Five structural gaps in error monitoring

Error monitoring means keeping track of bugs that happen after your software is live. It is production infrastructure that engineering teams use to stay up to date with their applications' production health.

Over the past six months, we spoke with more than 100 engineering teams across the industry. Five pain points came up consistently as the most urgent gaps in error monitoring:

01 **Poor clustering leads to too many duplicates.** Errors that share a root cause get split into separate issues, and unrelated errors get merged into one.

02 **Poor error prioritization.** The noisiest errors surface first, not the ones affecting the most users or the most critical paths.

03 **Configuration decays and requires maintenance.** Alert thresholds, ownership rules, and grouping rules go stale as the codebase and team change underneath them.

04 **Alerts are too noisy.** Too many alerts fire before anyone has investigated, sending the wrong signal to the wrong person.

05 **AI-generated fixes can't be trusted.** AI fix suggestions are generated without runtime context, leading to patches that mask symptoms rather than address causes.

This paper examines each gap: the external evidence, the structural reason it exists, and how Foam is built to close it.

About Foam

Foam is composed of agents that monitor your production services. It ingests traces, logs, and metrics over standard OTLP, clusters incoming errors into logical issues, prioritizes them by actual user impact rather than event volume, investigates each one against your codebase at the failing commit, and routes findings to the right engineers in Slack with a proposed fix as a pull request.

01 Poor clustering leads to too many duplicates

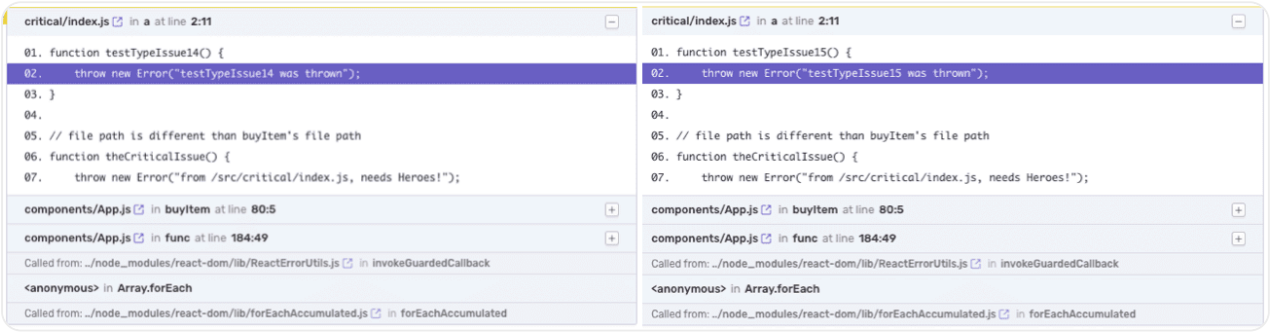
The industry's best shot at error grouping

The mechanism description and the stack-trace example below are taken directly from Sentry's own grouping documentation. [7]

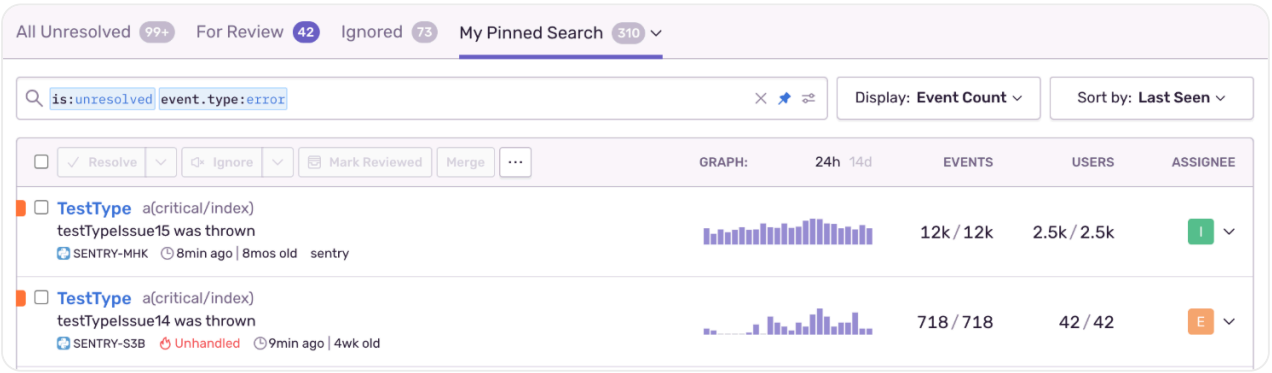
Sentry groups events by fingerprint, a hash derived from the stack trace, exception type, and message. Events with the same fingerprint become one issue. Events with different fingerprints become separate issues, even if they share the same root cause.

SENTRY FINGERPRINTING IN PRACTICE

Same error at two different times — Sentry splits them into separate issues because the function name changed between releases



Sentry creates two separate issues



Two issues with different ages (4wk vs 8mo), different event counts (718 + 12k split), and different assignees split because the function name varies between releases.

To fix this in Sentry, an engineer must manually write a fingerprint rule in project settings or merge existing issues by hand. [7] Sentry has since introduced AI-enhanced grouping using transformer-based embeddings, which their own data shows reduces new issues by 40% [4] [8]. This is meaningful progress, but it still operates within the

fingerprinting paradigm: grouping by textual similarity of the failure, not by causal analysis of what produced it.

Why fingerprinting fails to deliver

Fingerprinting groups by a property of the failure (the call sequence) rather than by a property of its cause. When the root cause is a connection-pool exhaustion, an upstream timeout, or any condition that surfaces through multiple call paths, the call paths multiply and the grouping fragments. When two distinct bugs share a common path through framework or library code, they collapse into one issue.

The two failure modes are fragmentation and false merging. In fragmentation, one incident generates multiple separate issues, each tied to a different stack frame, each triggering its own alert. In false merging, unrelated bugs with a shared library path land in the same issue, and a fix for one leaves the other silently open.

Both are downstream of the same structural constraint: fingerprinting is text pattern-matching applied to a snapshot of the call stack at the moment of failure. It has no model of what caused the failure, only of what was executing when it happened. Foam addresses this by further clustering on causal similarity rather than call-stack similarity, and by investigating each error before surfacing it.

> **Fingerprinting groups by a property of the failure, not by a property of its cause.**

How Foam improves clustering with a median 10.9X reduction

Foam clusters errors with Drain3 [2], a fixed-depth streaming parser that emits wildcarded token templates as cluster keys. Then two layers extend vanilla Drain3: AI-generated template proposals for cases where the message structure varies in ways token frequency cannot align, and Jaccard similarity scoring that collapses near-duplicate templates before they fork into separate errors. See below for two production examples where clustering reduces separate stack traces to a single Foam error.

FOAM CLUSTERING EXAMPLES

```
PG::QueryCanceled: ERROR: canceling
statement due to statement timeout ·
SELECT usage_units, period_start,
period_end FROM subscription_line_items
WHERE subscription_id = 'sub_A1B2C3' AND
billing_period = '2026-Q1'
```

```
PG::QueryCanceled: ERROR: canceling
statement due to statement timeout ·
SELECT usage_units, period_start,
period_end FROM subscription_line_items
WHERE subscription_id = 'sub_X9Y8Z7' AND
billing_period = '2025-Q4'
```

→

```
PG::QueryCanceled: statement timeout
· SELECT usage_units, period_start,
period_end FROM
subscription_line_items WHERE
subscription_id = <*> AND
billing_period = <*>
```

```
Redis::TimeoutError: Connection timed
out · app/services/session_store.rb:47
in get_session
```

```
Redis::TimeoutError: Connection timed
out · app/services/rate_limiter.rb:23 in
check_limit
```

```
Redis::TimeoutError: Connection timed
out · app/services/feature_flags.rb:61 in
evaluate
```

→

```
Redis::TimeoutError: Connection timed
out · <caller>
```

Three callers but one root cause: the shared Redis instance is unreachable. The session store, rate limiter, and feature flag service each talk to the same Redis host.

Clustering performance across production environments

The table below summarizes Foam's clustering pipeline running against 11 production environments spanning different industries and scales, collected over 3 to 9 months of production data per environment. This data was acquired during Foam's transition from Sentry-based grouping to its own clustering technology: each environment was simultaneously tracked by both systems, allowing a direct comparison of how many issues each produces from the same underlying errors.

INDUSTRY	SENTRY ISSUES	FOAM ISSUES	REDUCTION
Billing infrastructure	15,858	1,320	12x

AI infrastructure	259	93	2.8x
Education	2,220	204	10.9x
AI detection	8,101	667	12.1x
Field operations	2,345	140	16.8x
AI content	694	200	3.5x
Accounting	485	201	2.4x
AI evaluation	30,410	863	35.2x
Developer tools	1,216	327	3.7x
HR technology	2,179	413	5.3x
AI search	62,884	1,243	50.6x

The median reduction is **10.9x**, meaning an engineering team sees roughly one-tenth the number of issues to triage. At the high end, an AI search company saw a **50.6x reduction** (62,884 stack-trace-grouped issues collapsed to 1,243 root-cause-grouped Foam Issues). Even the smallest environments see a 2–3x reduction, eliminating duplicate triage and alerting.

A detailed methodology and accuracy breakdown will follow in a standalone white paper.

02 Poor error prioritization

Why incumbents built it this way

Event count and recency were the only signals available at the time incumbent tools were built. These systems do not ingest the full OpenTelemetry pipeline: the traces, logs, and metrics that carry customer identity, request context, and the specific product feature or API route affected on every request by default [1]. When traces are available at all, incumbent systems typically sample them, keeping only a fraction of requests and discarding the rest before they can be correlated with errors. Without complete telemetry, there is no way to know which customer hit the error, which trace it belonged to, or what business surface was affected.

Foam does not sample. It consumes all logs, all traces, and all metrics over standard OTLP, so every request is available for correlation. This is economically feasible because Foam uses a push-model architecture with cold storage in object stores and on-demand compute. The always-hot indexing cost that forces incumbents to sample does not exist, which is why complete data ingestion is financially tractable. Today, AI also makes it possible to understand semantic context from error messages and code, further widening the gap between what incumbents use and what is available.

> **The always-hot indexing cost that forces incumbents to sample does not exist in Foam, which is why complete data ingestion is financially tractable.**

Foam uses other signals to close the gap

Foam's prioritizer runs after root-cause analysis and combines three signals, each independent of raw event volume.

THREE SIGNALS INDEPENDENT OF EVENT VOLUME

01	**OTel metadata**	Because Foam ingests complete traces, every error carries the user, tenant, and business surface that produced it. Prioritization uses this metadata directly rather than inferring impact from event counts.

02	**Spectral anomaly shape**	Foam applies a short-time Fourier transform [21] to each error's event-rate time series. High-frequency spectral power signals a fresh spike; low-frequency power signals a chronic background rate. Two errors with identical 24-hour counts rank differently if their spectral signatures differ.
03	**RCA-derived severity**	The investigation agent classifies each error by failure mode (data loss outranks latency degradation, which outranks transient unavailability) and code-path criticality (payment and auth paths outrank logging utilities). A low-volume error with a severe classification surfaces above a noisier but lower-severity one.

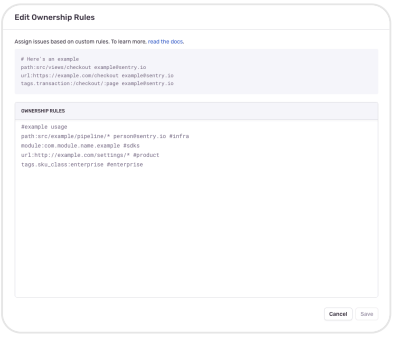
03 Configuration decays and requires maintenance

Why this matters

Error monitoring deployments are configuration-heavy. Each piece is written by a human against a specific snapshot of the codebase, and silently drifts as the codebase moves. Research on configuration drift consistently finds that static, rule-based configurations become unreliable in direct proportion to the pace of underlying system change [27].

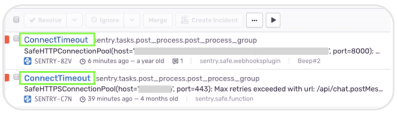
SENTRY: OWNERSHIP RULES [22] [23] [24]

Three separate config surfaces per project: fingerprint rules, ownership routing, and alert conditions. Each is maintained by hand and tied to the team structure at the time of writing.



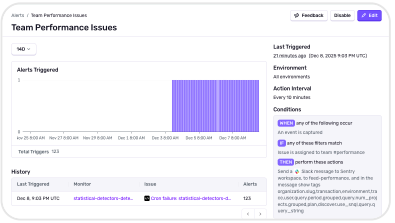
SENTRY: FINGERPRINT RULES

Custom grouping rules that must be written per error type or message pattern. Each rule ties to a specific codebase shape and silently drifts after refactors.



SENTRY: ALERT CONFIGURATION

Each project needs its own alert rules: trigger type, filter conditions, notification routing, and throttle intervals. All configured manually per project.



The most common forms of decay we observed:

Thresholds that aged out. A threshold written at launch reflects a different product's traffic. It now fires on normal load and gets suppressed, or sits too high to catch real regressions.

Routing rules pointing at the wrong people. An engineer moves teams. The ownership rule still names them. The alert lands with someone who cannot act on it.

Fingerprint rules written against a codebase that has since moved. A refactor changes the call path. The old rule now collapses two root causes into one issue, or splits one into three.

Decay the system cannot see. Source map uploads that stopped after a CI migration cause unsymbolicated stack traces. The error count is unchanged, so no alarm sounds.

How Foam addresses this

Foam replaces each decaying config surface with a live-derived equivalent:

Thresholds. The STFT-based anomaly detector scores each error against its own historical baseline, not a manually set number. An error that normally fires 500 times per minute and spikes to 600 is not anomalous. One that normally fires 0 and reaches 3 is.

Routing. Every assignment is re-derived at incident time from the current repository state. Static ownership mappings like CODEOWNERS files inevitably fall out of date as teams reorganize, code moves, and engineers change roles. Foam works around these decaying signals by analyzing actual code contributions, file-level change patterns, cross-service dependencies, and the live on-call rotation to identify who is best positioned to act, regardless of what any static file says.

Clustering. Drain3 templates are learned from the live message stream, not declared ahead of time [2]. As error shapes change, the template tree updates automatically.

04 Alerts are too noisy

Most engineers want to be first to know when they are the ones that broke production. A 2026 survey of 1,039 SRE and DevOps professionals found 77% of on-call teams receive at least 10 alerts per day, 57% say fewer than 30% are actionable, and 44% had an outage in the past year directly linked to a suppressed or ignored alert [5]. Google's SRE Book sets the baseline at two alerts per 12-hour shift [3]. The gap is an order of magnitude.

Where the noise comes from

A 2026 Microsoft and Omdia study of 300 security operations found that 46% of all alerts prove to be false positives and 42% go entirely uninvestigated [29]. The configuration decay described in the previous section (stale thresholds, drifted routing rules) feeds directly into this.

But even in a perfectly configured system, the alerting pipeline itself adds noise [28]:

No investigation before the alert. Incumbent monitoring alerts on event arrival, before any root cause analysis. The on-call engineer receives a raw notification and starts investigating from scratch.

Flapping alerts. Alerts that trigger and auto-resolve in cycles, generating tickets and notifications each time without ever requiring human action.

Duplicate rules across tools. The same failure condition monitored in multiple tools independently, each producing its own alert for a single incident.

Cascading alerts. One upstream failure (a database timeout, a misconfigured deploy) triggers errors across multiple services. Each service alerts separately, flooding the on-call rotation with what is effectively one incident.

How Foam addresses this

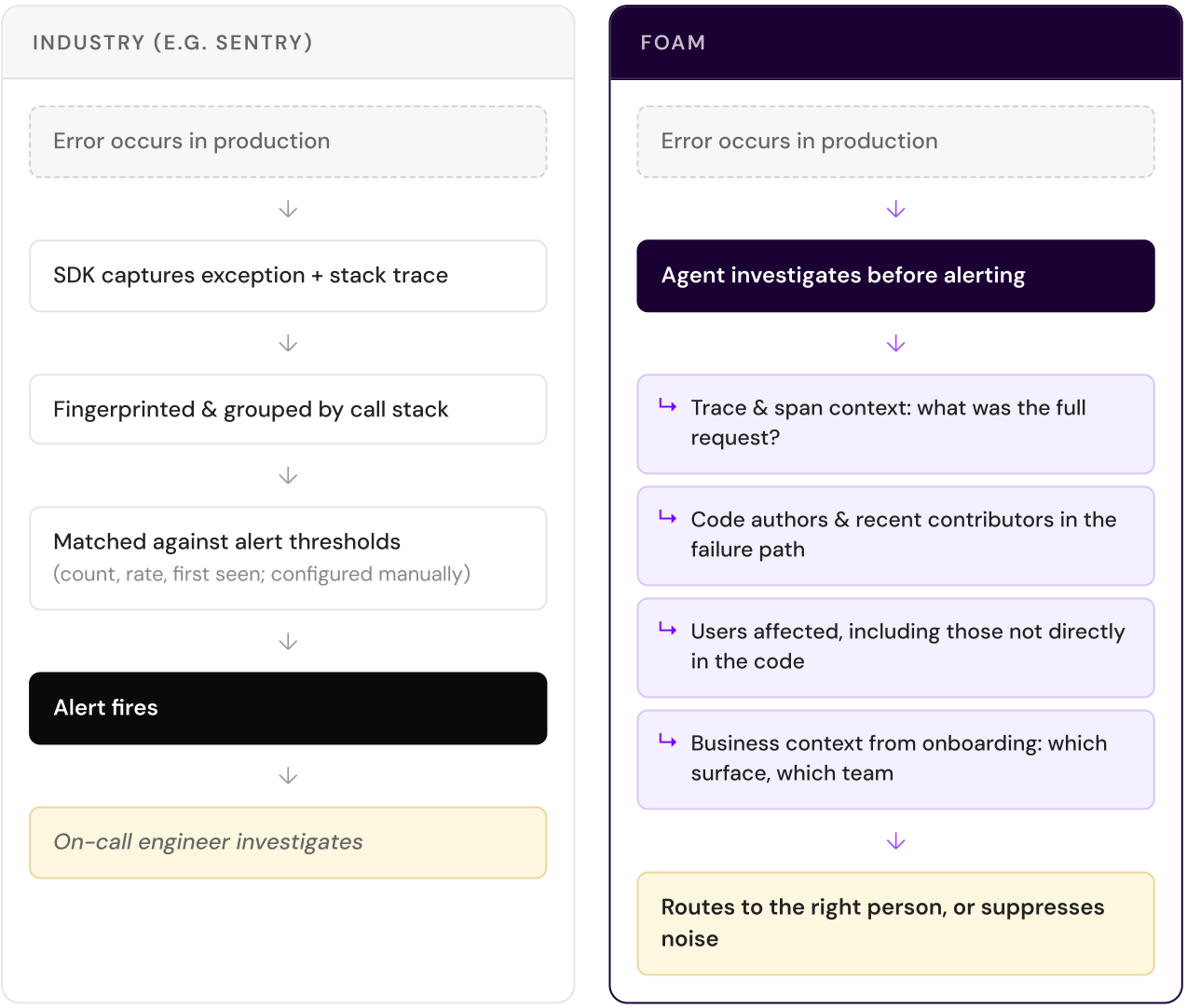
Investigation before alerting. Every alert is sent only after the investigation agent has completed root cause analysis. Each alert carries the root cause summary and a proposed fix.

Cluster-level deduplication. Repeat occurrences of a known cluster do not re-alert. Cascading errors traced to a shared upstream cause produce one alert, not one per service.

Routing derived at alert time. Rather than relying on static ownership rules (which decay, as shown above), the alert goes to the engineer most likely to own the fix,

identified from recent file activity, area ownership, business context, and codebase dependency analysis, not from git blame alone (which fails in over 40% of bug-fixing commits [6]).

ALERT PIPELINE COMPARISON



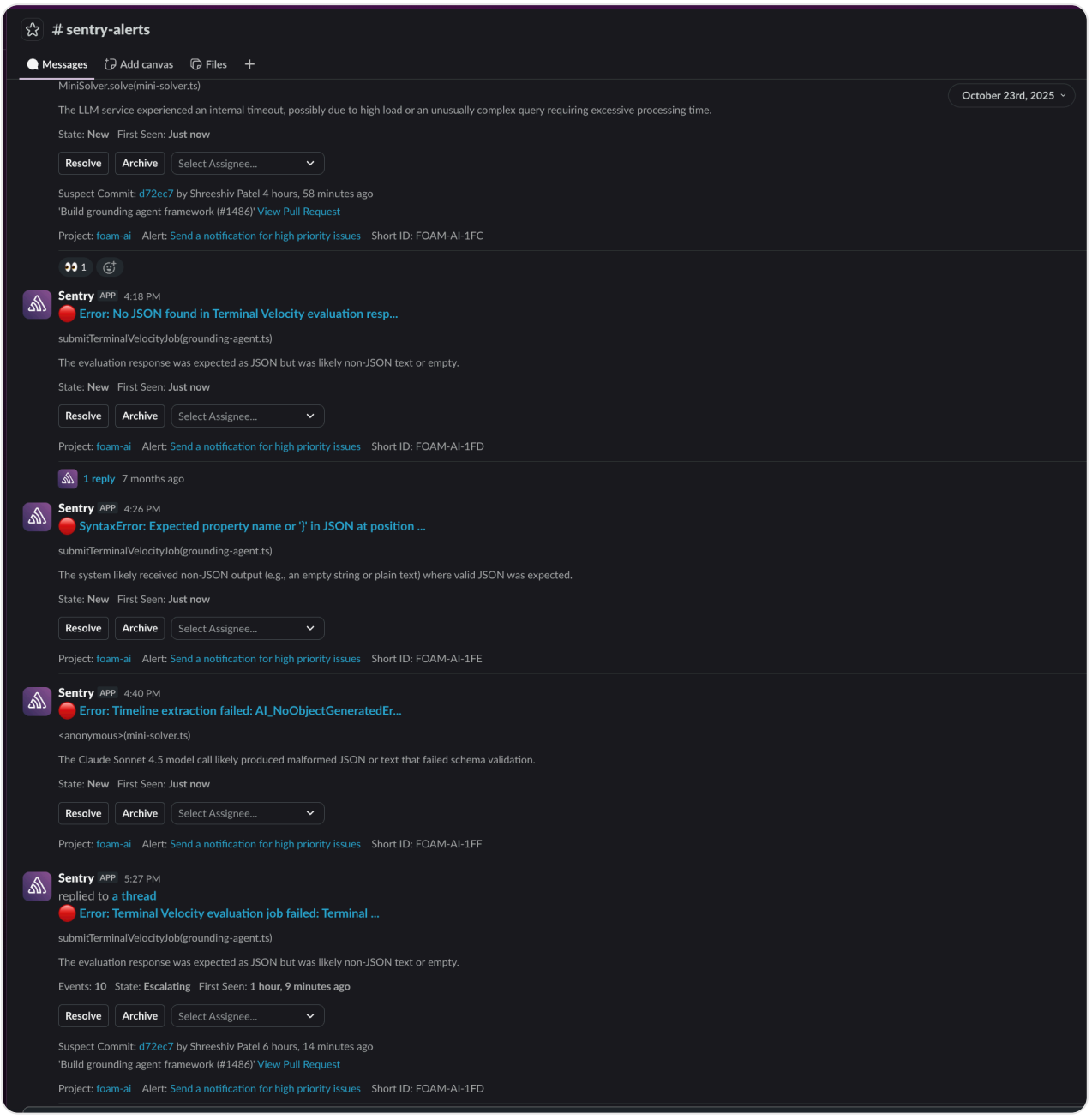
On our own team, we went from dozens of Sentry alerts per day to two Foam alerts across multiple days. Because every alert has already passed through RCA, clustering, and routing before it sends, the signal is high enough to post directly in the team engineering channel instead of a dedicated alerts channel.

> **Most engineers want to be first to know when they are the ones that broke production.**

Foam tags the specific engineer who introduced the regression directly in Slack, with the root cause already identified and a proposed fix attached.

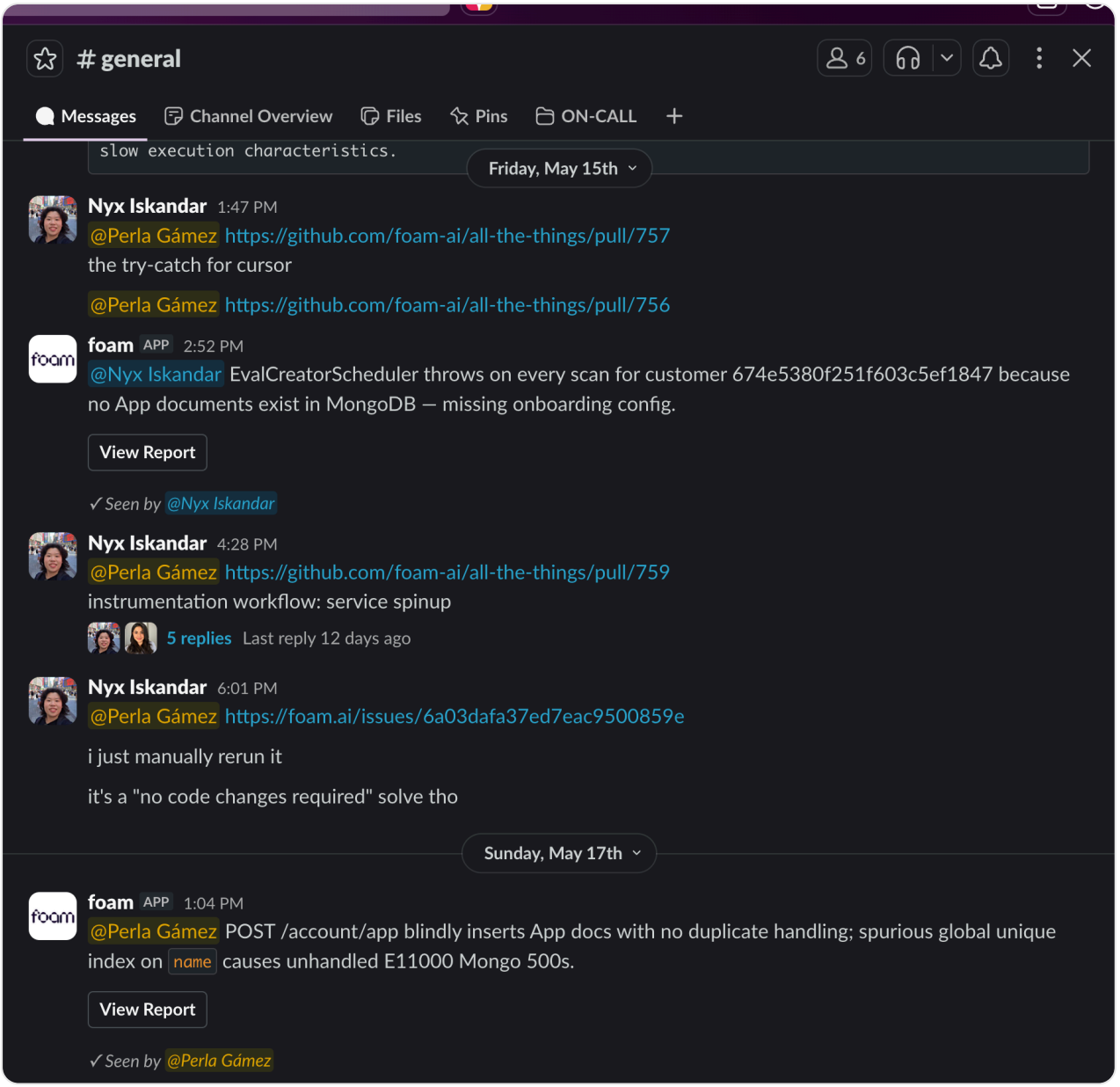
SENTRY: #SENTRY-ALERTS CHANNEL

A single afternoon. Five separate alerts in under two hours, each with Resolve/Archive/Select Assignee buttons and no root cause context. Every one requires a human to open, read the stack trace, decide if it is real, and figure out who should look at it.



FOAM: #GENERAL TEAM CHANNEL

Two alerts across multiple days, posted alongside normal engineering conversation. Each names the responsible engineer, states the root cause, and links to the full report.


general
6
Messages
Channel Overview
Files
Pins
ON-CALL
slow execution characteristics.
Friday, May 15th
Nyx Iskandar 1:47 PM
@Perla Gámez https://github.com/foam-ai/all-the-things/pull/757
the try-catch for cursor
@Perla Gámez https://github.com/foam-ai/all-the-things/pull/756
foam APP 2:52 PM
@Nyx Iskandar EvalCreatorScheduler throws on every scan for customer 674e5380f251f603c5ef1847 because no App documents exist in MongoDB — missing onboarding config.
View Report
✓Seen by @Nyx Iskandar
Nyx Iskandar 4:28 PM
@Perla Gámez https://github.com/foam-ai/all-the-things/pull/759
instrumentation workflow: service spinup
5 replies Last reply 12 days ago
Nyx Iskandar 6:01 PM
@Perla Gámez https://foam.ai/issues/6a03dafa37ed7eac9500859e
i just manually rerun it
it's a "no code changes required" solve tho
Sunday, May 17th
foam APP 1:04 PM
@Perla Gámez POST /account/app blindly inserts App docs with no duplicate handling; spurious global unique index on name causes unhandled E11000 Mongo 500s.
View Report
✓Seen by @Perla Gámez

05 Earning trust in AI-generated fixes

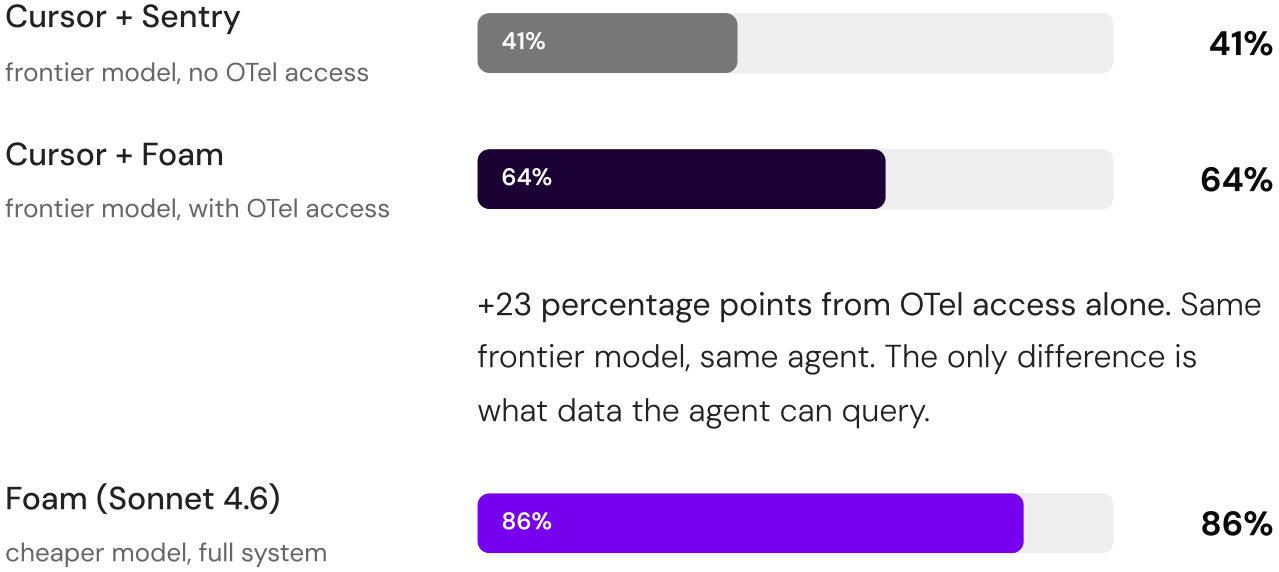
Developers do not trust AI-generated fixes. The 2025 Stack Overflow Developer Survey (49,000+ developers, 177 countries) found that 66% cite "AI solutions that are almost right, but not quite" as their biggest frustration, and more developers actively distrust AI tool accuracy (46%) than trust it (33%) [20]. The PatchTrack study found a median integration rate of just 25% for AI-generated patches [14], a security analysis of 20,000+ LLM-generated patches found that standalone LLMs introduced significantly more vulnerabilities than developer-written patches [13], and Lightrun's 2026 survey found that in 44% of cases where AI tools attempted to investigate production issues, they failed because the necessary execution-level data was not available [15].

Earning trust requires at least two things:

Give the agent the right inputs to drive accuracy. Foam's investigation agent operates against OpenTelemetry data [1] in a database optimized for OTel querying.

Improve and measure accuracy through production evals. Without a benchmark, there is no way to know whether the agent is getting root causes right or just producing plausible-sounding explanations. Teams that deploy AI-driven investigation without measuring accuracy are flying blind. Foam has developed hundreds of evals and continues to create new ones from real production cases. Recently we published the results and human-authored answer keys for 32 of these cases as a public benchmark spanning 8 technical categories [16]. The evals themselves are not published since they are derived from real production environments we are required to keep private; what is public is every answer key and the scoring methodology. Every change to our agents is scored against this benchmark before it ships. Fun fact: we last found that system design matters more than model size. Foam runs on Claude Sonnet 4.6, a smaller and cheaper model than the frontier models used in the baselines, and still outperforms them by a wide margin.

RCA ACCURACY ON THE FOAM BENCHMARK



A cheaper model with the right inputs and system design outperforms a frontier model without them.

Benchmark results and answer keys: github.com/foam-ai/benchmarks [16]

Closing the gaps

These five gaps are not failures of effort. They are the natural result of tools built in a different era, before OpenTelemetry [17] made rich production context universally available and before AI agents could investigate errors autonomously. The data and the techniques to close them exist today.

> **These five gaps are not failures of effort. They are the natural result of tools built in a different era.**

Foam develops its own OTel-native instrumentation packages so that every team, whether they already have OpenTelemetry or are starting from scratch, gets the full benefit of that context from day one. For teams that have invested in OTel, Foam plugs directly into the existing pipeline. For teams that have not, Foam provides a turnkey path to instrumented production services.

References

[1] OpenTelemetry Semantic Conventions — Span, resource, and metric attribute specifications. opentelemetry.io/docs/specs/semconv

[2] Drain3 — Streaming log template miner by IBM Research, based on the Drain algorithm. github.com/logpai/Drain3

[3] Beyer, C. et al. "Being On-Call." *Site Reliability Engineering*, Google, 2016. sre.google/sre-book/being-on-call

[4] Elser, T., Ferge, J. "Using a transformer-based text embeddings model to reduce Sentry alerts by 40% and cut through noise." Sentry blog. Documents Sentry's deployed AI-enhanced issue grouping. blog.sentry.io

[5] NeuBird AI. "2026 State of Production Reliability and AI Adoption Report." Survey of 1,039 SRE, DevOps, and IT operations professionals, February 2026. businesswire.com (press release)

[6] Shi, Y., Li, H., Adams, B., Hassan, A. E. "Beyond Blame: Rethinking SZZ with Knowledge Graph Search." Study of 2,102 validated bug-fixing commits finding over 40% cannot be identified by git blame alone. arxiv.org/abs/2602.02934

[7] Sentry Documentation — Issue Grouping. docs.sentry.io/concepts/data-management/event-grouping

[8] Sentry Developer Documentation — Grouping. Explains Sentry's hybrid hash + AI-embedding grouping pipeline and its open limitations. develop.sentry.dev

[13] Sajadi, A., Damevski, K., Chatterjee, P. "How Safe Are AI-Generated Patches? A Large-scale Study on Security Risks in LLM and Agentic Automated Program Repair on SWE-bench." Security analysis of 20,000+ GitHub issues finding standalone LLMs introduced significantly more vulnerabilities than developer-written patches. arxiv.org/abs/2507.02976

[14] Ogenrwot, D., Businge, J. "PatchTrack: A Comprehensive Analysis of ChatGPT's Influence on Pull Request Outcomes." Median integration rate of 25% across 338 pull requests from 255 GitHub repositories. arxiv.org/abs/2505.07700

[15] Lightrun. "State of AI-Powered Engineering Report 2026." Independent survey of 200 SRE and DevOps leaders conducted with Global Surveyz. globenewswire.com

[16] Foam AI. RCA benchmark results and human-authored answer keys. github.com/foam-ai/benchmarks

[17] OpenTelemetry — An observability framework for cloud-native software. opentelemetry.io

[20] Stack Overflow. "2025 Developer Survey." 49,000+ developers, 177 countries. survey.stackoverflow.co/2025

[21] Short-time Fourier transform — Wikipedia. en.wikipedia.org/wiki/Short-time_Fourier_transform

[22] Sentry Documentation — Ownership Rules. docs.sentry.io/product/issues/ownership-rules

[23] Sentry Documentation — Issue Owners. docs.sentry.io/product/error-monitoring/issue-owners

[24] Sentry Documentation — Code Owners. docs.sentry.io/organization/integrations/source-code-mgmt/codeowners

[27] "RIVA: Leveraging LLM Agents for Reliable Configuration Drift Detection." arxiv.org/abs/2603.02345; Kakarla, S. K. R. et al. "Finding Network Misconfigurations by Automatic Template Inference." NSDI 2020.

[28] Zhao, N. et al. "AlertGuardian: An Industrial Alert Life-Cycle Management Framework." Cloud alert fatigue and denoising at scale. arxiv.org/abs/2601.14912

[29] Microsoft & Omdia. "State of the SOC — Unify Now or Pay Later." Survey of 300 security professionals finding 46% of alerts are false positives and 42% go uninvestigated, 2026. A parallel pattern documented in security operations. microsoft.com/security/blog

FOAM RESEARCH · MAY 2026