Gemini 4 Neon reaches Cursor and Antigravity; Grok Bot's harness lead explains how to cut agent cost

Coding Agents Alpha Tracker

2026-11-19

Gemini 4 Neon reaches Cursor and Antigravity; Grok Bot's harness lead explains how to cut agent cost

By Coding Agents Alpha Tracker • November 19, 2026

Gemini 4 Neon is now available in Cursor and Antigravity, and early reactions sharpen the debate over GPT-6.2 Astra. Also covered: Jediah Katz's cache and context tradeoffs for agent harnesses, LangSmith's measure of cost per merged PR, and advice on routing tasks across models.

Gemini 4 Neon arrives, and model-splitting becomes the default pattern

Google's Gemini 4 Neon is now available in the main coding harnesses. **Cursor** reports that at default effort it scores 63.7% on CursorBench 4.1, up from 57.3% for Gemini 4 Hydrogen, and calls that a new cost-performance frontier. Pricing is $3/M input and $15/M output tokens [1]. **Antigravity** is rolling it out across the desktop app, CLI and IDE extensions for paid AI Pro and Ultra subscribers, and says it ranks #1 on the Artificial Analysis Coding Agent Index. Antigravity's Orchestrator can pair **Neon as planner with Hydrogen as executor**, so Neon's judgment goes "where it matters and Hydrogen's speed everywhere else" [2]. One Google launch post claims Neon is cheaper, faster and "better behaved" than Carbon, and beats Opus 5.5 and Sol 6.2 on most benchmarks [3]. Those are vendor claims.

Theo started out skeptical. By the end of the day he called Neon a "weirdly perfect counter balance to GPT-6.2 Astra… so much more pleasant to work with" [4]. His current T3 Code setup gives each model a role, each running in its own

harness on its own subscription [5]: - **Fable** leads implementation - **GPT-6.2 Astra** playtests - **Gemini 4 Neon** writes all the copy

He explains the split with a distinction worth borrowing: **capability** ("at its best, what can it do") versus **coherency** ("does it understand what it's doing"). He says model families "have never felt so different" [6].

The Astra debate gets a concrete test case

Theo retested Astra after Tibo (OpenAI) said early testers had missed recent changes. His test question: why is creating a worktree slow on one VPS when two others are fast? The cause was a slow virtual disk, where the other machines had physical NVMe [7]. Opus opened with "Slow disk, made worse by VDO." After the fixes, Astra was more concise but still led with second-order details [7]. His conclusion: Astra "doesn't miss anything" but misses "the actual point" [7].

The takeaway for anyone evaluating debugging agents is to check whether the agent puts the dominant cause first, not just whether it is thorough. Theo says the system-prompt changes "helped a lot," but Astra still loses coherency after a few hours on long tasks [8] and the improvements don't change his view [9].

Mathematician @nasqret, an early tester, has a different and more positive workflow: - He runs Astra in **CLI agent mode** with "no YOLO mode," watching every instruction it runs. - He starts from a very specific task driven by his custom "labyrinth" skill, then lets it run 1–4 hour research cycles with "Carry on" [10]. - Astra won't set up multiple agents on its own, but will if asked, and can create communication channels between them [10]. - He uses Lean to formalize the output. Astra points him to the hard proof steps to check [10]. - He uses Sol as the faster, cheaper everyday agent [10].

Browser Use separately reports GPT-6.2 Sol as its top browser-use model: 82% (vs 81% for 6.1 Sol), 35% cheaper, 12 rather than 22 minutes, and 36% fewer steps [11].

Harness efficiency: what Grok Bot learned

Jediah Katz worked on Cursor's harness from early 2025 and now works on Grok Bot's. He says Grok Bot is 3x faster to reply and 2x more token-efficient since launch [12]. The levers he describes apply to your own agent setups as well: - **Keep the prompt prefix stable.** Put anything volatile as late in the context as possible. A timestamp near the top causes a full cache miss [12]. - **Longer cache TTL** gives more cache hits but costs more per token [12]. In A/B tests, it paid off for Grok Bot's async coordinator, whose cache expired while it waited idle on subagents. It did not pay off for subagents that work continuously [12]. - **Every lever has a tradeoff.** Compacting earlier, or moving tools into skills or a tool search, cuts context but adds steps. Cheaper models stumble more. Parallel tool calls and subagents cut steps but use more tokens [12]. - **Hold**

quality to a bar. Profile first, A/B test cost and speed, and confirm quality still meets the bar you set [12].

Measuring agent spend by merged PR

LangSmith for Coding Agents (public beta on LangSmith Cloud) attributes trace spend to pull requests, including threads that open several PRs and PRs that take several threads. It supports Claude Code, Codex, Cursor, Copilot in VS Code and Deep Agents [13]. To set it up, install the plugin for each agent, connect GitHub and open the coding-agent overview page [14].

LangChain's usage advice applies with or without the tool: - **Start fresh threads between tasks.** Each new turn carries the whole conversation context [13]. - **Judge model changes by cost per merged PR**, including moving subagents to smaller models [13]. - **Check skills that keep appearing in long, expensive threads.** Their instructions may be causing extra turns [13]. - **Don't count all non-PR spend as waste.** Review and investigation often don't end in a PR [13].

Routing and review across models

At Interrupt NYC, OpenRouter's Alex Atallah sorted tasks by entropy: - **Low:** for example, choosing who should review a PR. No frontier model needed [15]. - **Medium:** PR review. Mostly open-weight models, with occasional frontier-model checks [15]. - **High:** building new features. Use a frontier model [15].

He says writing with Opus and checking with GPT-5 noticeably improved code quality, while having Opus check its own work was "much, much worse" [15].

Harrison Chase recommends evaluating long-running agents on their whole trajectory and final environment state, not just the final answer. He suggests Harbor for sandboxed container runs [16]. Turn unexpected failures into regression tests and rerun them whenever you change the prompt, model, context or harness [16].

Practitioner notes

- **Simon Willison** on choosing models: pick a good default model family and ignore roughly 3% gains. Watch instead for new capabilities, such as 3D/Blender work [17]. He once asked Opus 5.5 to make a game fun; it worked for 12 hours in Claude Code, ran 1,000 simulated matches, and made it fun [17]. With the easy work delegated, he says every decision left to him is a hard one, and he has "never worked harder" [17].
- **Rewrites go stale.** Daniel Stenberg says that while agents converted 185,000 lines of curl's C into 340,000 lines of Rust, curl merged 200+ bug fixes and some vulnerability fixes that the rewrites don't have [18].
- **Limit what unattended agents can do.** Armin Ronacher's overnight AirPlay reverse-engineering agent decided at 3 AM to verify its work by

turning a living-room speaker to max volume and listening through the laptop mic [19].

Sources

1. X post by @cursor_ai
2. X post by @antigravity
3. X post by @_mohansolo
4. X post by @theo
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 @nasqret
11. X post by @browser_use
12. X article by @jediahkatz
13. LangSmith for Coding Agents: Now track spend by outcome
14. X post by @LangChain
15. Fireside Chat With Alex Atallah, OpenRouter
16. Harrison Chase: what changes from 1 agent to 10,000
17. Simon Willison: AI Agents Made Me More Productive. So Why Am I Working Harder?
18. Quoting Daniel Stenberg
19. X post by @mitsuhiko