Closing the Loop

Why Self-Healing Operations Need Better Detection

Why every AI SRE needs a better trigger

AI can act. Can your systems tell it what matters?

Grepr

www.grepr.ai

Table of Contents

01
AI can act. Can your systems tell it what matters?
Why every AI SRE needs a better trigger

02
The Trigger Problem
Why AI SRE agents wait for something else to sound the alarm first

03
MTTD and MTTR
What's the actual difference between finding a problem fast and fixing it fast?

04
Observe
How do you observe the complete telemetry stream without indexing it all?

05
Detect
How do you identify problems that show up before any rule exists for them?

06
Detect and Learn
Why every AI SRE needs a better trigger

07
Remediate
Turn a validated finding into governed action

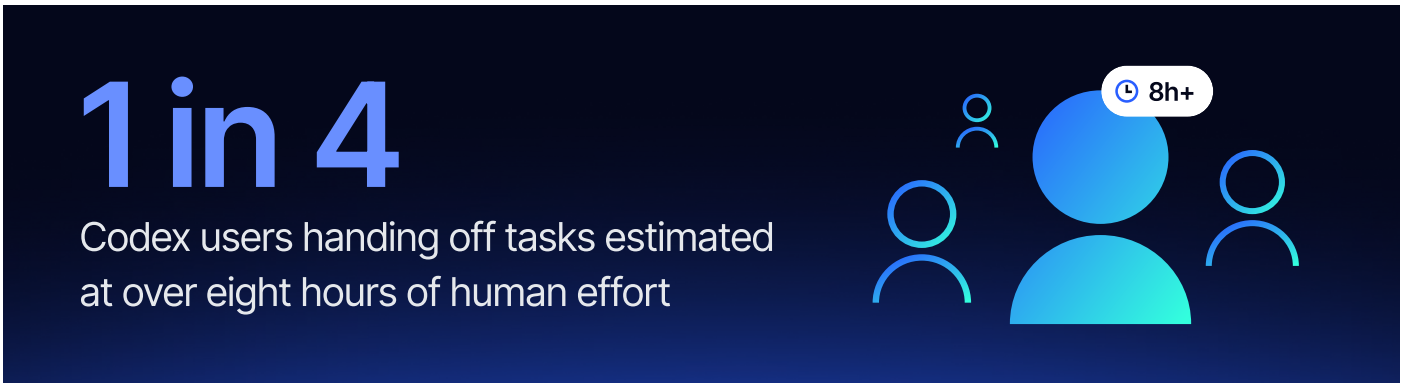
08
Closed-loop readiness scorecard
How ready is your operations stack to Observe, Detect, and Remediate?

09
Closed-loop maturity
What should you fix first, and what can wait 90 days?

10
Where Grepr fits
Add the missing trigger without replacing the tools your team already trusts

Why every AI SRE needs a better trigger

By June 2026, OpenAI's own usage data on Codex showed how quickly autonomous AI work was expanding. About 1 in 4 Codex users were already giving the AI agent tasks that would take a person more than 8 hours to complete. As software delivery accelerates, production systems are changing faster than traditional monitoring coverage can keep up.



Most alerts still depend on conditions someone anticipated and configured in advance. A threshold gets crossed. A known failure mode appears. A rule fires because someone had already imagined what could go wrong. As systems change faster, that model leaves more room for failures that sit outside existing alert coverage.

The operational cost of that gap is already showing up. In March 2026, an internal Amazon briefing document reviewed by CNBC identified "GenAI-assisted changes" as one factor in a broader "trend of incidents." Amazon later removed that language from an updated document and said only one of the incidents was AI-related, with none caused by AI-written code.

> **The point is not that AI caused the incidents. It is that the pace of software change is increasing while detection still depends heavily on what teams already know to look for.**

AI SRE systems are getting better at investigating incidents, proposing fixes, and in some cases executing them. But all of that starts after something has been detected. If the monitoring layer does not recognize that something has changed, the agent has nothing to investigate.

That is where the real gap begins. Most operations stacks still rely on rules written for a system that may already have changed underneath them.

Self-healing operations begin with a better trigger: a system that can observe what is changing, detect what matters, support the right response, and verify whether that response worked.

Why AI SRE agents wait for something else to sound the alarm first

AI SRE agents have gotten good at the parts of incident response that come after something goes wrong. Feed one an alert, and it can pull the relevant logs, compare the current pattern against last week's, propose a likely cause, and in some setups, even execute an approved fix.

Work that once depended entirely on an on-call engineer can now be investigated by an AI agent.

The process still needs a starting signal. A threshold is crossed, an alert fires, or a dashboard shows that something is wrong. Each of these depends on someone deciding in advance what behavior should trigger attention. If the problem falls outside that coverage, it may stay unnoticed until the impact becomes visible or a customer reports it.

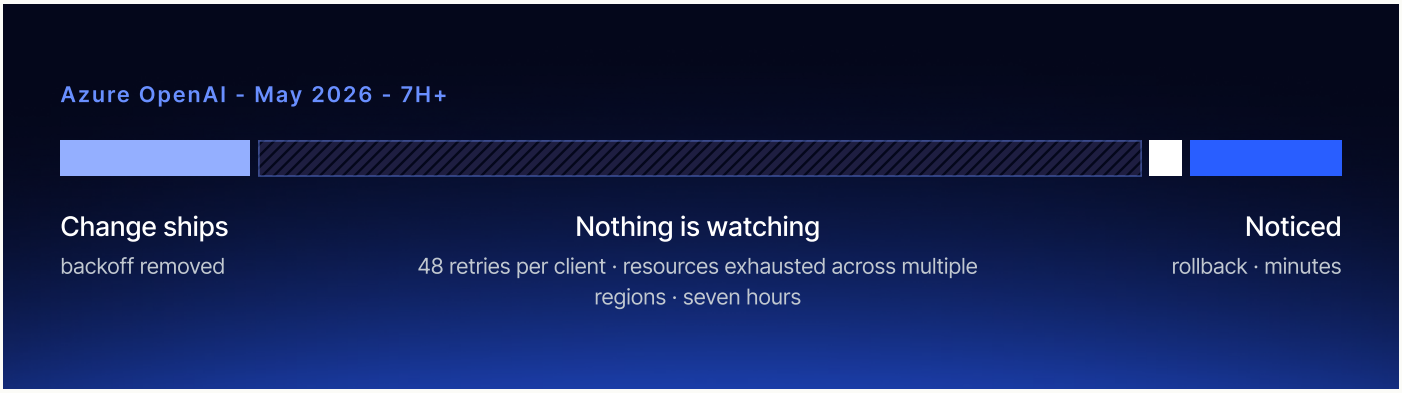
Take the May 2026 Azure OpenAI outage. An upstream API-layer change caused capacity-related failures to surface as transient, retriable errors. Multiple request-handling layers then retried immediately and repeatedly, without sufficient backoff or jitter. In some cases, a single failed request generated up to 48 additional retry attempts.

The resulting retry amplification overwhelmed shared inference-routing resources across multiple regions, with customer impact lasting more than 7 hours.

Once someone noticed, the response was straightforward: roll back the change and restore the backoff logic. The harder part was detection. Retries without backoff sat outside the existing alert coverage, so the impact spread before the behavior was recognized.

An AI SRE agent sitting on top of that same monitoring setup would have inherited the same blind spot.

The agent can only investigate once something tells it there is a problem worth investigating.



What's the actual difference between finding a problem fast and fixing it fast?

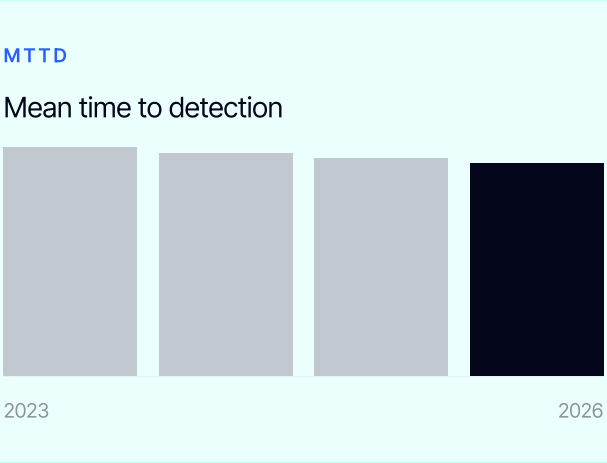
→ Two numbers are useful here, and they measure different parts of the incident lifecycle.

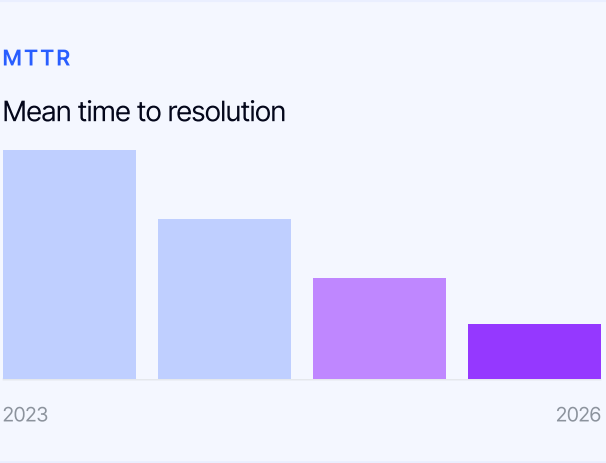
MTTD

Mean time to detection, or MTTD, measures how long it takes to discover that something is wrong.

MTTR

Mean time to resolution, or MTTR, measures how long it takes to resolve the problem after it has been detected.





MTTD hasn't seen the same progress as MTTR. The agent still depends on the monitoring layer to tell it that something needs attention. If a failure sits outside existing alert coverage, the agent inherits the same blind spot.

AI SRE tools have made significant progress on MTTR.

Once an issue is flagged, an AI agent can investigate, gather context and support a response faster than a person working alone.

As AI-assisted development increases the pace of change in production, the number of behaviors that have never been seen or anticipated will grow with it. Faster resolution only helps after the problem has been found.

A better trigger needs to do five things:

01 Observe the telemetry needed to identify changes as they happen, without leaving structural blind spots.

02 Recognize novel behavior that falls outside existing alert coverage.

03 Determine whether the change is problematic or routine.

04 Supply enough context and evidence for a human or agent to act.

05 Retain the underlying telemetry so the evidence can be recovered later for validation, backfill or investigation.

How do you observe the complete telemetry stream without indexing it all?

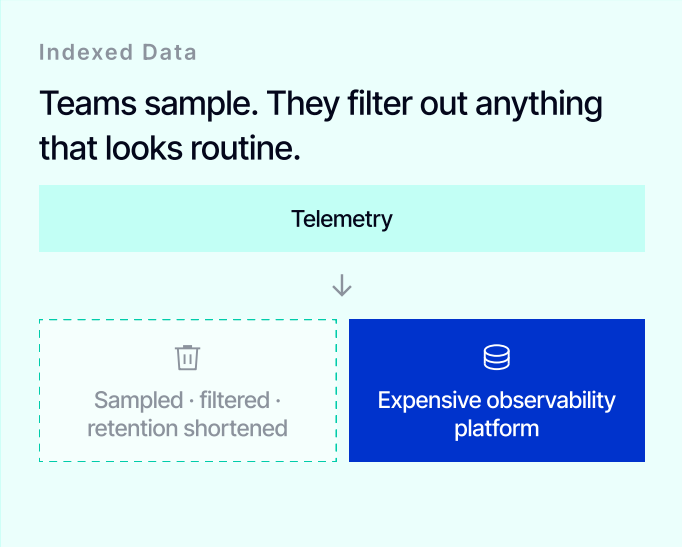
Most observability tools can only analyze the data you send them, and sending everything is expensive enough that most teams stop well short of that.

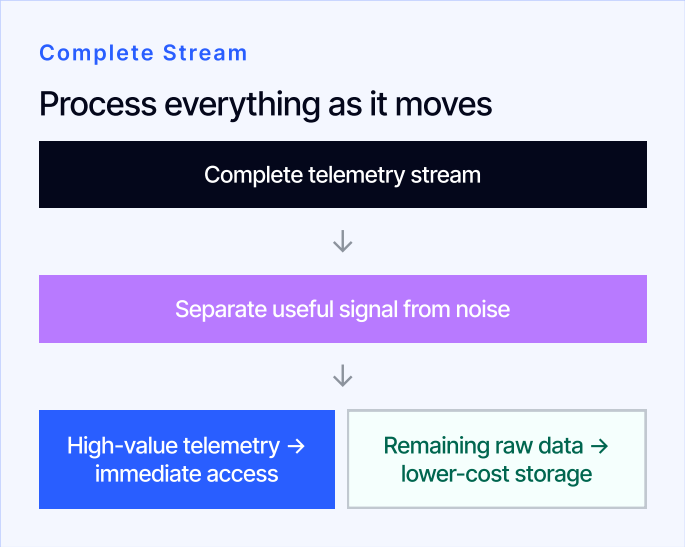
Teams sample. They filter out anything that looks routine. They shorten how long they keep raw logs. Some sources stay out of the pipeline entirely because the cost of ingesting them always outweighed the value.

→ The problem is that detection quietly becomes a budget decision.

The real question shifts from whether data is worth watching to whether the budget allows watching it. Whatever gets cut is gone because the bill was hard to justify, regardless of how much it mattered.

COMPLETE STREAM VERSUS INDEXED DATA

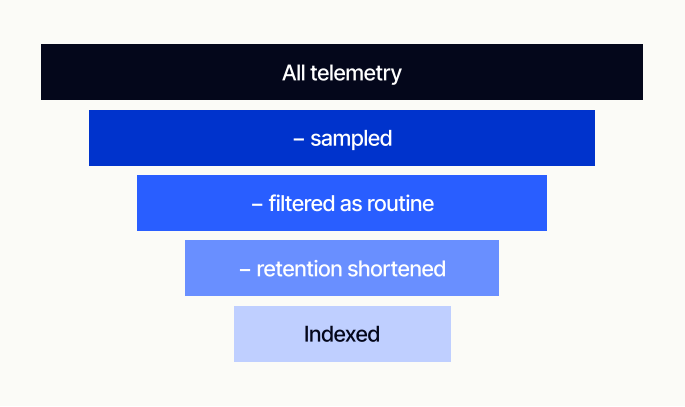




An ideal observability architecture can process everything as it moves, separate useful signal from noise, forward high-value telemetry to the tools that provide an SRE's engagement layer (eg, dashboards and alerts), and preserve the remaining raw data in lower-cost storage. That raw data can then be used for deeper investigations, backfill, validation, forensic evidence, or compliance requirements.

Such an architecture can align the value of the data with the cost of collecting it, storing it, and making it accessible. It avoids the cost of duplicate spans or repeated status messages or other noise in the more expensive, primary observability platform but makes everything available in a cost-effective tier.

The goal is not to indiscriminately reduce volume. It is to create information-dense telemetry without losing the unusual evidence that may matter later.



FILTERED OUT HERE

Cloudflare, February 2026

One empty parameter in a routine cleanup query. Twenty-five percent of BYOIP prefixes deleted. A five-hour outage. The evidence was in the telemetry stream, but nothing was watching that behavior closely enough to act on it.

The real question should not be whether the budget allows a source to be watched. It should be how the complete stream can be processed economically.

Collecting the complete telemetry stream does not require indexing every raw event in an expensive observability platform. Those are different things.

Detect

How do you identify problems that show up before any rule exists for them?

Known log pattern

Repeats as expected

14:02:11.204	payment.completed
14:02:11.389	payment.completed
14:02:11.552	payment.completed
14:02:11.740	payment.completed

Expected pattern

Changed log pattern

Detects a new pattern

14:02:11.204	payment.completed
14:02:11.381	retry.started
14:02:11.590	retry.started
14:02:11.802	retry.started

New pattern ×3 · No predefined rule required

Static alerts work exactly as well as the person who wrote them could predict. That holds up for known failure modes, the ones you have already seen or can reasonably imagine.

Everything outside that coverage grows as services change, schemas shift, and new dependencies get added faster than anyone can revisit the rulebook.

A better detection layer starts by noticing when the shape of the telemetry changes:
A new pattern appears, a familiar pattern changes frequency, or separate behaviors begin occurring together.

At production scale, sending every event to an LLM for investigation doesn't "scale." It's too slow, expensive, and noisy. You need an efficient way to inspect the complete stream and identify the perturbations worth investigating.

A semantic reasoning layer can then examine those changes, connect relevant context, and decide whether the behavior is meaningful.

This is a precise, testable claim: the system can surface behavior that predefined rules would have missed because it watches for change instead of checking only a fixed list of known conditions. It is not a promise to catch every unknown failure.

Context makes the difference between a useful flag and a false alarm.

01

Did the behavior follow a deployment?

02

Which service or dependency changed?

03

Did other telemetry change in the same window?

04

Has similar behavior appeared before, even in a different service?

05

Which customers, regions, or infrastructure components are affected?

06

Does system behavior align with the code change that was just shipped?

Answering those questions is what separates "something's different" from "something's wrong."

Before the finding becomes another alert, the system should gather corroborating evidence. Some investigations require additional observation over time or greater granularity from a specific source.

A system that escalates every anomaly immediately only replaces alert fatigue with anomaly fatigue.

Why catching something once still leaves you exposed the next time

Catching an unusual pattern once is a good start, though it stops short of continuous protection.

If the system forgets what it learned after the investigation ends, the same failure mode can return and slip through detection again.

→ A validated finding needs to become continuous detection logic.

The sequence is specific:

01 Notice an emerging signal.

02 Theorize about what the change could mean.

03 Validate the hypothesis against the available evidence.

04 Codify the validated finding as continuous detection logic.

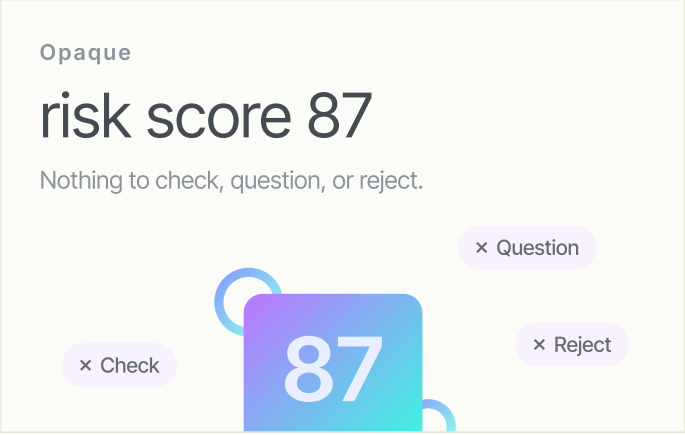
05 Keep watching and revise that logic as the system changes.

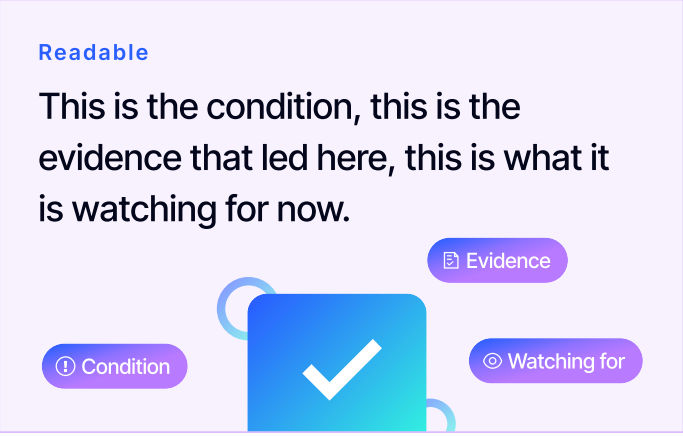
Once deployed, the detection can monitor future telemetry without requiring a human to recreate the investigation or manually write the logic from scratch.

How the detection is presented matters as much as what it monitors.

An unexplained risk score leaves an operator guessing why the system flagged something and whether the finding should be trusted.

> Inspectable detection logic gives the operator something concrete to read, review, approve, question, or remove.





The system should show:

- The condition being monitored
- Why the condition matters
- The evidence that led to the detection
- What future behavior will cause it to trigger

Durable detection should not mean permanently frozen detection.

Services are added, schemas change, baselines move, and old detections lose their usefulness.

Machine-generated detections need the same ongoing review as any other part of the monitoring system. The difference is that coverage can now grow with the system instead of waiting for a human to rebuild it after the next incident.

Remediate

Turn a validated finding into governed action

What happens once an issue is detected

Once the system finds a meaningful problem, the appropriate response depends on the finding's severity, confidence, potential impact, available playbooks, and organizational policies.

A minor anomaly might just need a note for the next person reviewing dashboards.

Something with real potential impact might need a human paged immediately. A well-understood, previously-seen issue might be safe to hand straight to an existing AI SRE agent, or even fixed automatically if it's already been pre-approved for that kind of action.

The response can include:

01	02	03	04	05
Alerting a human responder	Recommending a remediation	Handing the investigation to an AI SRE or specialized agent	Requesting approval to execute an action	Running a pre-authorized remediation automatically

Whatever path the system chooses, the handoff needs to provide more than an alert label.

Remediation plan	6 required fields
✓ What changed	✓ Why it matters
✓ Which systems are affected	✓ What evidence supports the finding
✓ What action is recommended	✓ What the risk and rollback steps look like if that action needs reversing

That level of detail is the difference between telling someone to investigate and giving a human or agent enough evidence to act immediately.

Authority should be defined before an incident, not improvised during one.

Organizations need clear policies governing which tools an agent may access, which actions require approval, which remediations may run automatically, and when a human must take control.

EVERY PATH SHOULD PRODUCE AN AUDIT TRAIL

01	→	02	→	03	→	04
What happened		Why it happened		Who or what approved it		What action ran

The loop closes only after the system checks the outcome.

Did the action resolve the issue? Did it create an unexpected effect elsewhere? Should the detection logic, remediation plan, or approval policy change based on the result?

> Observing the outcome turns a one-time fix into a learning loop that can improve future detection and response.

Closed-loop readiness scorecard

How ready is your operations stack to Observe, Detect, and Remediate?

The ideas above are easier to support in the abstract than to apply to your own environment.

Use the following questions to identify the parts of the loop that are already working and the places where your team still depends on cost-constrained visibility, manually maintained rules, or improvised response.

ON OBSERVATION

	Question	Response
01	Do you find that some of the telemetry data you need for troubleshooting is missing?	○ Yes ○ No ○ Not sure
02	When investigating an issue, can your teams access and query all of the telemetry data they need?	○ Yes ○ No ○ Not sure
03	When telemetry volume needs to be reduced because of cost pressure, is there a structured policy, with oversight, for deciding what stays readily available and what moves to lower-cost storage?	○ Yes ○ No ○ Not sure

ON DETECTION

Question		Response
04	Does your alert coverage include SLOs across the full user experience, rather than relying only on individual endpoint metrics?	○ Yes ○ No ○ Not sure
05	How often do leading-indicator alerts catch issues before they escalate into major production incidents?	○ Yes ○ No ○ Not sure
06	Are new leading-indicator alerts mainly created after incidents and post-mortems, or can your monitoring surface new behavior before a rule exists?	○ Yes ○ No ○ Not sure

An honest answer matters more than an optimistic score.

Review your last several incidents and ask which ones were detected by something watching for change and which ones were discovered through luck, impact, or a customer complaint.

> Continue with remediation readiness before selecting your maturity stage.

Closed-loop maturity

What should you fix first, and what can wait 90 days?

When responding to incidents, the questions get more concrete. How much time does your team spend understanding what happened before it can act?

Does every finding still land on a person, or are some investigations handed off to an AI SRE agent or automation?

Is there a defined policy for what an automated system can do on its own, or is that decided case by case during an incident?

If someone asked you to show what an automated system did last month and why, could you?

When your team catches something genuinely new, does that finding become a rule that keeps watching for the same behavior, or does it end when the incident review closes?

Which of the four stages are you in?

→ Most teams land on one of four stages.



Reactive static monitoring

Alerts and dashboards get built or updated after something has already broken or a customer noticed.



Proactive static monitoring

Monitoring is reviewed before each release, but it still depends on someone anticipating the likely failure modes in advance.



AI-assisted response

An AI agent helps investigate and act once something triggers an alert. This can speed up resolution, while still inheriting whatever blind spots already exist in the monitoring that triggered the alert.



Adaptive closed-loop operations

The system continuously watches the full telemetry data stream, catches behavior that appears before a rule exists for it, and turns validated findings into monitoring and action. Humans set the boundaries, approvals, and policies rather than running every step manually.

If you are behind the fourth stage, and most teams are, 90 days is enough to make real progress.

The next 90 days

If you are behind the fourth stage, 90 days is enough to make meaningful progress.

01 Make sure your AI agents can access and query the telemetry data they need through APIs or MCP servers.

02 Identify telemetry blind spots caused by filtering, sampling, or dropping data. Collect the full telemetry stream where needed, while routing lower-value data to lower-cost storage and keeping higher-value data readily accessible.

03 Test proactive anomaly detection with an AI agent for triage on a recent incident, and see whether it would have surfaced the issue earlier than your existing static rules.

04 Measure true positives, false positives, and false negatives, and track whether the system catches issues your existing rules would have missed.

05 Retain the underlying telemetry so the evidence can be recovered later for validation, backfill or investigation.

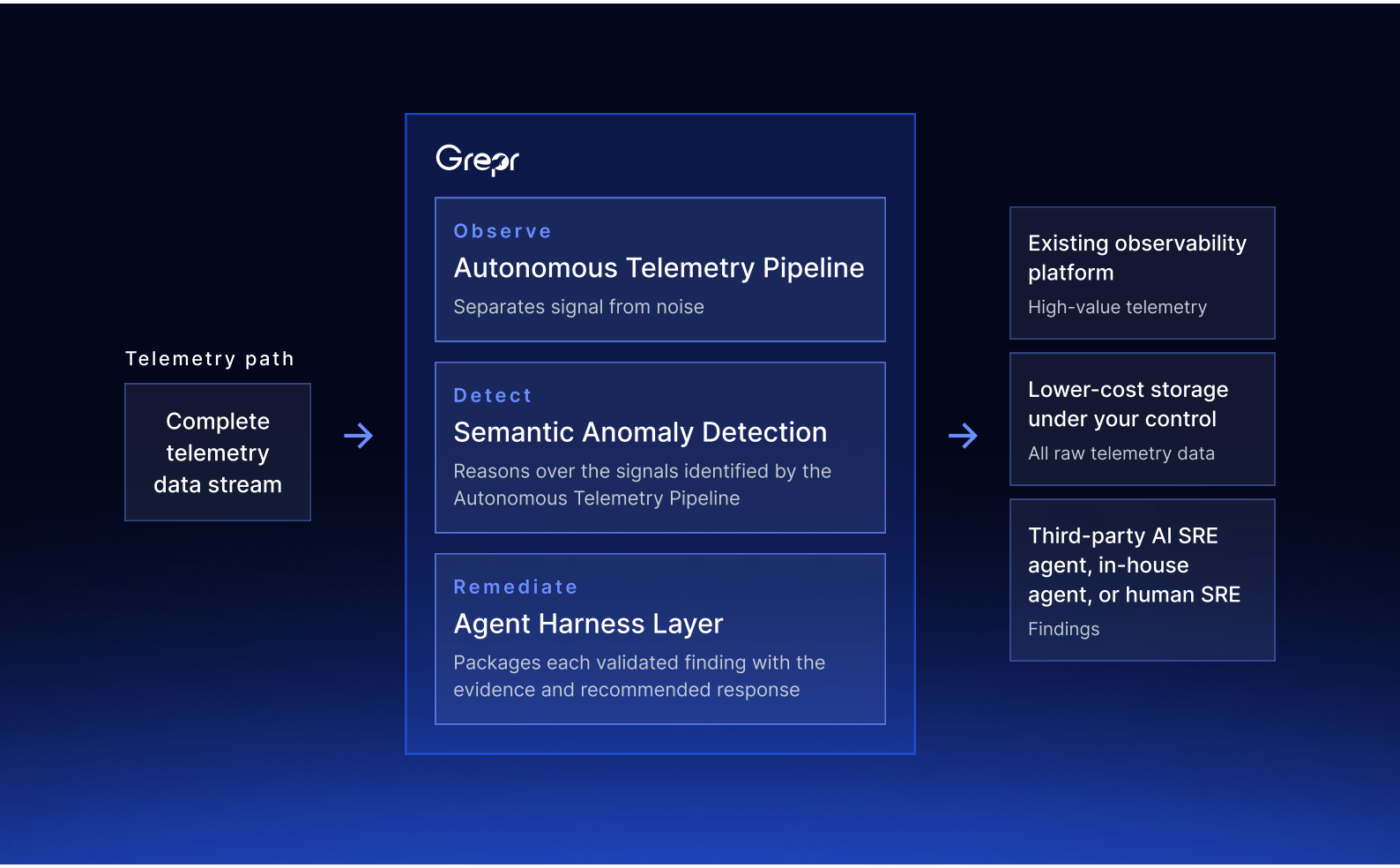
90 days is enough to make measurable progress even if your team is not ready for fully closed-loop operations. Improving response speed alone leaves the trigger problem in place. Improving detection without a controlled path to action leaves the loop unfinished.

Where Grepr fits

Add the missing trigger without replacing the tools your team already trusts

Grepr is designed to close the gap between telemetry collection, detection, and response.

It supports the full Observe, Detect, and Remediate loop while working alongside the observability platforms, AI SRE agents, automation tools, and response processes already in place.



Together, these layers give AI SRE agents the trigger they are currently missing. Grepr improves what feeds the existing operations stack rather than forcing teams to replace tools they already trust.

Observe

Autonomous Telemetry Pipeline

Grepr's Autonomous Telemetry Pipeline sits directly in your telemetry path and processes the complete telemetry data stream automatically, without requiring you to write manual rules. Its ML-powered detection engine separates signal from noise, forwards high-value telemetry to your existing observability platform, and preserves all raw telemetry data in lower-cost storage under your control.

This allows you to observe all of your telemetry data without requiring every raw event to be indexed at premium cost.

Detect

Semantic Anomaly Detection

Grepr uses AI to reason over the signals identified by the Autonomous Telemetry Pipeline instead of waiting for a predefined alert to fire.

Grepr AI connects relevant context, tests hypotheses against the available evidence, runs custom AI-powered workflows to triage, troubleshoot, and remediate where possible, and creates continuous static detections for issues that could emerge in the future based on its findings.

Those detections can run deterministically against future telemetry rather than requiring repeated inference on every raw event.

Grepr detects anomalous behavior that rules-based alerts would have missed. The difference is that something is now watching for behavior outside the rulebook.

Remediate

Agent Harness Layer

Grepr packages each validated finding with the evidence and recommended response needed for action.

Based on your prompt and configuration, Grepr can use its own findings to orchestrate remediation workflows across various tools, or send those findings to a third-party AI SRE agent, an in-house agent, or a human SRE.

The actions and orchestrations Grepr AI can execute are tied to the permissions, confidence thresholds, approval requirements, and audit policies you define. Familiar, low-risk remediations may run automatically. Higher-risk or unfamiliar actions can require human approval.

After an action runs, the system observes the result and tunes itself to improve future detection and response.

How ready is your operations stack to Observe, Detect, and Remediate?



To identify the gaps and see how Grepr can help close the loop.

Schedule a demo →

Grepr