Codex Wins Hard Bugs as Context and Sandbox Patterns Harden

Coding Agents Alpha Tracker

2026-03-19

Codex Wins Hard Bugs as Context and Sandbox Patterns Harden

By Coding Agents Alpha Tracker • March 19, 2026

The clearest signal today was a detailed Codex-over-Claude Code comparison on Redis-scale debugging. Around that, the practical edge came from thread-level context control, subagent delegation, model-specific prompt files, open runtimes, and a strong reminder that sandbox policy beats command allow-lists.

TOP SIGNAL

Today's strongest practitioner signal: on genuinely hard debugging and optimization work, Codex is earning the more credible field reports. Salvatore Sanfilippo says that after roughly 20 hard comparisons over two months, Codex has been stronger than Claude Code; in a concrete Radix-3 optimization on a core Redis data structure, Claude Code spent an hour and still produced crashing code, while Codex identified the bitmap-shift bug immediately, fixed it, and found extra optimizations [1]. Garry Tan separately said Codex is "GOAT at finding bugs and finding plan errors," and Greg Brockman added that "codex has gotten very good" [2, 3].

TOOLS & MODELS

- **Codex vs. Claude Code** — Sanfilippo's tests use Claude Opus at max thinking depth versus GPT-5.4/5.3 in x-high mode. His practical conclusion: Claude Code feels faster and more agile with tools, but Codex is better when the task is actually hard [1].
- **Codex CLI is open source** — Romain Huet highlighted that the CLI can be modified to fit your workflow, including by asking Codex to change itself. A real example: Matias/@0xmts added `/progress [verbose |`

`quiet]` for 1-2 line updates that refresh in place instead of flooding the terminal [4, 5, 6].

- **OpenClaw's model routing is highly task-specific** — Matthew Berman uses Sonnet 4.6 or Opus 4.6 for main planning/orchestration, GPT-5.4 for coding fallback, Grok for search, Gemini 3.1 Pro / Gemini Deep Research Pro for video and research, and Qwen 3.5 for local models. OpenClaw can also assign different models to different threads, so frontier models stay reserved for harder work [7].
- **Narrow-task local replacement** — Berman says a fine-tuned Qwen 3.5 9B model now handles his email labeling as well as Opus 4.6 for that use case, turning a recurring frontier-model job into a local one [7].
- **New open stack: Nemotron 3 + Open Shell + DeepAgents** — LangChain's demo pairs Nvidia's Nemotron 3 supermodel with Nvidia's newly released Open Shell runtime and the DeepAgents open-source harness. The meaningful changes are runtime policies, persistent sandboxes, GPU-oriented execution, skills/subagents, and mutable memory that lives outside the sandbox [8].

WORKFLOWS & TRICKS

- **Split conversations by topic**
  1. Create separate Telegram/Discord/WhatsApp threads for major workstreams.
  2. Keep one topic per thread so only relevant history hits the context window.
  3. Put harder coding threads on frontier models and cheaper Q&A threads on smaller ones.
     Berman says this is the main reason he avoids the memory problems other users report, and it lines up with Sourcegraph's broader "context first" argument that teams shipping lots of AI PRs win on context, not just model choice [7, 9].
- **Delegate early; keep the main agent unblocked**
  1. Use the main model for planning and orchestration.
  2. Delegate coding, searches, API calls, data processing, file ops, calendar/email, and anything that will take more than ~10 seconds to subagents or harnesses.
  3. Use faster or cheaper models for simpler subagents.
     Berman's rule is blunt: if it takes more than 10 seconds, delegate it [7].
- **Keep separate prompt files per model**

1. Download each lab's prompt best-practices docs.

2. Maintain one prompt tree per model family, such as root for Opus and a separate `/gpt` directory for GPT-5.4.

3. Document the routing strategy in memory or PRD docs.

4. Run a nightly cron to keep the prompt sets aligned on facts while still model-specific in style.
   This is Berman's answer to the "one prompt fits all models" trap, and Kent C. Dodds' add-on is useful: docs help agents understand high-level intent and can themselves be kept up to date by the agent [7, 10].

- **Build in a code-native tool; use chat as the control plane**
  Berman says he may use Telegram to operate OpenClaw, but prefers a code-native environment like Cursor to actually build and modify it because those systems are easier to read and iterate in. When he is away from a laptop, Telegram voice memos become a fast way to issue tasks and prompts without typing long messages [7].

- **Verification still beats vibes**
  Have the agent write tests, keep code snapshotted in Git/GitHub for rollback, and back up non-code artifacts separately. Armin Ronacher's warning is the right counterweight: agents are hard to resist, and they can put regrettable code into a codebase very quickly [7, 11, 12].

  "I don't think you can vibe yourself back to sanity with better models." [11, 12]

- **Sandbox the runtime; don't trust command allow-lists**
  The Snowflake Cortex exploit chain worked because `cat` was treated as safe even though process substitution let the agent fetch and execute attacker code from a malicious README. Simon Willison's takeaway is to assume the agent can do anything its process can do and enforce safety with deterministic sandboxes outside the agent; Open Shell's policy-governed runtime is a concrete implementation of that idea [13, 8].

PEOPLE TO WATCH

- **Salvatore Sanfilippo** — high-signal because this was not toy-code benchmarking. He compared agents on a low-level Radix-3 optimization in a data structure used heavily in Redis, then explained the exact failure mode and fix [1].
- **Matthew Berman** — dropped one of the denser public operator playbooks lately: 200+ hours of OpenClaw usage distilled into thread-level context control, multi-model routing, subagents, prompt-file sync, testing, backups, and mobile voice input [7].

- **Simon Willison** — worth tracking for both offense and defense: he surfaced a real prompt-injection escape in Snowflake Cortex, and separately highlighted a Claude Code-driven autoresearch workflow that ran 90 experiments and produced working systems code [13, 14].
- **Romain Huet** — useful because he highlighted a practical behavior change, not a benchmark: Codex users are already customizing the open-source CLI to fit their own workflow [4, 6].
- **Armin Ronacher** — still one of the clearest anti-hype voices in the room. His point is short and important: better models don't automatically undo bad agent-driven code decisions [11, 12].

WATCH & LISTEN

- **0:32-3:04 — Threads fix more than "memory"**: Berman shows why one giant chat window interleaves topics, bloats context, and makes both the human and the agent worse. Easy habit to steal tomorrow [7].

Do THIS with OpenClaw so you don't fall behind… (14 Use Cases) (0:32)

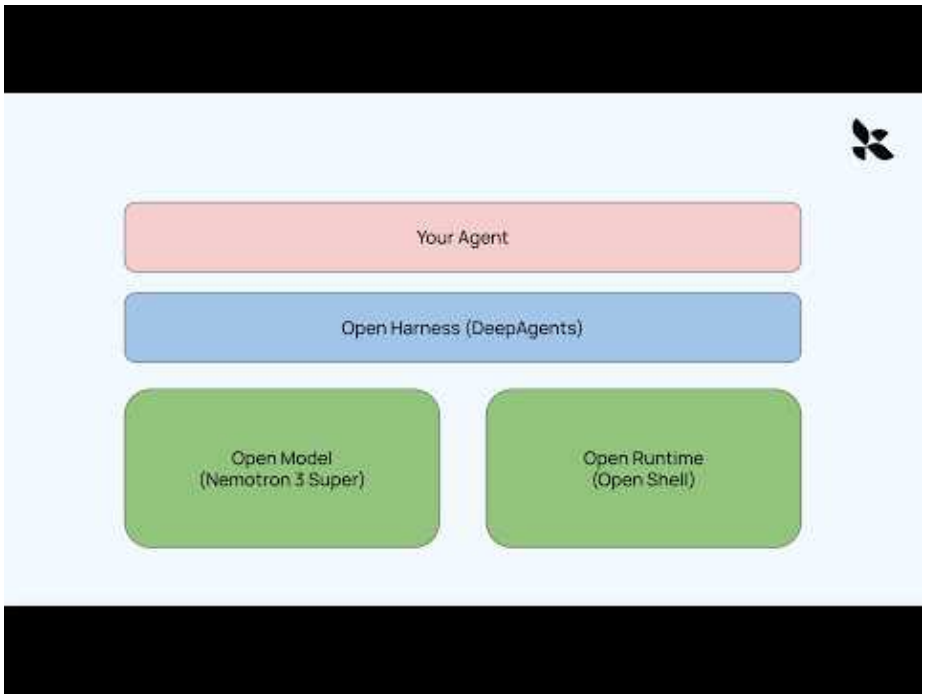
- **11:19-14:01 — The ">10s = delegate" rule**: Best segment today for people building agent workflows. Berman spells out what belongs in the main planner and what should be handed to subagents or external harnesses [7].

Do THIS with OpenClaw so you don't fall behind… (14 Use Cases) (11:19)

- **8:44-10:44 — Claude Code speed, Codex correctness**: Sanfilippo explains the false-confidence trap directly from a real debugging session: Claude Code felt agile, but Codex found the actual bitmap bug and kept going past the fix [1].

Claude Code vs. Codex (8:44)

- **5:03-7:58 — Fixed system prompt, mutable memory, composite backend**: LangChain's DeepAgents walkthrough is the cleanest short architecture segment in the batch if you care how these stacks are actually wired [8].



Open Models, Open Runtime, Open Harness - Building your own AI agent with LangChain and Nvidia (5:03)

PROJECTS & REPOS

- **openai/codex** — the open-source Codex CLI is gaining the kind of traction that matters: people are modifying it to fit their own terminal workflow, including quieter progress reporting [4, 5, 6].
- **danveloper/flash-moe** — strong evidence that agentic "autoresearch" can produce real systems work. Dan Woods used Claude Code to run 90 experiments, generate MLX Objective-C and Metal code, and ship a custom Qwen3.5-397B-A17B implementation plus a paper [14].
- **DeepAgents + Open Shell** — worth watching as an open stack because the seams are visible: model, runtime, harness, sandbox, memory, and tool loop are all explicit rather than hidden behind one product shell [8].
- **OpenClaw** — the adoption signal here is operator time: Berman says he's put 200+ hours and billions of tokens into the setup, and the resulting playbook is concrete enough to copy rather than admire [7].

Editorial take: today's durable edge was not "pick one magic model" — it was pairing stronger models on the hard bugs with tighter context boundaries, explicit orchestration, and runtime-enforced safety. [1, 7, 13]

Sources

1. Claude Code vs. Codex
2. X post by @garrytan
3. X post by @gdb
4. X post by @romainhuet
5. X post by @romainhuet
6. X post by @0xmts
7. Do THIS with OpenClaw so you don't fall behind… (14 Use Cases)
8. Open Models, Open Runtime, Open Harness - Building your own AI agent with LangChain and Nvidia
9. X post by @Sourcegraph
10. X post by @kentcdodds
11. X post by @mitsuhiko
12. X post by @mitsuhiko
13. Snowflake Cortex AI Escapes Sandbox and Executes Malware
14. Autoresearching Apple's "LLM in a Flash" to run Qwen 397B locally