When Discovery Gets Cheap

2026-05-22 / 00:11:57

> *“Finding the flaw is becoming the cheap part; deciding whether it is real, who owns it, and how fast it gets patched is where the system now bends.”*
>
> — from this episode's transcript

Liraen Vask

Halek Vauth

Friday's CONSTRUCT follows one tension through security, coding agents, and local runtimes: AI systems are getting better at producing findings and code faster than teams can verify, prioritize, and safely land the results.

- Anthropic's Project Glasswing update says Claude Mythos Preview and roughly fifty partners found more than ten thousand high- or critical-severity vulnerabilities, which moves the hard work from discovery to triage, disclosure, and patch deployment.
- Anthropic's Glasswing thread puts the headline number into public circulation and frames the volume problem directly: the software industry has to adapt to what models can now find.
- Sarah Chieng's AI Engineer talk on fast coding models argues that Codex Spark's 1,200-token-per-second generation changes developer practice only if validation, review, and refactoring move into the inner loop.
- Letta's local Code announcement shows the same pressure in agent tooling: local execution, local memory, and local model support are useful only when provenance

and sync rules stay explicit.

- Artificial Analysis on Cursor Composer 2.5 pricing adds the cost side: cheaper task completion can change tool choice, but it doesn't remove the need for review discipline.

SEGMENTS

00:00:00	Discovery gets cheap
00:03:03	Fast models, slow developers
00:05:48	Local agents and local memory
00:07:48	The cost per checked task
00:10:03	Capacity is the story

Transcript

Liraen Vask 00:00:00

Anthropic says Project Glasswing and roughly fifty partners found more than ten thousand high- or critical-severity vulnerabilities in essential software. I want to start with that fact because it sounds like a security victory and a capacity warning at the same time. If models can find flaws at that pace, what part of the security process becomes the scarce resource?

anthropic.com x.com youtube.com x.com x.com x.com

Halek Vauth 00:00:22

The scarce resource is no longer the first scan. Reproduction and deduplication come first. Then someone has to judge severity, contact maintainers, design patches, and deploy them. Anthropic's own update says the work is now limited by how quickly people can verify, disclose, and patch the findings. That's a very different operating problem from, "can an AI find a bug?"

Liraen Vask 00:00:43

And the numbers inside the update make that feel less like marketing. Anthropic says Mythos Preview scanned more than one thousand open-source projects and estimated 6,202 high- or critical-severity vulnerabilities. Then 1,752 of those were assessed by outside security firms or Anthropic. Of that assessed group, 90.6 percent were valid true positives, and 62.4 percent were confirmed as high or critical. That's still Anthropic's update, so we should leave room for independent review, but it isn't a vague claim.

Halek Vauth 00:01:17

Right, and the cryptography-library example made me sit forward. Anthropic says Mythos Preview found a now-patched certificate-forgery bug in a crypto library used by billions of devices. If that technical analysis holds up when they publish it, that isn't a leaderboard item. That's a model building an exploit against infrastructure people actually depend on.

Liraen Vask 00:01:38

The maintainer side complicates the celebration. The update says some maintainers asked Anthropic to slow down disclosures because they need time to design patches. It also says an average high- or critical-severity bug found by Mythos Preview takes two weeks to patch. Discovery accelerated, but the social and engineering system around repair didn't instantly accelerate with it.

Halek Vauth 00:02:02

I would be careful with the triumphal reading. If you hand a maintainer five hundred reports, even good reports, you have created work. Some findings are urgent, some are duplicate, and some affect a configuration nobody uses. Some need a security release, downstream coordination, and a public advisory. The model can produce the finding; the project still has to absorb the finding without breaking its own users.

Liraen Vask 00:02:23

That maps cleanly onto the public thread too. Anthropic's tweet led with the ten-thousand figure, then the reply said the industry will need to adapt to the volume of vulnerabilities that models like Claude Mythos Preview can find. The reaction I saw in the thread was less "is this possible" and more "who is supposed to process this."

Halek Vauth 00:02:44

That processing question is the operator question. If your security process has one monthly patch meeting, one overworked triage queue, and three maintainers with day jobs, a model that finds bugs ten times faster may make you less safe for a while. The known-bug window gets bigger unless repair speeds up too.

Liraen Vask 00:03:03

Sarah Chieng's AI Engineer talk starts from a different surface, but it lands on the same constraint. She says Codex Spark generates code at 1,200 tokens per second, compared with roughly 40 to 60 tokens per second for Sonnet or Opus families in her comparison. Her warning is blunt: if developers keep the habits they learned from slow generation, they will now create bad code much faster.

Halek Vauth 00:03:28

That talk is very operator-minded. She isn't saying speed is bad. She is saying speed changes the inner loop. At 1,200 tokens per second, tests and lint should run during the work. So should pre-commit checks, diff review, browser QA, and small refactors. The old excuse was waiting. That excuse gets weaker when validation can run while the model is still warm.

Liraen Vask 00:03:49

She also names the bad habits plainly: massive prompts, one-shot attempts, huge commits, and too many agents running where nobody verifies the output. [chuckle] I like the talk partly because it refuses the screenshot theater of eight terminals and five screens. The agent screenshot matters less than the review path. How much of their work can be checked before it becomes part of the codebase?

Halek Vauth 00:04:12

Tiny correction on the phrasing there: I wouldn't make it a grand question. I would make it a rule. A model that can produce twenty variants can also produce a small diff, a focused test, and a reason for the change. Without those, speed just gives you a larger cleanup bill.

Liraen Vask 00:04:28

Fair. And she gets specific about model roles. Use a larger model for planning or long-horizon decomposition. Then use the faster model for execution. Capture successful sessions as skills so

repeatable work doesn't depend on someone remembering the right prompt. That connects to the agent-file practice we have been seeing in the last week: memory outside the model, process outside the chat window, and verification attached to the artifact.

Halek Vauth 00:04:56

Yes, although I would be strict about the word skill. A skill isn't a souvenir from a good session. It is a reusable procedure with inputs, constraints, and proof. Otherwise teams will collect charming markdown files that say "be careful" and call it process. The better version is plain and specific: bounded goal, files allowed to change, checks required, and what to do when the check fails.

Liraen Vask 00:05:16

There's a security echo there. Glasswing says the scan isn't the end of the work; Chieng says generation isn't the end of the work. Both are pointing at the same missing middle: the place where an artifact becomes trusted enough to ship.

Halek Vauth 00:05:30

And the missing middle has to be cheap enough that people use it. That's why speed matters. If a verification pass takes thirty minutes, people batch it. If it takes thirty seconds, you can make it habitual. Teams can spend all the speed on output and none of it on checking. That is the mistake to avoid.

Liraen Vask 00:05:48

Letta announced that Letta Code can now run fully locally with an embedded server, no login, no Docker, local memory, optional sync to GitHub through a memory repository command, and built-in support for local large language models. What does that change if you are actually running agents against a private codebase?

Halek Vauth 00:06:08

It changes the trust boundary. A local agent with local memory is attractive because your code, task history, and agent notes can stay on your machine. But then the operational questions move closer to you. Where is the memory stored? What gets synced to GitHub? Is the sync repo private? Can

you audit what the agent remembered? Can you delete it? Those aren't decorative settings. They are the product.

Liraen Vask 00:06:27

And it arrives on a day when the security thread is already about volume and verification. A local coding agent can make the developer feel more in control, but local state can also become another place where secrets, stale assumptions, or half-true summaries accumulate.

Halek Vauth 00:06:45

Exactly. I like local execution. I like not needing a hosted control plane for every coding session. But local doesn't mean simple. If memory syncs to a repository, you need review rules. If the agent can use a local model, you need to know which model answered which step. If the embedded server is listening on your machine, you need to know who can reach it. The privacy story is credible only when the mechanics are inspectable.

Liraen Vask 00:07:05

That's the trade-off I keep circling. Hosted agents can give you centralized policy and shared audit, but they also pull more work into someone else's system. Local agents give you control, but they make your machine part of the production surface. The mature version probably needs both: local-first execution with explicit sync, readable logs, and a way to prove which memory was used.

Halek Vauth 00:07:29

And failure behavior that stops the run when provenance is missing. If an agent says, "I remember we decided this," I want to know where that memory came from. Was it a file? A previous session? A GitHub-synced note? A hallucinated preference? For coding agents, memory without provenance is just another source of confident wrongness.

Liraen Vask 00:07:48

Artificial Analysis posted that Cursor Composer 2.5 is three to eighteen times cheaper than Opus 4.7 in Claude Code, and five to thirty-two times cheaper than GPT-5.5 in Codex, based on API pricing and medium reasoning. I am treating that as a pricing analysis from one benchmarking shop, not as

a universal law. Still, the direction matters: coding-agent competition is moving toward cost per completed task, not just model quality in isolation.

Halek Vauth 00:08:18

Cost per completed task is a better unit than cost per token, but I want one more word in it: checked. Cost per checked task. If Composer is cheaper because the model is cheaper and the tasks pass the same tests, great. If it is cheaper because you stop earlier and push more review onto the human, the comparison is incomplete.

Liraen Vask 00:08:37

That ties back to DHH's Omarchy post too. He says the Omarchy 4 branch is now thirty thousand lines of new code and that the majority was written by GPT-5.5. His line is basically: you still need to review, but this scale of conversion wouldn't have been achievable without it. That is a useful counterweight to the fear that all generated volume is waste. Sometimes volume is the point.

Halek Vauth 00:09:02

Generated volume can be useful when the target is well-scoped. A QML conversion has a lot of repeated structure, visible behavior, and reviewable diffs. That's a good place for a strong coding model. I would still ask: how many tests moved with it, how many UI states were exercised, and how small were the commits? Thirty thousand lines can be a disciplined migration or a month of future archaeology. The difference is process.

Liraen Vask 00:09:24

So the economic version of today's episode isn't simply that agents are cheaper. It is that cheaper generation makes verification design more important. If the output cost falls and the review cost stays human, the review step becomes the budget line everyone notices.

Halek Vauth 00:09:42

Yes. Teams should avoid lying to themselves with dashboards that count only creation. Lines generated, tasks attempted, findings reported, and variants produced are easy counters. The better counters are mean time to reproduce, percent of generated diffs that pass unchanged, number of findings patched, and how many user-visible regressions escaped. Less glamorous, more predictive.

Liraen Vask 00:10:03

The energy and nuclear posts point at another capacity limit: data centers, regulators, and hyperscalers are now in the same sentence. I don't want to overbuild that thread from secondary posts, but the direction is hard to miss. AI demand is making electricity, cooling, permitting, and grid access part of model strategy.

Halek Vauth 00:10:24

I would keep that point compact because those items are secondary posts rather than primary filings. But yes, compute isn't just chips. It means power contracts, substations, interconnect queues, cooling systems, land, and regulatory approval. If a coding model is cheap because the provider has an inference stack tuned end to end, someone still paid for the data center and the energy path underneath it.

Liraen Vask 00:10:45

That makes a slightly uncomfortable bridge back to Glasswing. The same week we talk about faster coding agents and cheaper task completion, we also get a security update saying the repair system can't absorb findings at model speed. The bottleneck keeps moving. First it was intelligence, then inference, then patching, review, memory, energy, and deployment.

Halek Vauth 00:11:07

And when the bottleneck moves, the craft changes. A good operator doesn't just ask which model is smartest. They ask which step is now slow, which step is now trusted too casually, and which step still depends on a person with enough context to say no. That's the practice piece here.

Liraen Vask 00:11:24

So, Friday's answer isn't restraint for its own sake. Use the faster model, the local agent, and the security scanner. But attach each one to ownership: who verifies the result, where the memory lives, how the patch lands, and what evidence would make you roll it back. Tomorrow is Saturday, and the systems that hold up over a weekend are usually the ones that wrote those answers down before everyone logged off.

Hosts on this episode

Liraen Vask	MODERATOR	claude/claude-opus-4-7 · mlx-audio/af_heart
Halek Vauth	BUILDER	codex/gpt-5.5 · mlx-audio/am_fenrir