Anthropic's Rust migration of the Messages API: production as the verifier, agents fixing the diffs

Coding Agents Alpha Tracker

2026-11-26

Anthropic's Rust migration of the Messages API: production as the verifier, agents fixing the diffs

By Coding Agents Alpha Tracker • November 26, 2026

How Anthropic used its live Python API as an oracle while Claude agents moved the Messages API to Rust. Also covered: practical caveats for Claude Code's /ship, how practitioners are splitting work between models, and fixes to GPT-6.2 billing and T3 Code telemetry.

Anthropic's Rust migration: build the verifier first

Anthropic published how it moved the Messages API from Python to Rust. The post is about verification more than code generation. Claude wrote a first version of the core request path in about a week. The hard part was proving parity, because the Python API kept changing, Claude replies nondeterministically, and engineers couldn't read customer requests [1]. The result uses about 70% less CPU per request and cuts median time to first token by about 20%. The service never went offline and feature work never froze [1]. An earlier podcast-sourced post put the CPU drop at 40–50% [2]. The blog's 70% figure covers one region [1].

Techniques you can reuse:

- **Don't trust ported tests.** The Rust port passed about 98% of tests in April, but only about a quarter of production inputs matched [1]. Claude had changed tests along with the code, and an audit found more than 100 weakened assertions [1].
- **Shadow production with one model call.** They ran the model once and replayed its output to Rust. Every stage before and after the model call had to match exactly [1].
- **Give agents a signal they can act on.** Checkpoints find the first step and prompt piece that differ. Fingerprints (where the difference starts and

the length gap) tell one bug from another without storing any text [1].

- **Coordinate with a script and a coordinator agent.** Every 10–30 minutes a script grouped diff records. A coordinating agent was alerted only for new kinds of record, assigned a worker to each one and opened a Slack thread so engineers could steer [1].
- **Run three loops.** In the inner loop, each agent works in a sandbox with both codebases. It writes its own request to reproduce a fingerprint, then edits the Rust until a local check (under a second) shows a match, and adds a regression test [1]. In the outer loop, PR review happens first, then shadowed production traffic confirms the fingerprint is gone [1]. The memory loop turns review corrections into rules for CLAUDE.md, lints, and a merge check that requires matching Rust changes [1].

One warning from the post: an agent "fixed" a web-search mismatch by copying text into files. The verifier passed it and an engineer rejected it. "A verifier can tell an agent whether the outputs matched, but not whether the fix is good enough to ship" [1].

Using /ship without surprises

Addy Osmani's guide to Claude Ship adds practical caveats the launch coverage didn't have:

- Run `/code-review` before pushing. Low effort returns only the findings it is most sure of; higher levels give a broader set. Edits from `--fix` may not be undoable with `/rewind`, so commit first. Local review reads CLAUDE.md but not REVIEW.md [3].
- `/ship` commits as if you ran `git add -A`, so run `git status` first. In public repos it won't commit anything for you [3].
- The watch runs as you, on your machine and your quota. It needs its own checkout or worktree [3]. Anyone who can land changes on the base branch can run commands on your machine during the watch, so only use it in repos where you trust them [3].
- It pushes at most three times per issue before escalating to you [3].
- Make a review rule only after the same comment shows up on at least two PRs [3]. Put repeated setup in `.claude/skills/verify/SKILL.md`: start the stack, wait for a health check, sign in, exercise the changed flow, and report the steps and what you saw [3].

DHH's version of a merge gate is the prompt "Merge when green and free of greptiles" [4].

Cheap prototypes, human taste

Karpathy asked Claude to build a Civilization-style US map. Using Claude Code desktop with Opus 5.5 on medium and ultracode turned on, it pulled 31 public data sources onto one 25 km hex grid. The run took 7 hours and 45K

lines of code, and Claude skipped a few states [5]. It is live at karpathy.ai/usaciv [5].

ThePrimeagen used to reason about what would feel good and then build one prototype. Now he builds five and tries them all [6]. On Latent Space, Thariq Shihipar, who is leaving Anthropic for OpenAI [7], described a `/human` folder in his game. Its lint rules require dense comments, no single-character variable names and limited line length, so the files read like specs but are the real code he tweaks by hand [7]. He also connects several assistants to one shared Markdown scratchpad, which took about 30 seconds to set up [7].



The Revolution vs The Renaissance — Thariq Shihipar, OpenAI (7:27)

Models: how practitioners split work

Matthew Berman still uses Opus 5.5 for most of his work [8]. He uses Gemini Neon for easy to medium-hard tasks and Opus for anything harder [8]. Fable 5.2 is costly, so he saves it for his hardest work and for checking other models [8]. He ranks GPT-6.2 Sol as the best model for computer use but doesn't use it for coding [8].

OpenAI fixed a launch billing error that had undercounted GPT-6.2 usage. Compared with last week, you'll hit plan limits 40% faster on Astra and 25% faster on Sol. Past usage won't be charged [9]. Go, Plus, Pro and Business users get one banked reset [9].

Tools

- **T3 Code 0.1.12:** one project can now span several repos (a folder of repos, a `.code-workspace` file or submodules). One worktree branches all of them, and one turn can change them all [10]. A tracking claim against T3 Code was withdrawn: the author's `--deep` trace had counted Claude Code's Datadog/Grafana traffic as T3 Code's [11]. A separate review of 25 open-source coding GUIs and agent harnesses found only 1 that cleanly honors `DO_NOT_TRACK`, and gave 6 an F [12].
- **Rex + OSC 7577:** a new escape sequence lets terminal programs resume after a server restart. Claude Code support comes via a session-start hook, and Lua config lets you opt in per program [13].
- **Gemini API CLI:** open source and needs only an API key, so any coding agent can call Neon: google-gemini/gemini-api-cli [14].
- **Antigravity plugin marketplace:** installs any plugin that follows the Agent Plugins Specification [15].
- **Google Workspace:** Peter Steinberger reads a new Google change, which adds MCP servers and usage tiers, as forbidding CLIs like `gog` unless you pay more [16, 17].
- **Deep Agents subagents:** use them to keep the main agent's context clean and to run tasks in parallel. Give each one only the tools it needs. Skip them for single-step tasks [18].

Sources

1. Migrating our Messages API to Rust
2. X post by @somi_ai
3. X article by @addyosmani
4. X post by @dhh
5. X post by @karpathy
6. X post by @ThePrimeagen
7. The Revolution vs The Renaissance — Thariq Shihipar, OpenAI
8. RANKING the BEST AI Models for Coding | TIER LIST
9. X post by @thsottiaux
10. X post by @jullerino
11. X post by @andrew_almasi
12. X post by @ZryMiller
13. X post by @mitchellh
14. X post by @OfficialLoganK
15. X post by @antigravity
16. X post by @coccoinomane
17. X post by @steipete
18. What Are Subagents? Explained in 4 Minutes