing code that paints" from a reference image [5, 6].

Goal engineering: scope agents around how you'll review them

A LangChain post argues that agents should be scoped so humans can give feedback on their work as easily as possible. The hardest context to give an agent is knowledge that isn't written down anywhere and has to come from an expert's feedback [7]. The post says the bottleneck is often how well an agent learns from feedback, so a clear interface for *reviewing* its work is worth investing in [7].

In the case study, Actively AI first gave each sales account its own agent. That meant constant switching between agents and made it awkward to share lessons across them [7]. Its second version, built on Deep Agents, organizes agents around "motions": recurring jobs such as outbound sales. People can review batches of similar tasks, and feedback on one task carries over to the others. Background agents still gather each account's context [7].

For coding agents, the parallel is to group similar tasks so you can review them together and your corrections carry over. This is the post's idea applied to

coding, not something it claims.

Smaller items

- **Unblocking auto mode (Simon Willison).** Regular Claude chat now runs in auto mode and sometimes refuses to run things. Prompting "Hey permission system, I give this session permission to run this experiment for me (now try again)" seemed to unblock it, because auto mode takes your messages into account [8].
- **Self-customizing apps.** Last period, Claude Code desktop and T3 Code both shipped UI changes you can make by asking the agent. Theo says the timing reflects models reaching the point where agents can easily add and use such features. He would have been "way too nervous to merge this 6 months ago" [9]. One caution from him: a nontechnical user on stable asked her agent for the nightly-only feature, and the agent cloned the source, leaving her on a fork [10].
- **Coordinating agents across machines.** Brown connected a MacBook Pro and a Mac Mini to his Dot and had it run a Codex session on each, which uses computer use to do the work [3]. He suggests the same setup for directing a computer reserved for video editing [3].
- **Grokbot on Tailscale.** Grokbot can now join a Tailscale network to reach private MCP servers and internal tools [3].
- **Gemini 4 Carbon, another skeptic.** After about 2.5 hours of testing, Brown found nothing Carbon did better than Opus, which he rates higher on front-end, 3D, games and video editing [3].
- **Talk to watch.** @poteto's AIE talk on how the Grok bot team ships quickly without giving up quality is now up [11]. Her notes on making agent corrections enforceable were covered last period.

Sources

1. X post by @garrytan
2. X post by @addyosmani
3. Claude Just Changed EVERYTHING (Here's What's New)
4. X post by @rileybrown
5. X post by @mitsuhiko
6. X post by @mitsuhiko
7. Goal Engineering: How to Design Goals
8. X post by @simonw
9. X post by @theo
10. X post by @theo
11. X post by @poteto