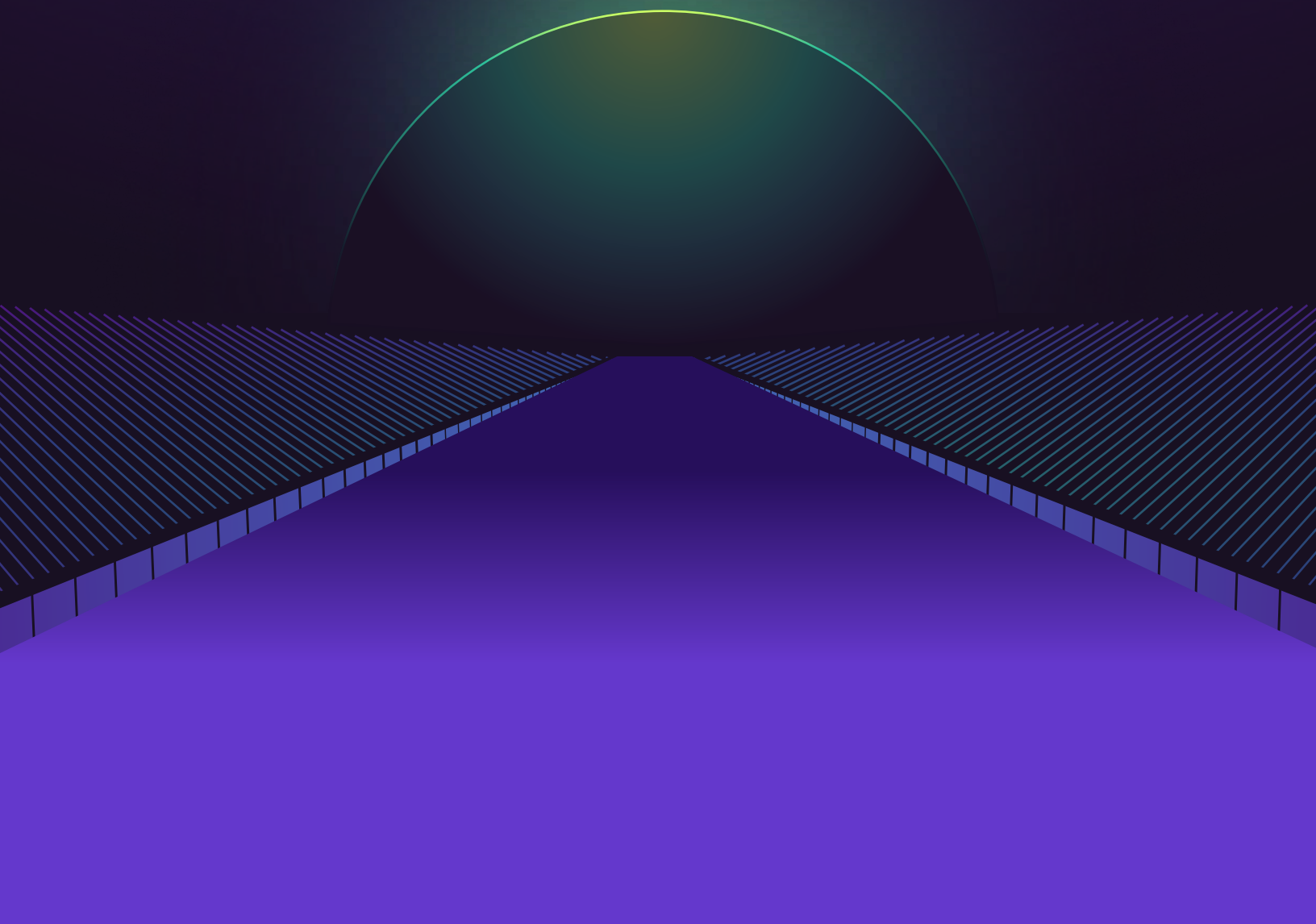


Gremlin

Speed and Resilience

HOW TO BUILD RELIABILITY GUARDRAILS FOR AI-DRIVEN
DEVELOPMENT



How to Build Reliability Guardrails for AI-Driven Development

The promise of AI is that we can get more done faster with less. And we've seen this promise realized across industries with agentic coding. Teams are deploying code at a pace unimaginable five years ago, so much that GitHub needs to 30X their capacity to handle the flood of code.

Unfortunately, going that fast isn't without risk. Thirty times the code often means thirty times the issues. And that assumes agentic-generated code has the same error rate as human-written code, which isn't always the case.

An analysis comparing AI code and human code found that AI generated an average of 1.7x as many issues per pull request. At the same time, 78% of companies report more incidents with AI, while 82% have faced AI production incidents in the last six months.

1.7x

More issues per PR
with AI

78%

Report more incidents
with AI

82%

Report AI prod incidents
in last 6 months

"AI-generated code is bringing architectural decisions that people aren't really looking too hard at, and they're catching them in hindsight, which will lead to reliability issues."

— **Engineering leader at a global CRM company**

If we're going to keep moving at the speed of AI, how can we move fast without breaking things? And, more importantly, how can we maintain customer trust and reliability? This is why AI-driven development demands reliability guardrails. These help prevent catastrophic outages and keep systems reliable without slowing you down.

Read on to learn more about reliability guardrails and how to build them with AI.

Reliability guardrails stop risks before they become catastrophic outages



The goal of reliability guardrails is to keep your systems on track when a failure mode is introduced in your code. Without guardrails, a failure mode can be released into production and turn into an outage when things go wrong. With guardrails, the failure mode is caught, addressed, and the systems get back on the road quickly without downtime.

And that's really the key concept: how can we keep agentic development reliable while still allowing it to move quickly? While QA testing can identify some of these risks, most only show up when the code is fully integrated into a production system or a production-like environment.

Resilience testing helps find those risks. By safely creating failure conditions, we see how the systems react, uncovering outage-causing risks before they become incidents.

That testing is the key difference between standards and guardrails. Reliability standards like docs, checklists, or even asking an LLM to check the code are more like the white lines painted on a road. They generally guide your path, but if something goes wrong, you'll blow right by them.

By using resilience testing as a gate, you create firm guardrails that actually stop reliability risks from reaching production, send them for correction, and keep deployment on track.

“Yes, we can write a lot of code. Yes, we can ship a lot of changes. But do we have confidence that those changes are going to withstand production?”

— **Kolton Andrus, Founder & CEO, Gremlin**

This is the core concept behind Chaos Engineering and fault injection testing, practices that Gremlin has been instrumental in pioneering. But without the right framework, toolset, and processes, these can be manual, complicated processes that can easily slow down deployment.

And if that wasn't ideal before, it's unacceptable in the age of AI.

The two logjams created by reliability guardrails



Proactive reliability really runs into hurdles at two distinct points, and the bottlenecks they create can make it untenable for fast-moving teams. As a result, engineering teams will skip proactive detection and settle for finding risks the hard way when they cause an outage.

Logjam #1: Finding reliability risks

The first step is uncovering potential risks. A decade ago, in the early days of Chaos Engineering, this required manual experiments, expert knowledge, and a high tolerance for risk. As a result, most teams limited these efforts to only critical systems or experimental efforts, rather than testing widely across their systems.

Fortunately, this particular logjam has been cleared with automation that can passively detect risks and safely run fault injection tests to uncover risks standardly and at scale across entire systems.

So this is the first step in building reliability guardrails: make sure you have an automated, safe tool capable of rolling it out across your system. This maximizes detection results and makes it easy to uncover risks without slowing down.

Logjam #2: Fixing reliability risks

Once we've uncovered a risk, we need to fix it. On the bright side, proactively uncovering the risk means we can fix it on our schedule rather than trying to patch it in the middle of an outage. But it still takes time to figure out the problem and address it.

The difference between successful and unsuccessful reliability efforts often comes down to whether or not they can unblock this logjam. Successful programs married the automated testing from above with accountability and defined workflows that add risk remediation work to sprints or existing processes.

Every company that wants to build effective reliability guardrails needs to do this, but in the age of AI, manually evaluating the problems and coming up with solutions simply isn't fast enough to keep up with agentic pipelines.

"It'd be a real shame if we can produce software twice as fast, but we can only release it at an even slower rate than we were before."

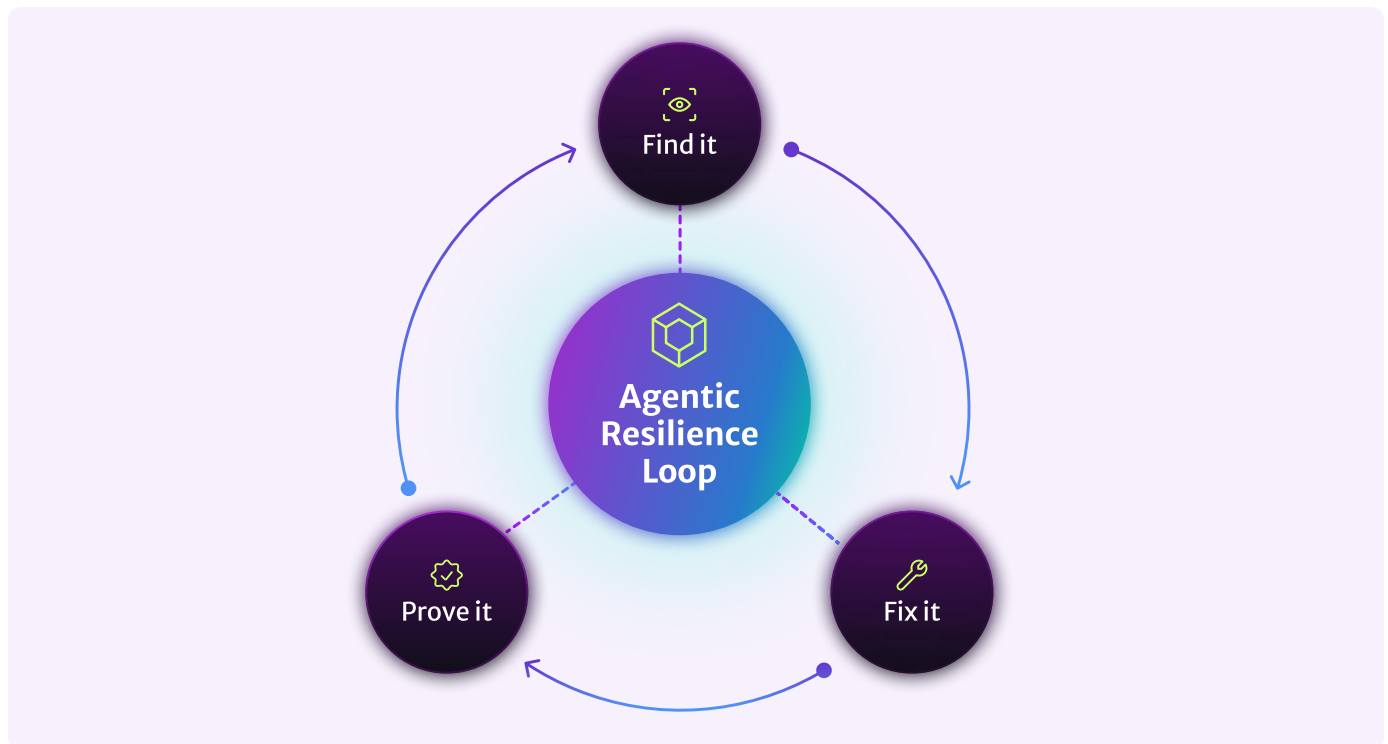
— Sam Rossoff, CTO, Gremlin

To truly remove the logjam, we need a way to automate both the finding and fixing steps.

Clearing both logjams faster with AI

To keep up with the pace of AI-enabled development, we need to create AI-enabled reliability practices that cover the entire process, from finding issues to fixing them to getting deployments back on track.

We need to create an agentic resilience loop that automates the entire process and integrates with our agentic pipelines.



Find it

Building on established automation and technology, we use AI to run passive detection, trigger tests when code reaches the right moment in the pipeline, determine which tests failed, and route any risks or failed tests for remediation.

Fix it

Previously, this step required expert engineers to dig into the systems and uncover issues, but what if we could use that expertise and data from past tests to give LLMs the context to propose fixes?

LLMs on their own won't get far. But paired with a harness that includes the right tools, skills, and context—such as real cause-and-effect data on how systems fail and recover—they form a stronger foundation. We can generate specific remediation recommendations and even code diffs designed to address the uncovered risk. And we can package up this suggestion for review, so instead of digging into the systems and tracking down a root cause, all an engineer has to do is send the code to their coding agents for integration.

Prove it

But how would we know whether or not the fix is effective? Has it really addressed the root cause of the problem? Fortunately, we can prove it by repeating the same test that uncovered the risk in the first place.

For example, a scaling test might create a spike in CPU usage to simulate a surge of traffic. If the system fails to scale correctly, then the test fails, and a recommended fix is created. Once the fix is integrated, we can run the exact same test to make sure it passes this time. If not, then we suggest another fix.

This final step is the essential part of reliability guardrails. It's where we go from uncovering issues to getting the show back on the road quickly and with provable confidence that we've removed the risk.

“It's not enough that I implemented a fallback. I have to test that the fallback works the way I expect it to.”

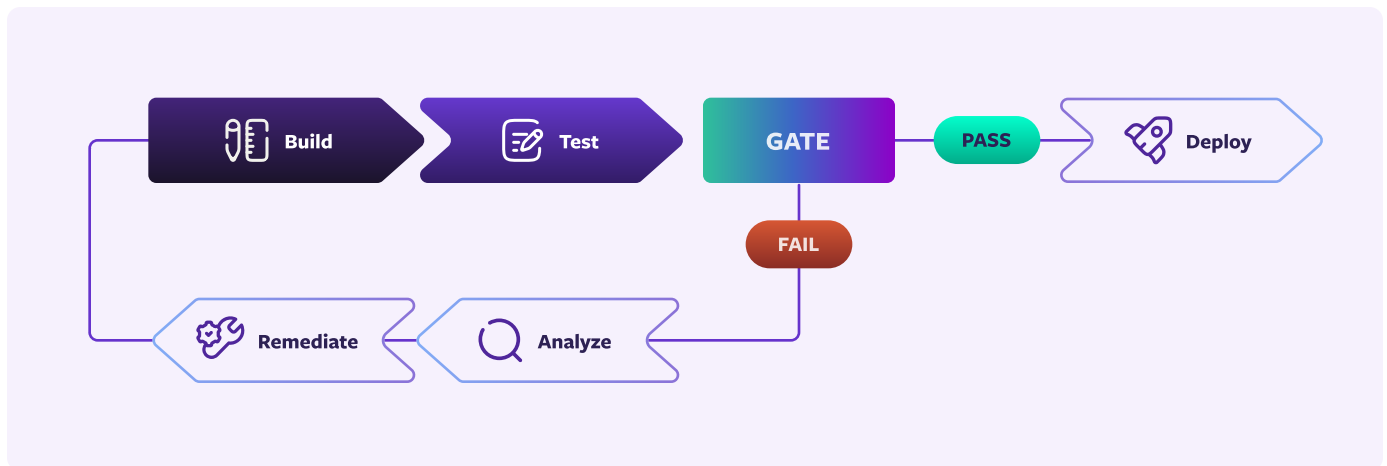
— Sam Rossoff, CTO, Gremlin

To truly remove the logjam, we need a way to automate both the finding and fixing steps.

Fitting guardrails into your pipeline

The easiest place to put reliability guardrails is as a gate, either in a production-like staging environment or in your actual production environment. That last part is crucial. Systems behave differently under a full production load, revealing risks that seemed perfectly fine under QA testing.

So make sure your testing phase happens as a gate either right before full deployment.



If the test is passed, the code can be deployed with proven resilience. If not, it can be sent quickly through the Agentic Resilience Loop above.

By using reliability guardrails as a gate like this and coupling them with AI-enabled remediation and testing, we can prevent risks from reaching production without encountering any logjams that would slow us down.

Gremlin makes it easy to build reliability guardrails



Gremlin Platform + Foresight AI

Build resilience into everything you ship and run

Gremlin replaces backward-looking incident metrics with forward-looking reliability scores based on how your systems actually respond to failure

Foresight AI scans and tests your systems for potential failures, fixes them, and verifies resilience. Let AI accelerate your development, not your incident count.



Professional Services

Deploy experts to your reliability program

Our reliability engineers work in your environment to find risks, prioritize fixes, and make sure they're addressed. With Gremlin Professional Services, your teams get the reliability expertise and visibility to create measurable results quickly.

Find out more and schedule a demo at [Gremlin.com](https://www.gremlin.com)