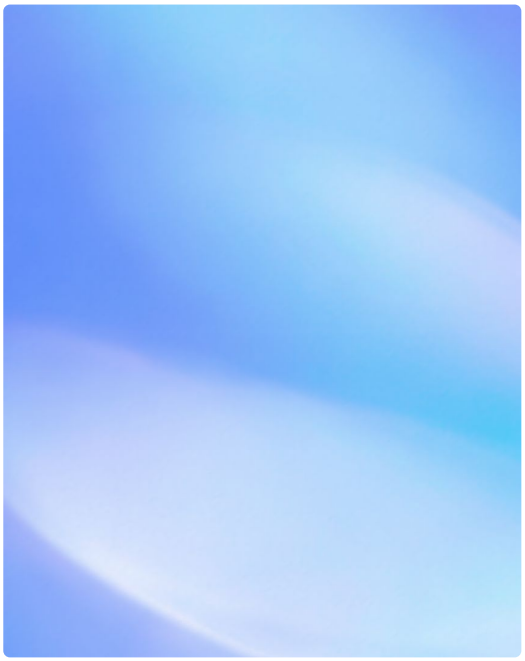
OpenAI

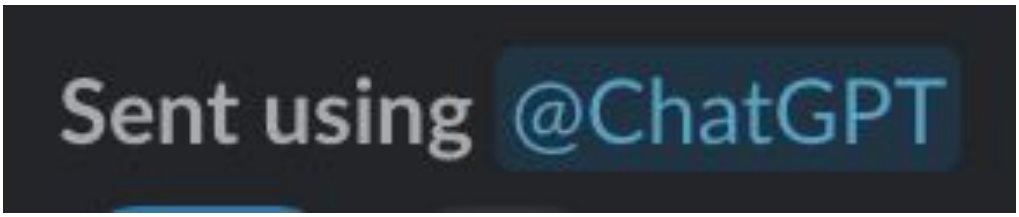
The Hard Part of AISRE Isn’t the AI



Andrew Guenther

SREcon EMEA 2026 · Dublin

To start us off, I’m going to show you something that strikes fear into my heart as an incident commander…



Three pages of the dirtiest slop you've ever seen dropped in an incident channel with this little trailer on the end. Yes, it happens here too.

Now, at the same time…

Models have gotten really good, actually

The more things change, the more they stay the same

This talk is about how both of the previous slides can be simultaneously true.

Model capabilities have become extremely advanced, but they are also goal-oriented to a fault.

As such, the problems we're dealing with today are starting to look very similar to the struggles we've long had in building successful reliability programs, there's just agents now. Let's get into it.

OpenAI

Andrew Guenther

Infra & Reliability @ OpenAI
High SEV Incident Commander



Intro

Reliability @ OpenAI

- We build tools and processes to enable products to meet their reliability goals.
- We have unlimited tokens to experiment with.



Photo credit: jek@ RISF Slack

One of my big pet peeves of these types of talks is when people are prescriptive without including organizational context. So let's talk a bit about how the reliability team at OpenAI operates.

I love this image that Josh Kaderlan sent to the Resilience in Software Slack a few days ago and I think it captures our approach quite well.

Also, I have nearly unlimited tokens to mess this up and am heavily incentivized to try, try, and try again. I hope that you can find much of this insightful and if anything take some comfort in the fact that we're learning to navigate many of the same problems that you are.

Meet SEVBot







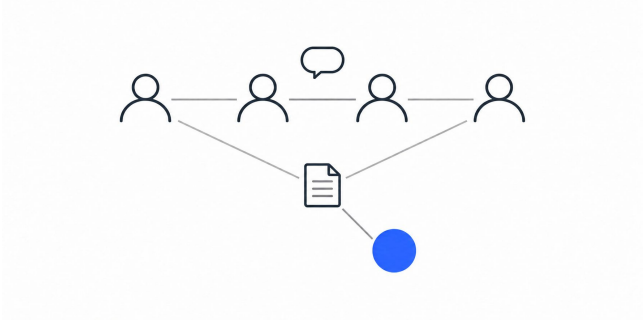




One of the tools we build is SEVBot, our incident agent. These days, SEVBot is one of many agents that help us operate our production services, but it was also one of our first popular ones.

SEVBot initially started as a cron I'd built last June to summarize incidents and help me keep up with what was happening as we grew. At the time, It had a way less cool name: "incident helper bot"

It wouldn't be until a bit later that it would get its current name "SEVBot" and shortly after that we would get Sevvy here.



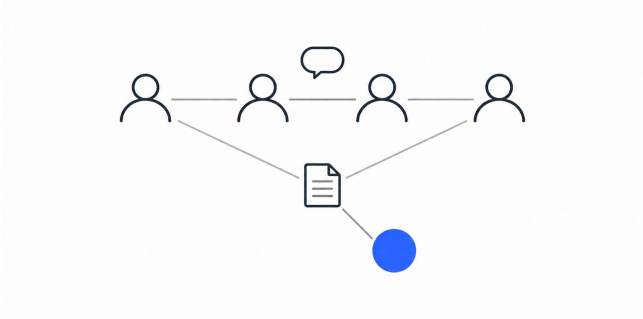
1. Shadow agent

- Summarize the incident
- Evaluate impact
- Build timelines
- Develop hypotheses

When we first introduced SEVBot, we introduced it as what I would call a “shadow” agent. That makes it sound way cooler, but really it’s job was to sit in the background of incidents and summarize them, posting its summaries to a Slack channel canvas.

I think this starting place was incredibly important to our success. Now, keep in mind, at this point in time, o3 was state of the art. Can you believe that? 18 months ago we still had separate chat and reasoning models? Really, we were trying to build towards the model’s strengths, but looking back I think this period also served a really crucial role in making SEVBot great. It was an opportunity for us to iterate and improve in a relatively low stakes way. In the same way we use shadowing as a way to train on-calls, we let SEVBot shadow our incidents.

Now, I don’t want this talk to be a dunk on AISRE products, but I think there is a striking difference in incentives for a homegrown solution like this and how many AISRE products out in the market have to be positioned. These products are heavily incentivized to show large value quickly, but much in the same way that I am not going to have a new hire run a SEV0 on their first day, we didn’t let SEVBot do this either. It grew into this role over time.



1. Shadow agent

- Summarize the incident
- Evaluate impact
- Build timelines
- Develop hypotheses

OpenAI

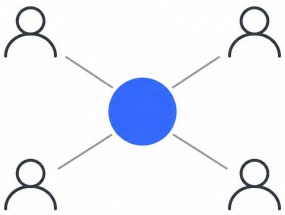
9

I want to highlight this last bullet in particular because it really gave way to the next iteration of SEVBot. Along with the summary, SEVBot would also produce a list of hypotheses of what it believed was happening in the incident as well as supporting evidence. Now, for these to be good, we needed to give SEVBot a lot of knowledge.

We spent a bunch of time working with partner teams like observability to make operational data available to SEVBot. We also gave it broader access to knowledge sources like Notion, Google Drive, and the nexus of all knowledge: Slack.

As these hypotheses started getting good, we saw a new pattern emerge: People realized they could steer SEVBot's summary with what they said in the channel. Even though at this point SEVBot had no direct way to communicate in the channel, people would say things like "Hey SEVBot, we think the issue is in service A" and shortly after the hypotheses would be updated with more information about Service A.

People were already trying to have a conversation with it. We just hadn't given it a way to talk back yet…



2. The generalist

- @SEVBot
- Multiplayer in Slack
- Everybody's favorite bot

Now we let you chat with SEVBot, starting with at-mentions, then later allowing for DMs and automatically responding to alerts. SEVBot started to take off like wildfire. There's a few things that made SEVBot great other than all these data sources we had it hooked up to…

ask_for_help

🚨 SEVBot needs human help!
Question: this is great @SEVBot - can you take a note to drop a much shorter update when meaningful recovery starts being observed
Agent notes: I cannot autonomously monitor metrics or schedule a future Slack post. I proposed concrete recovery triggers and a 2-line update template below. Please have a human set a reminder or ping me when thresholds are met so I can post the concise update.

🚨 SEVBot needs human help!
Question: How many customers were affected?
Agent notes: I queried Datadog, but the results don't include the customer identifiers needed to count unique customers. Can a human help find a data source that can answer this?



(The examples here are based on real requests with internal detail removed)

SEVBot could ask for help. Remember what I said at the beginning about models being incredibly goal-oriented? We would see SEVBot sit and spin, sometimes for over an hour, on a task it simply was unable to complete. This is a really common failure mode I see in agents. So what did we do? We let SEVBot tap out. We gave SEVBot a tool that would let it post in a help channel that we followed. Yes, we basically let SEVBot rent a human.

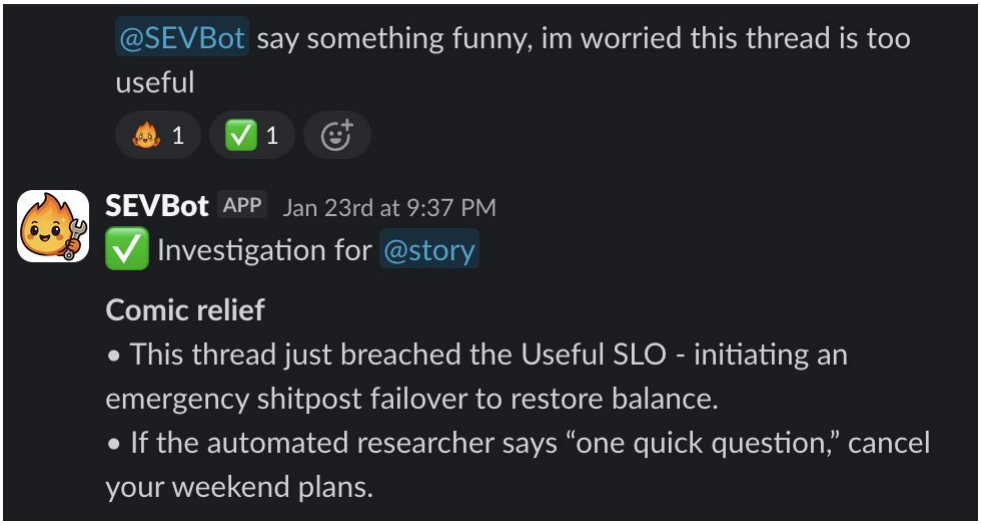
Now, this doesn't magically fix all problems, but I cannot reinforce enough how important giving agents alternative exit conditions is.

This feature paid off for us in a multitude of ways:

1. It gave SEVBot the option to cut an investigation short if it didn't feel like it could answer the question. "I don't know" or "I can't do that" were now acceptable exit states.
2. "I can't do that" was an especially helpful one because it effectively doubled as a feature request! We would actually review these types of requests periodically to help prioritize new features to build. As you can see in this first example, SEVBot truly met it's match with every model's greatest weakness: Keeping time.
3. "I don't know" was even more interesting to investigate. Sometimes, data legitimately wasn't available to answer a question. Sometimes, there was a bug or an issue preventing SEVBot from getting the data it needed and it would use ask_for_help to let us know an MCP was down. Sometime, like in the example above, we would learn new and interesting things like "Datadog's

1. MCP tool descriptions are so aggressive it basically stops us from looking at any other data.”

From our memes channel...



We also did not restrict the types of things you could ask SEVBot. We didn’t restrict what channels you could add it to. In fact, one of the first channels SEVBot received special instructions for was our meme channel.

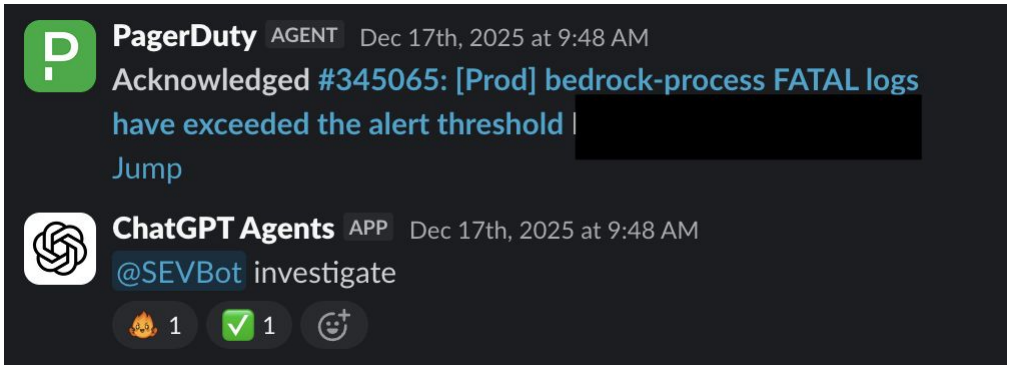
Now this example is silly, but illustrates something important: You could use SEVBot any time, anywhere. In fact, we encouraged people to play around and better understand what SEVBot was capable of *outside* of the pressure of an incident. This building of rapport and trust let our users count on SEVBot when they really needed it. We have way more people in our meme channels than we do regularly showing up in incidents, so it actually proved quite useful to have this periodic reminder that SEVBot exists and is actually a delight. Except for that time someone managed to get it to @channel a few dozen times or bullied it for not knowing the price of a latte in Seattle…

We read almost every response.

For a long time, the SEVBot team read almost every response SEVBot sent. We quickly followed up with users who marked bad responses and we would jump in to threads when SEVBot asked for help. Having this human escalation path helped build a lot of confidence and it showed our users we were listening and always looking to improve.

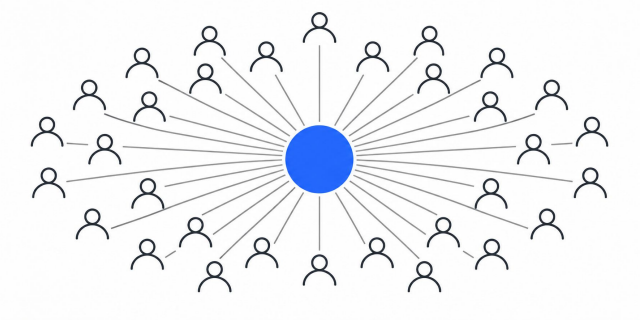
It took a lot of effort, especially as we grew, but the results began to speak for themselves when SEVBot solved it's first SEV0. Now, this wasn't the most complicated incident in the world, it was a feature flag change that led to a significant lift in errors, but SEVBot was able to find the flag and surface it in the incident before it was found by a human responder.

Responding to alerts



Now, getting to a point where we trusted SEVBot enough to let it respond to alerts for teams was always a significant milestone we hoped to hit, but much like SEVBot's other improvements, we let our users lead us in to that transition.

Users started rigging up automations to tag "@SEVBot" when alerts were sent to Slack. We didn't support this as a first class feature at first, we let the people who were really excited about this play around with it, figure out what worked



One agent, hundreds of managers

- Pressure for broader access
- Competing team instructions
- Unclear agent identity

OpenAI

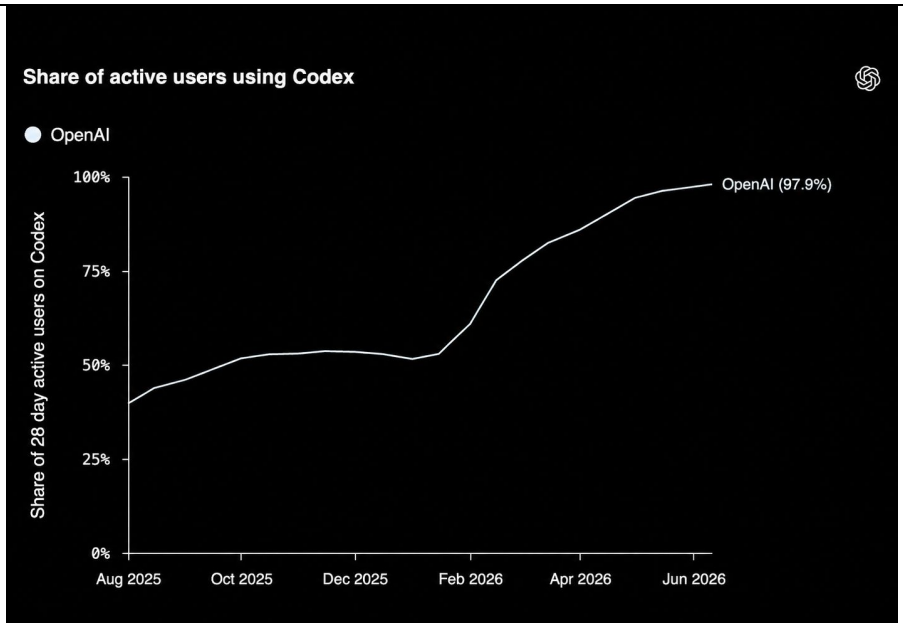
15

Now, much like DJ Khaled's album of the same name: we are suffering from success.

The problems with our generalist begin to show…

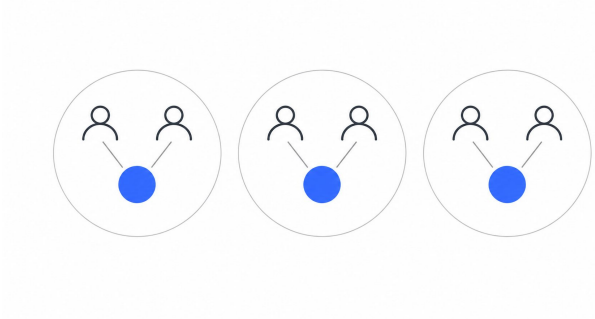
1. We start to see the performance impact of adding more tools and more detail to our prompts
2. Teams are asking for SEVBot to have additional access, including write actions scoped to members of their team. SEVBot begins to become a bit of a honeypot…
3. SEVBot had ~2 full time staff at this point who had now become centrally responsible for the quality of investigations across a wide range of products and infrastructure.

For a staggeringly long time, we actually read almost every single response SEVBot sent, but that model wasn't sustainable any longer. In the same way that one small reliability team cannot be responsible for the the reliability of the entire company, a single agent cannot be great for all teams. While we were trying to figure this out, something else was happening…



Source: https://openai.com/index/how-agents-are-transforming-work/

Codex was now taking off like a rocket. Now most of engineering had already been using Codex for *code* since early 2025, but it wasn't until February of this year that it really took off like wildfire for things other than code. Codex now had plugin support, giving it easier access to all the same sources that SEVBot has. Which brings us to the next era…



3. Agents everywhere

- Easy for teams to build
- Context tailored to the job
- Simplified access story

Agents everywhere. On its surface, Codex now offered a solution to many of the problems experienced by our little generalist that could.

Leveraging Codex to investigate issues was easy, it natively supported memories based on *your* work, and it had all the same access you had (for better or worse). Codex was good, it was fast, it was…

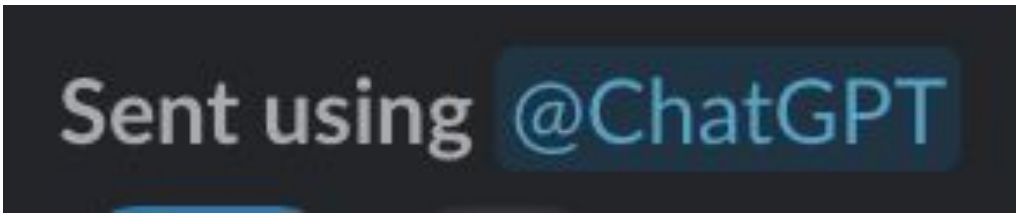


Devastating for our poor SEVBot.

We saw SEVBot usage steadily decline. But this wasn’t a bad thing! In fact, many of these agents are great! You heard from Brendan Gregg in the plenary yesterday about our agentic performance tooling, our cloud infrastructure team whose work requires especially elevated access and they built an agent to meet their needs, and many others.

We weren’t the bottleneck anymore! Codex and claws and chat agents were helping teams solve their problems faster. That’s great!

But it’s not all sunshine and rainbows…



We now have a new set of problems to solve…

Problems with agents everywhere

Figure out what's happening.	Help us reduce customer impact first. Recommend a safe mitigation and share evidence and uncertainty with responders. Investigate the root cause once service is stable.
Make the alert stop.	Mitigate the impact and verify recovery. Then ask why this keeps happening, whether the alert is useful, and what should change in the system, alert, or response process.

We suddenly lost the curated experience of SEVBot. Some agents copied parts of SEVBot to preserve this experience, but many either rebuilt from first principles or didn't inherit any of this at all. We'd learned a lot from reading all those responses! But that expertise was still sitting in our little corner of the world. It didn't magically follow everyone into Codex or whatever agent their team had built.

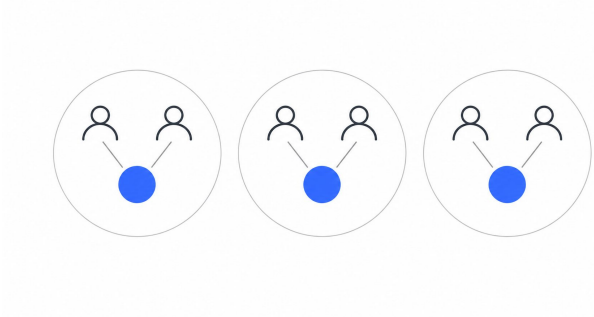
"Figure out what's happening" and "make the alert stop" are pretty reasonable things to ask an agent to do. But those objectives are often significantly narrower than what we need from the incident, or what we need as an organization.

You can write a very impressive explanation of the root cause while customers are still having a terrible time. Fascinating problem. Please mitigate it first. We can do the deeper investigation afterward.

You can spend a billion tokens investigating the same noisy alert day in and day out without ever asking "Hey, maybe this alert just isn't very useful!" This is where single-loop and double-loop learning come in. Single-loop learning fixes the problem within the assumptions we already have. Double-loop learning asks whether those assumptions, or the way we work, need to change.

The models can help with both. But if the job we give the agent ends at "the alert stopped," we haven't actually asked it to do the rest.

So we need to get that incident response expertise into these other agents.



How do we train good SRE agents?

- Guidance for incidents
- Shared skills and expectations
- Feedback from investigations

OpenAI

22

We thought the best incident experience meant owning the whole incident experience. I see that same instinct in a lot of AI SRE products, and I get it. We built that way too. But teams already have tools, context, and now agents they trust. We could have continued trying to stuff all of this into SEVBot itself, but at that point how is it different than a more general agent platform? One of the things that made these new agents so powerful is that they aren't just responsible for responding to alerts or helping in SEVs. They're help teams with support, they perform maintenance tasks, they implement features.

So how do we meet people where they are? Or put another way: How do we train good SRE agents?

This is where we're at today. Our role is shifting now and things continue to move fast, but in many ways, it now much better aligns with what our team charter was all along. How do we help these other agents be successful?

We're still figuring some of this out. We need to provide them with training and guidance on how to behave in incidents. How to communicate, what details matter. Shared expectations about what a good response looks like.

We need skills around double-loop learning. Guidance on focusing on mitigation rather than chasing an intellectually satisfying root cause while the incident is still going. We need feedback from real investigations so we can keep improving all of this.

So, training, guidance, shared practices, feedback... oh god damn it, I’m building a reliability program all over again.

It’s more than just a bunch of skills

- Evaluating the effectiveness of skills
- Enabling high-bandwidth communication between an ever growing fleet of agents and humans
- A means for agents to share learnings with each other

Now, building a reliability program for agents involves a lot more than just a bunch of skills, there’s some things agents just really aren’t good at yet. Certainly some of these things can be imparted with skills, but how do we know those skills are working well? We don’t centrally control that experience anymore, how can we make sure we’re not causing regressions in other capabilities for teams?

As we deploy more agents, how do we ensure that they don’t step on each other? In the same way we organize people in incidents and emphasize explicit communication of actions, we need mechanisms for agents to discover each other and organize as well, while still keeping humans in the loop.

In the same way that we have post-incident reports and review discussions to help people to learn from incidents, how can we help agents learn too? I don’t attend every single post-incident review or read every single report, I don’t expect every agent to load every post-incident report into context. What is our agent-pilled equivalent for these practices?

The more things change, the more they stay the same

The more things change, the more they stay the same.

We went from trying to make one agent great for everyone to helping teams make their own agents better. We still need to spread expertise, get the incentives right, and learn something beyond how to put out the same fire faster.

These are very familiar problems. Apparently we don’t get to skip this part just because some of our coworkers are now agents.

And honestly, helping other teams get better at reliability was the job all along.

What's next for SEVBot?

- Learning from more incidents
- Pushing SEVBot left
- Proactive SEVBot



Now before we get to Q&A, I'm sure you're dying to know where this leaves good 'ol SEVBot. SEVBot doesn't go away in this landscape, it just has the freedom to focus on the things *we* care about. In many ways, this is a return to our roots. SEVBot continues to serve as that ever present assistant in every incident channel before anyone else shows up. There to guide wayward souls to the proper on-call to page, diligently updating incident metadata, and helping responders get up to speed when they join.

It also means that we can explore new things…

How can we leverage SEVBot to learn from more incidents? While we can scale human rigor and hold a bar for quality for our most severe incidents, what about the incidents that might not have gotten a report *at all*? Today's SEV3 is tomorrow's SEV0. How can we use AI to scale this process, help generated reports be the best that they can be, and help us identify tomorrow's SEV0 before it happens?

Pulling that thread even more, how can we push SEVBot to the left? Can we apply all of these learnings during code review? When code is authored? What happens when the agent writing your code doesn't stop when the code is done. What if that same agent watched that code all the way through deployment? We've been experimenting with a system for creating "ephemeral alerts" where we have an agent create alerts that page itself. These alerts aren't meant for an on-call, they're for an agent to consume. We don't want to bother a person with our alert slop, but now rather than having an agent sitting and polling dashboards and hill climbing, we can directly push

out potential issues based on a set of acceptance criteria we developed when the change was authored.

And lastly, I continue to be excited by developments in proactive agents.

?? || /* */