Scout 1.0: Post-Training a Model for SRE Investigations

Datadog AI Research[1]

We introduce Scout 1.0, Datadog's first model post-trained for site reliability engineering (SRE) investigations as part of the Bits Investigation agent. We use reinforcement learning (RL) to specialize an open-weights model for investigating production alerts using the same Datadog tools that engineers use to understand their systems. The recipe combines realistic training environments seeded from real-world incidents and real alert telemetry, rewards that encourage useful investigations, and asynchronous RL at scale. We benchmark models using AlertEval, our internal human-expert annotated investigation benchmark. Scout 1.0 medium matches GPT-6.1 Sol high and Claude Opus 5.5 high at 3-8× lower estimated inference cost, and performs comparably to Fable 5.1 medium and GPT-6 Astra medium at 12× and 19× lower cost, respectively. With test-time scaling, Scout 1.0 xhigh matches Fable 5.1 xhigh at 4× lower cost. Results on the public SREGym benchmark show that improvements also transfer to a different harness and task distribution. This report describes our evaluation, training data, RL environments and infrastructure, and RL methodology, and closes with a case study of a real internal incident.

Date: October 15, 2026

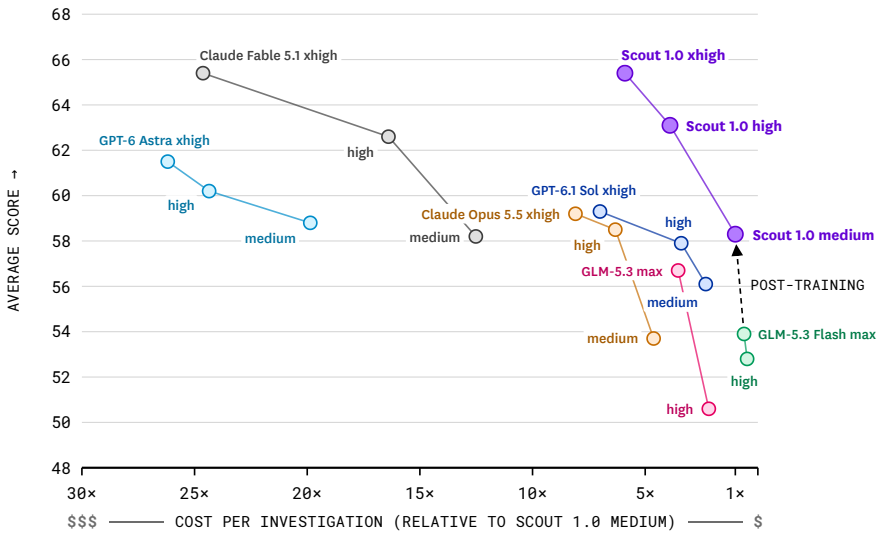

Figure 1 AlertEval quality versus estimated inference cost. Cost is relative to Scout 1.0 medium, and lower cost is to the right. Post-training adds a significant improvement over GLM-5.3 Flash, and test-time scaling [1] establishes a new Pareto frontier for investigations.

[1]See Contributions and Acknowledgments for the full author list. Please send correspondence to airesearch@datadoghq.com.

1 Introduction

We are introducing Scout 1.0, Datadog's first model post-trained for site reliability engineering (SRE) investigations as part of the Bits Investigation agent [2, 3]. We start with GLM-5.3 Flash [4] and post-train it using reinforcement learning (RL) to investigate production alerts through the same tools that engineers use to understand their systems. The training recipe combines realistic environments, rewards that encourage useful investigations, and RL at scale. Notably, we extensively rely on real-world incidents and telemetry from real alerts to build high-quality simulations that are absent customer data but have known causes.

By specializing the model on SRE investigations, we achieve a better price-performance curve on our target scenarios when compared with general-purpose frontier models. We evaluate Scout on AlertEval, our internal human-expert annotated investigation benchmark. In its medium effort mode, Scout achieves a score comparable to GPT-6.1 Sol and Claude Opus 5.5, at 3-8× lower inference cost, and performs comparably to Fable medium and Astra medium at 12× and 19× lower cost, respectively. Our highest-effort mode, Scout 1.0 xhigh, achieves state-of-the-art accuracy that outperforms GPT-6 Astra and matches Claude Fable 5.1 xhigh with 4× less cost. These results taken together show that Scout establishes a new Pareto frontier for Bits Investigation (Figure 1).

In this report, we explore key aspects of our research efforts and results. We cover the following areas:

- **Evaluation** (Section 2). We collected a set of expert-curated incident data that combines real alerts, archived telemetry, and reference incident explanations. Our AlertEval benchmark consists of a subset of this data, along with an LLM-based judge that assesses agentic performance via the accuracy and depth of its causal findings. Additionally, we share results on the public SREGym benchmark [5] in order to show generalization to a different harness and task distribution.
- **Training data** (Section 3). We generated investigation scenarios with known root causes at scale. Crucially, in order to build realistic simulations, we rely on real-world incidents and telemetry data from real alerts to seed these simulations and preserve the causal structure of real-world production problems, without training on customer data directly.
- **RL environments and infrastructure** (Section 4). We built interactive training environments that reproduce the Bits Investigation interface, allowing agents to practice realistic tool use at scale. We built our asynchronous RL stack from a fork of SkyRL [6] running on Ray [7], making substantial enhancements to support production-scale post-training and improve throughput and reliability.
- **RL methodology** (Section 5). We use a modified version of GRPO [8] that accommodates stale rollouts and mitigates overly long investigations. To promote transfer to production, we use a hybrid reward that balances correctness, exploration, and evidence grounding, and employ "harness fuzzing" to vary optional tools and interaction budgets during training.
- **Case study** (Section 6). We present an internal incident, a Postgres autovacuum alert caused by a stale replication slot, to illustrate the challenges of SRE investigation. We show how Scout performs a complex root cause analysis by connecting evidence across components and time, testing competing hypotheses, and reconstructing a causal chain from symptom to underlying cause.

2 Evaluation: Replaying Real Investigations with AlertEval

Existing investigation benchmarks [5, 9–12] are either small scale, fully synthetic in nature, or both. The need for a comprehensive benchmark led us to embark on a company-wide effort to collectively record internal incident evaluation data [13]. Because we need domain experts to evaluate complex systems, our engineering infrastructure and internal teams have directly curated thousands of evaluation points. To achieve this goal, we incorporated annotation tooling directly into our core incident-response workflow. Furthermore, as customers interact with Bits Investigation, we continuously use their feedback to inform new evaluation scenarios, expanding the diversity of environments, edge cases, and failure modes our benchmarks cover.

One of the byproducts of our data collection efforts is *AlertEval*, an internal benchmark for SRE investigations. It combines real alerts, archived telemetry, and expert human labels for incident causes. The labels provide a

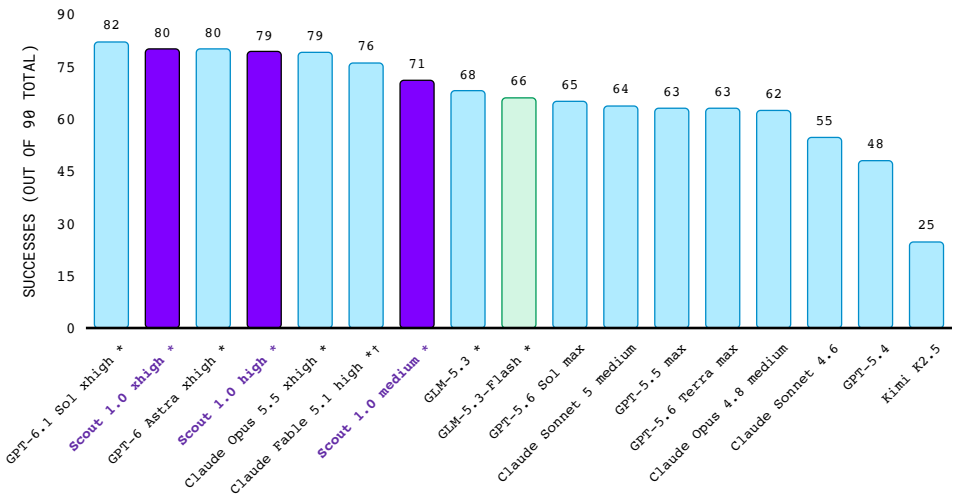

Figure 2 SREGym (90 faults): faults resolved end to end. Each bar counts the faults that a model diagnoses and mitigates correctly, at different test-time effort levels. Scout 1.0 medium improves significantly on the base GLM-5.3 Flash model and on GLM-5.3. With test-time scaling [1], Scout 1.0 high is competitive with current frontier models.

*Our runs, one attempt for each fault. Other rows are reproduced from the sregym.com leaderboard. Each model shows its best agent harness. †For some tasks, Claude Fable's safety classifier declined a request. For these, we fall back to Claude Opus 5.5 at the same effort level.

reference explanation to compare with a candidate model's findings. We train on simulated investigations generated by GenSim, our simulation framework described in Section 3, and evaluate on real-incident investigations through AlertEval. This tests whether the learned investigation behavior transfers beyond the simulated environments used for training.

We evaluate the model's findings with "DeepJudge," our own rubric scored by LLM-as-judge [14], which compares an investigation with the reference label. It scores both the immediate cause and deeper causal findings. It also penalizes contradictions and spurious findings that are absent from the reference.

We report the average DeepJudge score on a 0–100 scale. The rubric awards partial credit, so a score of 57.5 does not mean that the model solves 57.5 percent of incidents. It measures how well the findings match the labeled explanation under this rubric, including both the main root cause finding as well as the auxiliary deeper findings.

2.1 SREGym

In order to measure how well Scout generalizes and ensure it has not fit to specifics of the harness or task distribution we have targeted, we also evaluate using the publicly available SREGym benchmark [5]. In particular, we evaluate on the full benchmark (90 tasks, no noise) using the supplied Stratus harness [15]. We additionally test the latest versions of Opus, Sol, Fable, and Astra in order to augment the baseline provided in the public leaderboard.

We find that Scout 1.0 medium significantly improves upon the performance of the base model GLM-5.3 Flash, surpassing GLM-5.3 at more than 4× reduction in cost. Through additional test-time scaling [1], we find Scout 1.0 xhigh is competitive with the best models on this benchmark, while still maintaining at least 2× cost savings compared to all competitors (10× savings compared to Astra- and Fable-class models) (Figure 2).

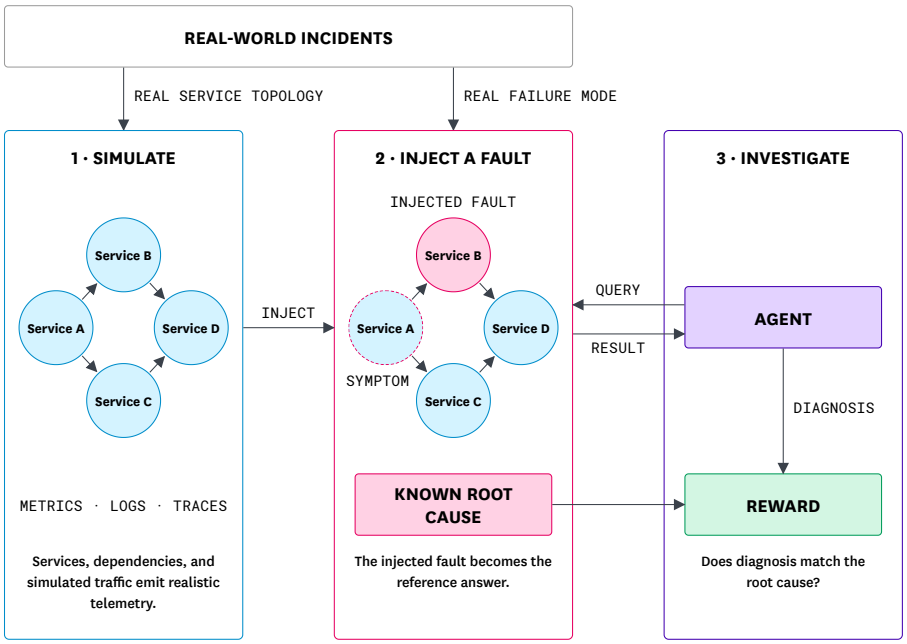

Figure 3 Constructing a verifiable investigation task in GenSim. GenSim produces an alert and queryable telemetry from the injected fault. The investigation receives a reward based on how well it explains that fault.

3 Training Data: Building Realistic Investigation Scenarios

While AlertEval provides us with a rigorous benchmark, using a similar annotation process to generate training data would expose two key bottlenecks: obtaining unambiguous ground truth answers through expensive expert annotations, and acquiring all associated telemetry for each incident. AlertEval addresses these challenges by relying on human experts to annotate specific real-world incidents where selected telemetry is explicitly archived for long-term retention. However, scaling this approach for RL training is prohibitively costly. We need a way to generate realistic investigation scenarios with verifiable ground truth at scale without relying on manual labeling or requiring historical telemetry archives.

To do this, we built GenSim, a simulation framework that allows us to deploy realistic behavioral twins of the types of applications our customers use and to deterministically inject realistic failure modes based on what we see in the wild, in the spirit of chaos engineering [16]. With GenSim, our operational experience guides which applications, faults, and evidence conditions we simulate. Each investigation scenario includes an alert, the associated telemetry, and a reference root cause (Figure 3).

Using GenSim, we can translate a real-world incident into a fully replayable investigation scenario even if its associated telemetry is no longer available. Furthermore, while real-world incidents can contain a significant volume of telemetry that distracts from the investigation, their archived representations do not retain all of this telemetry and are much sparser. In contrast, because GenSim is fully controllable, we can deliberately tune the signal-to-noise ratio to create easier or harder investigation scenarios for training.

3.1 From Postmortem to Simulation

For all GenSim environments, we attempt to match the real-world distribution of production alerts. For some examples we go further and seed the simulation with postmortems to replay specific real-world incidents. We employ two different methods for translating these postmortems into simulations (Figure 4).

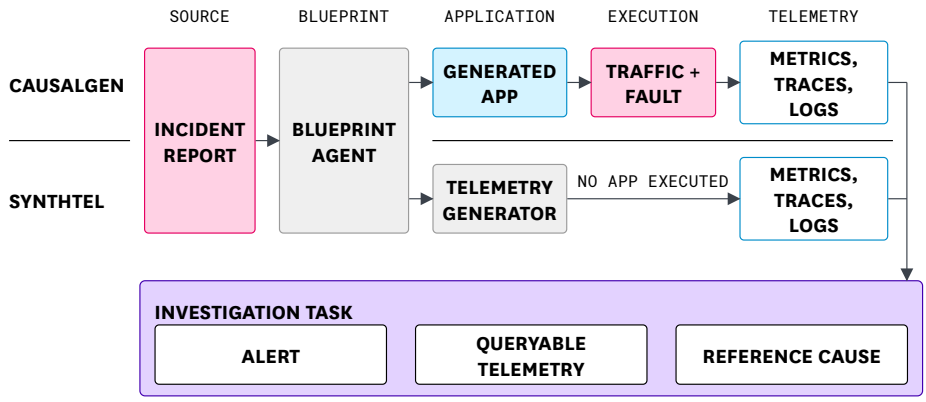

Figure 4 GenSim turns real incidents into investigation tasks. CausalGen runs generated applications under traffic and faults. SynthTel generates the telemetry directly. Both produce an alert, queryable evidence, and a reference cause. In practice, we train on a curated mix of both kinds of GenSim data.

CausalGen. CausalGen starts by agentically extracting the causal structure of a real-world incident. It converts that structure into a scenario blueprint, i.e., a structured description of the environment, the triggering event, and the full causal sequence between the trigger and the alert. A specialized agent then uses that blueprint to implement and deploy a **fully working distributed system** in a sandboxed cluster. We then trigger the incident chain from the initiating fault to the observed symptom. This produces the telemetry that the model then investigates during training.

SynthTel. SynthTel also starts from a blueprint, but unlike CausalGen, it generates the telemetry directly. Its agent produces code that directly outputs coherent logs, metrics, traces, and related context without running the underlying application. This gives us another way to represent an incident mechanism as an interactive investigation scenario.

Both approaches require consistency across the simulated evidence. Services and timestamps must agree, and the different signals must reflect the same fault. Where a scenario requires baseline or recovery behavior, the telemetry must represent those periods as well. We apply extensive quality checks for causal and temporal consistency.

These two approaches allow us to trade off **cost** against **fidelity**. SynthTel scenarios are faster and cheaper to generate at scale, but may produce less realistic telemetry. CausalGen, meanwhile, produces accurate telemetry by running in real time, with real components, just as an actual system would. It is therefore bottlenecked by compute and by the wall time of the simulation.

3.2 Data Mix and Curation

We enforce causal and temporal consistency in our data generation pipeline. This is only a starting point, though, because a coherent incident can still make a poor training scenario. The relevant evidence may be missing from the archived telemetry, or the answer may leak through the alert description. We therefore treat validity, difficulty, and coverage as separate curation problems.

One important feature of useful scenarios is *solvability*: we look for scenarios where the necessary evidence is actually visible to the agent through the tools. Some scenarios may simply lack a sufficient degree of solvability to provide a good learning signal. On the other end of the spectrum, we avoid leaking simulation details in alerts, service names, metadata, or diagnostic messages that may accidentally give away the answer.

In practice, the line between helpful diagnostic context and a leaky simulation abstraction can be thin. For example, real-world alerts often include a "runbook," a list of recommended troubleshooting steps. If a

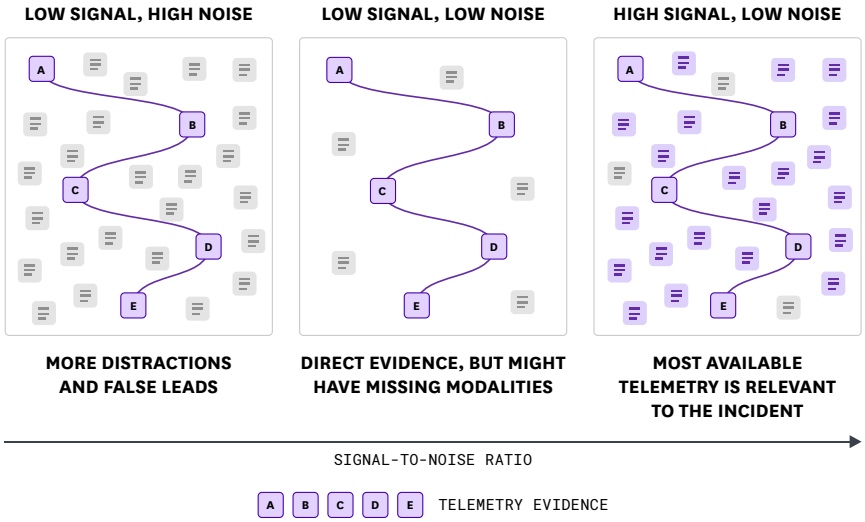

Figure 5 Telemetry volume alone is a poor proxy for difficulty. Additional telemetry can add distractions or corroborating evidence, and sparse telemetry can limit the available context. Controllable simulations let us vary telemetry density and signal-to-noise ratio around a known incident. The highlighting is for illustration only. The agent must find the relevant evidence itself.

simulated runbook becomes too detailed, the scenario can become trivial. With the help of automated heuristic checks and adversarial agents, we scan the generated scenarios and investigation trajectories for shortcut opportunities: signals that could inflate scores without teaching useful investigation behavior, a form of reward hacking [17, 18].

We favor scenarios that require deep investigation, where the agent must reason and comb through irrelevant or ambiguous evidence. Working backward from a known root cause can make it tempting to include only the telemetry that explains the failure. But telemetry from unaffected components can also provide useful evidence, helping the agent rule out competing plausible explanations (Figure 5).

Beyond individual scenario validity, we consider whether a scenario is learnable for the current model and additive to the existing dataset. A learnable scenario offers meaningful learning headroom without being effectively impossible for the current setup. As in prior work on difficulty filtering [19], we screen scenarios through repeated rollouts and remove those the model solves consistently. This difficulty estimate is specific to the model, investigation interface, judge, and sampled rollouts, so we treat it as a setting-specific curation signal rather than an intrinsic property of the input scenario.

An additive scenario expands coverage instead of repeating an already well represented application-fault combination. Datadog's experience across varied production systems guides which architectures, technology stacks, scales, and observability setups our simulations should cover. The training telemetry itself is generated within our simulated environments.

The same type of fault in a different underlying system can have a vastly different presentation and require a different investigation path. Service topology and instrumentation can also change the evidence available to the agent. We therefore measure coverage across the product of system types and faults (Figure 6). In order to preserve at least a minimal sample of each fault-application pair after applying difficulty filtering, we monitor the retained data mix and, if needed, rebalance it by optimizing between coverage and data quality. This ensures that focusing on more challenging examples does not eliminate critical areas of the exploration space, allowing us to preserve essential diversity along both axes.

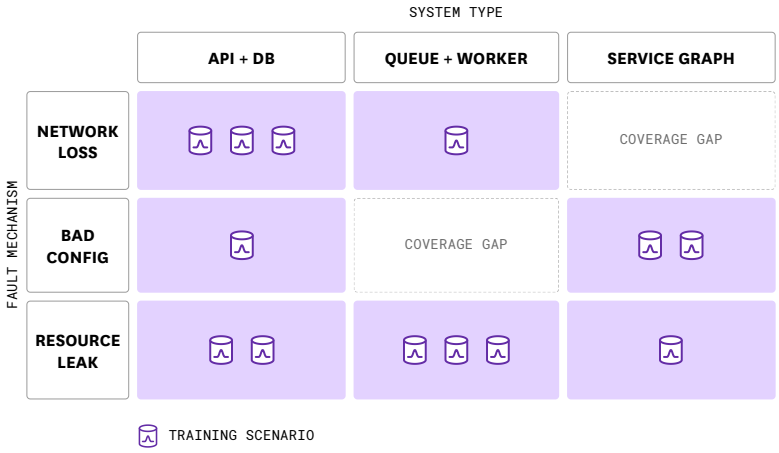

Figure 6 Balancing difficulty with coverage. The same fault can require different investigation paths across system types. Each icon is one training scenario. Empty cells show that covering each dimension separately can still leave combinations missing, so difficulty filtering must preserve coverage across both dimensions. Counts are illustrative.

4 RL Environments and Infrastructure

We built our own training infrastructure on top of open-source tools to train Scout. We next describe the investigation environment and model-training systems needed to run reinforcement learning at scale.

4.1 RL Environments that Reproduce the Bits Investigation Interface

GenSim creates realistic investigation scenarios with their associated telemetry, but in order to perform RL, we need to make this data available in an interactive environment that the model can query during training rollouts. To do so, we created a sandboxed backend that reproduces the Datadog tools and APIs needed for investigation and answers requests from archived scenario data.

An investigation begins with the alert and monitor context, which establish the affected service and time window. The RL environment then allows the model to query metrics, search logs, inspect traces, follow dependencies, track changes (deployments, feature flags, etc.), inspect code, and leverage many other tools available within the Datadog platform.

We keep the training environment as close as possible to the real production environment. Training a model in the specific harness it will be used in confers a significant performance advantage. The model encounters the specific tool names, arguments, and response structures that it needs in Bits Investigation. It learns how to ask for evidence through that interface and interpret the results.

Each investigation receives its own scenario context and data scope. Sessions share immutable archives while their query results and investigation state remain isolated. A local query service reads the archived telemetry from an isolated data store. This design separates repeated training queries from live production systems and reduces dependence on remote services. It also lets us scale simulation compute needs alongside training. The same archive can support many investigations that ask different questions in different orders.

Query syntax, empty results, limits, and errors all influence the model's next decision. If a query returns no matching logs, the model needs to interpret that result and choose a useful follow-up. These interactions make the environment a place to practice investigation, rather than a collection of questions with fixed answers.

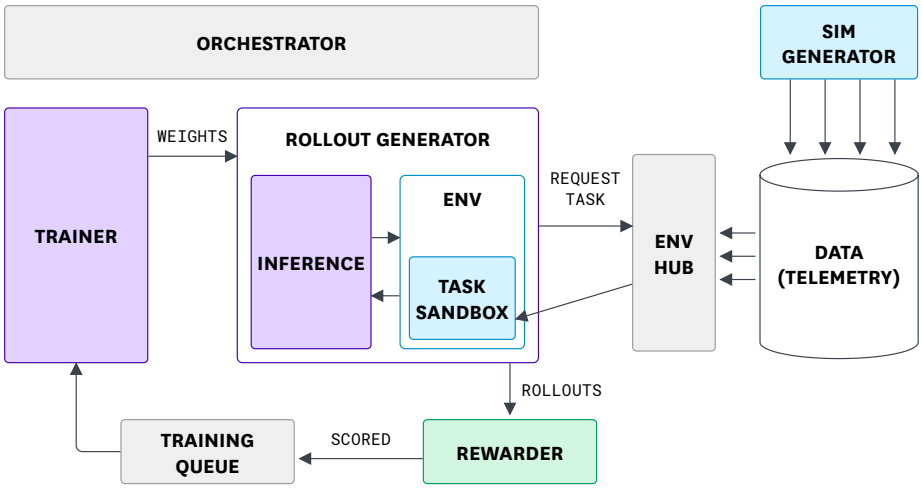

Figure 7 How the RL stack connects training, rollouts, and simulations. Rollout workers conduct investigations, and the local simulation sandbox serves their tool requests from isolated scenario data. Judges score completed investigations, and the system assembles eligible groups for the trainer. Updated weights return to the generation workers.

4.2 An Internal Stack for Large-Scale Post-Training

To train Scout, Datadog built an internal RL stack for scalable post-training. Like most other recent post-training efforts [20–22], we use asynchronous (async) RL, where training and inference are disaggregated and concurrent. We allow controlled off-policy updates via a staleness budget [21, 23]. The RL training pipeline itself is thus a complex distributed system that must orchestrate many different components (Figure 7).

Our platform builds on top of the open-source RL ecosystem, starting with a fork of the popular SkyRL framework [6] running on Ray [7]. We worked closely with the SkyRL team throughout, benefiting from their feedback and guidance, and have upstreamed several improvements back to the project. To productionize our post-training runs and optimize them for throughput and reliability, we also made substantial enhancements beyond the open-source base, discussed in detail in Appendix A.

5 RL Methodology

We next describe the adapted version of GRPO we use for learning, as well as our reward shaping and harness fuzzing efforts to promote transfer to our production environment.

5.1 Learning Algorithm

We use async RL as in most recent agentic training setups [20–22], which also informs our algorithm choice. Our algorithm is based on Group Relative Policy Optimization (GRPO) [8], but uses importance sampling and clipping based on the inference policy's distribution π_{rollout} dividing our current trainer policy π_θ (with asymmetric clipping bounds ϵ_ℓ, ϵ_h [24]), as proposed in the GLM-5 technical report [22]. In order to mitigate overlong trajectories, we adapt the loss averaging and divide by a constant maximal length L instead of the rollout length, as proposed in Dr. GRPO [25]. Our loss formulation is thus as follows:

$$\mathcal{L}(\theta) = -\mathop{\mathbb{E}}_{\substack{x\sim\mathcal{D} \\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{rollout}}(\cdot|x)}} \left[\frac{1}{GL} \sum_{i=1}^{G} \sum_{t=1}^{T_i} \text{clip}\left(\frac{\pi_\theta(y_{i,t} \mid x, y_{i,:t})}{\pi_{\text{rollout}}(y_{i,t} \mid x, y_{i,:t})}, 1-\epsilon_\ell, 1+\epsilon_h \right) A_i \right], \tag{1}$$

where the advantage A_i is defined as the standardized reward with respect to the group of size G sharing the scenario, x is the prompt (and environment) sampled from our set of tasks $\mathcal{D}$, and T_i is the length of the

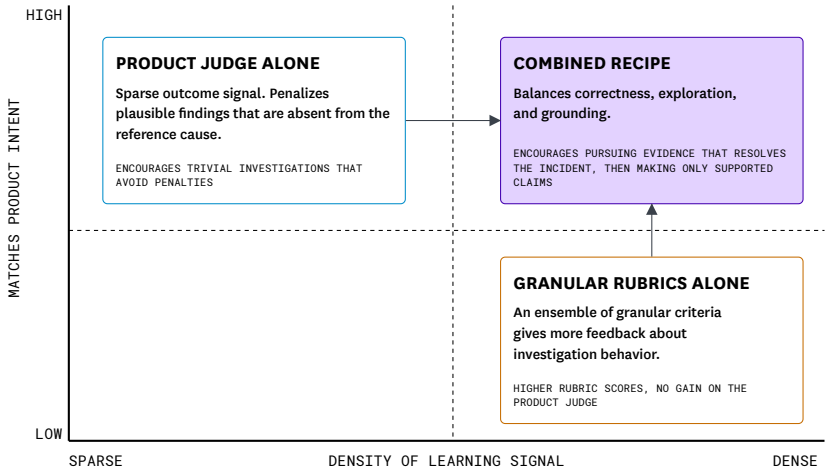

Figure 8 A useful reward matches product intent and gives a dense learning signal. A conceptual view of reward design. Our combined recipe balances correctness, exploration, and grounding. The positions are illustrative, not quantitative ablation results.

generation y_i. In (1) we ignore tool use, but we did train with tools in our rollouts. Tool output tokens were masked out in the loss [26]. As in DAPO [24], we do not apply any direct KL penalty, and we do not use entropy regularization.

5.2 Aligning Generation with the Learning Calculation

Aligning the distribution of our inference engine to our trainer is crucial because it allows us to train on policy. While some of the misalignment comes from stale rollout weights and is irreducible, a significant part also stems from other differences between inference engines and the trainer, like expert routing and numerical differences in the prediction head. To improve numerical stability, we compute the prediction head in fp32, as in MiniMax-M1 [27]. We also employ Rollout Routing Replay (R3) [28] to eliminate differences due to expert routing (Appendix A.3).

5.3 A Useful Evaluation Signal Is Not Always Enough for Learning

In our experiments, we found that reward alignment between the training environment and downstream product evaluations was crucial. DeepJudge measures whether the final findings match the reference cause, but that relatively sparse signal provides limited guidance about how to investigate. Its penalties for additional findings also affect what the model learns to report.

When we used this signal for training, the model could reduce its exposure to penalties by limiting its investigation and its claims. We observed a tendency toward trivial investigations, a form of reward hacking [18]. The lesson was that correctness needs to work alongside incentives to pursue relevant evidence.

We also studied denser feedback through an ensemble of granular rubrics [29]. However, higher scores on the granular rubrics alone did not translate into better scores on AlertEval when assessed by DeepJudge. The extra guidance needed to remain connected to the outcome that mattered in the product. Measuring more aspects of behavior did not, by itself, establish that connection.

After many training ablations, we developed a specialized hybrid reward signal that balances correctness, exploration, and grounding (Figure 8). The desired behavior is to pursue evidence that correctly diagnoses the incident, then make the claims that this evidence supports.

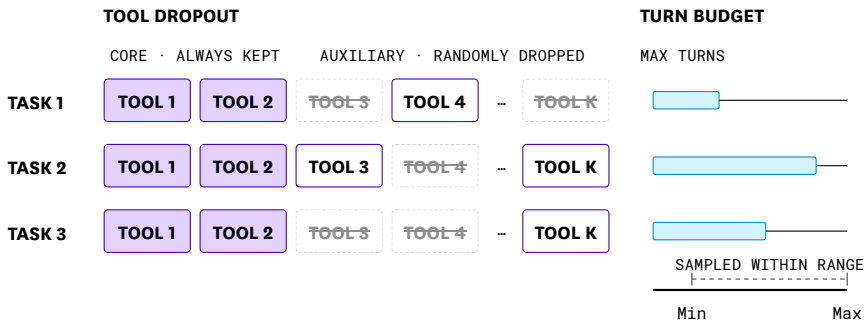

Figure 9 Harness fuzzing changes the tools and turn budget for each task. Optional tools are dropped at random, and the turn budget is sampled within a range. Core tools remain available, and investigations compared within a learning group receive comparable conditions.

5.4 Adapting to Different Tools and Budgets

Another hurdle in achieving generalization to the production setting was due to mismatches in tool surface: even though we do our best to mimic the production toolset, there are some tools and data sources that we cannot reproduce in the local sandbox environment. And even if we could perfectly match the current production harness in training, the production tools are always changing as we ship improvements to our Model Context Protocol (MCP) [30] server and other internal APIs. A good model needs to be robust to some amount of variability in prompts and available tools.

To make the model more adaptable to this type of drift, we employ "harness fuzzing" techniques to vary the conditions of an investigation during training (Figure 9), similar in spirit to domain randomization for sim-to-real transfer [31]. Tool dropout randomly removes selected optional tools or tool groups, while turn-budget randomization changes how many interaction rounds the model can use.

These changes encourage the model to adapt its strategy. When a tool is absent, it needs another route to the evidence. When the budget is shorter, it needs to prioritize the questions that can most directly resolve the incident and avoid deeper investigations into potentially tangential signals.

We preserve the core tools needed to keep the scenario solvable. We also keep conditions comparable within each group of investigations used for the same GRPO policy update. This lets us vary the scenario conditions while retaining a meaningful comparison between sampled attempts.

6 Case Study: Bits and Scout on a Real Incident

We now look at a real internal incident at Datadog that was correctly diagnosed by Bits using Scout. This case study helps to highlight some of the properties of SRE investigations that make them inherently challenging.

Investigation combines aspects of a deep research problem with long-horizon reasoning across many steps. The agent must find a small amount of useful evidence within a large volume of telemetry. It must then use that evidence to distinguish competing explanations.

The agent conducts its investigation through specialized tools requiring highly technical domain expertise. It reads the alert, forms hypotheses, queries telemetry, and decides upon its next action as evidence arrives. The investigation can span logs, metrics, traces, service dependencies, recent changes, and many other sources of production telemetry stored in an observability platform like Datadog.

To succeed, the agent must have a correct "mental model" of the underlying distributed system. It must reason correctly in order to connect its observations and form a valid root cause analysis (RCA) [32, 33].

6.1 A Symptom Can Be Several Steps Away from Its Cause

The investigation in our case study begins with an alert. The visible symptom appeared in Postgres: all 30 autovacuum workers were busy for more than an hour, while dead rows accumulated. Autovacuum is the background process that cleans orphaned and deleted database tuples, thus preventing disk space from filling up unnecessarily [34].

Importantly, the initiating event occurred about two hours before the alert, with a Kubernetes node rotation. A node drain [35] removed a read replica and its local disk. The replacement replica started on another node with an empty disk. Meanwhile, the primary retained the old replication slot, which kept its cleanup horizon from advancing [36]. Dead rows accumulated, and autovacuum workers stayed busy without completing the necessary cleanup (Figure 10).

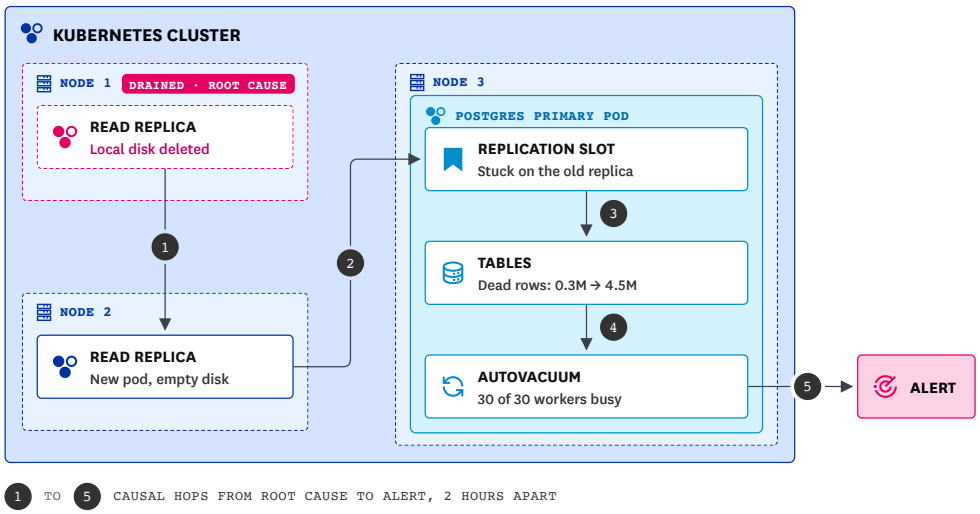

Figure 10 Five causal hops separate a node drain from the alert. The chain runs from a Kubernetes node drain to the Postgres autovacuum alert. The initiating event and the visible symptom occur on different components, about two hours apart.

Crucially, the observed symptom occurred multiple hops away from the root cause. No single observation explains the incident. The agent must connect the alert to database behavior, replica state, and an earlier infrastructure event. A query for recent database errors can return relevant evidence, but it cannot establish the entire chain.

6.2 Working Backward from the Alert

Bits Investigation with Scout connected the autovacuum alert to the earlier node drain. The investigation found increased dead rows and database scanning without a corresponding increase in query counts. It also found a replacement replica that was not ready, repeated requests for unavailable replication logs, and the preceding node event.

Together, those observations supported a stale replication slot as the cause that prevented cleanup from advancing. The investigation worked backward from the symptom and tested alternatives along the way. Figure 11 shows the evidence and decisions that led to the explanation, and Appendix B shows the resulting conclusion in the Bits Investigation interface.

This example motivates the training problem. We need environments that support alternative paths through realistic evidence. We also need rewards that distinguish a supported causal explanation from a plausible story.

Figure 11 The run worked backward from the alert to a drained node. Selected steps from the investigation of the autovacuum incident. The agent connects metrics, logs, replica state, and the node drain to explain why cleanup stalled. The final step reports a diagnosis and proposed recovery actions.

7 What's Next

Scout shows how specialization can improve the balance of investigation quality and inference cost. We train agents on environments built from real incidents, give them realistic tools to investigate what went wrong, and reward them for reaching useful conclusions.

Our internal RL stack gives us a foundation for further research. We are now building the next version of the platform. We plan to expand compute capacity, improve simulation coverage, and study new data mixtures and reward designs. We also plan to extend this research toward remediation and other Datadog use cases, with evaluations suited to each scenario. As we do so, we plan to incorporate additional specialized modeling techniques for non-text modalities, including time series [37, 38], traces, and more.

Contributions and Acknowledgments

Core Contributors

Listed alphabetically.

Othmane Abou-Amal, *Project Lead*
Abdelhakim Benechehab
Bogna Blaszczyk, *Data & Environments Lead*
Lucas Braz
Etienne Brodu
Ben Cohen, *Project Lead*
Quentin Gendre Robin
Oskar Hallström
Howard Huang, *RL Infrastructure Lead*
Shridhar Kumar
Samuel Müller, *RL Methods Lead*
Richard Nguyen
Raphael Olivier
Sergey Pastukhov
Afshin Rostamizadeh, *Project Lead*
Satya Narayan Shukla
Ameet Talwalkar, *Project Lead*
Laetitia Teodorescu
Bruno Volpato, *RL Infrastructure Lead*
Xunyi Zhao
Yilun Zhou

Contributors

Listed alphabetically.

Loren Arthur
Walid El Bouchikhi
Zhengda Lu
Charlie Monzillo
Arun Parthiban
Suhaas Reddy
Daniel Shan
Kan Wang
Madison Xu

© Datadog 2026

Acknowledgments

Our work is made possible by the efforts of numerous teams at Datadog and beyond. Special thanks and acknowledgment to:

Jean-Baptiste Abbadie, Luca Abbati, Ahmed Abdulkadir, Oliver Allweyer, David Asker, Samer Atiani, Benjamin Barton, Guillaume Bertrand, Vincent Brisse, Maxim Brown, Brianne Bujnowski, Benedetto Buratti, Clémence Burnichon, Alessandro Cappelli, Lia Castaneda, Alexandre Castera, Ching-Wei Chen, Pawel Chojnacki, Victor Cojocaru, Nicole Danuwidjaja, Guillaume Deacken, Murat Duman, Robin Eisenberg, Rikki Endsley, Spencer Ewall, Jake Femminineo, Justin Francis, Kit Freddura, Jessica Fu, Christian Garcia-Neal, Raphaël Gavache, Azat Gaynutdinov, Elizabeth George, Dan Green, Tyler Griggs, Yash Gupta, Sumanth Hegde, Daniel Hesslow, Amber Hu, William Huang, Alan Hudson, Matthias Huels, Joe Jones, Donghwan Kim, Hannah Kim, Shijin Kong, Boris Kopin, Emaad Khwaja, Julien Launay, Chris Laverdiere, Alexis Lê-Quôc, Yanbing Li, Avi Lissack, Max Livingston, Dengke Liu, Savita Manghnani, Etienne Maugars, Carlysle McNaught, Raphaël Meudec, David Mitchell, Zubairuddin Mohammed, Nico Monchy, Madison Moore, Jessie Nadler, Sesh Nalla, Matthieu Neau, Scott Opell, Paul Osman, Baptiste Pannier, Laurent Paris, Jonathan Parisot, Olivier Pomel, Pinghao Qi, Ani Ray, Thomas Renault, Ibrahim Ridene, Cristian Rodriguez, Charlie Ruan, Laxmi Sravya Rudraraju, Omri Sass, Anushka Saxena, Gabriel Schwab, Karim Siala, Jordan Singleton, Astraea Skoutelli, Mark Spicer, Rob Tandy, Eric Tang, Michael Troehler, Geneva Vallecilla, Mitch Ward, Michael Whetten, Piotr Wolski, Gerald Woo, Reilly Wood, Scott Yak, Carl Yeksigian, T Zhang, Congyao Zheng.

References

[1] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

[2] Daniel Shan and Tristan Ratchford. How we built an AI SRE agent that investigates like a team of engineers. Datadog blog, https://www.datadoghq.com/blog/building-bits-ai-sre/, 2026.

[3] Dan Green, Kai Xin Tai, and Brianne Bujnowski. Meet the new Bits Investigation: Deeper reasoning, twice as fast. Datadog blog, https://www.datadoghq.com/blog/bits-ai-sre-deeper-reasoning/, 2026.

[4] Z.ai. GLM-5.3-Flash-BF16 model card. https://huggingface.co/zai-org/GLM-5.3-Flash-BF16, 2026. Revision a5b45eb41df6402735dedc900be14a42e8d5e538, accessed 2026-10-13.

[5] Jackson Clark, Yiming Su, Saad Mohammad Rafid Pial, Yifang Tian, Lily Gniedziejko, Hans-Arno Jacobsen, Yinfang Chen, and Tianyin Xu. SREGym: A live benchmark for AI SRE agents with high-fidelity failure scenarios. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026. URL https://openreview.net/forum?id=A5bNdaH8CI.

[6] Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. SkyRL-v0: Train real-world long-horizon agents via reinforcement learning. https://github.com/NovaSky-AI/SkyRL, 2025. Our fork is based on commit 29f0fdf9291aa17f80af85e065fa83e579fe56c5, vendored on 2026-05-13.

[7] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 561–577, 2018. URL https://www.usenix.org/conference/osdi18/presentation/moritz.

[8] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

[9] Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. OpenRCA: Can large language models locate the root cause of software failures? In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=M4qNIzQYpd.

[10] Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. RCAEval: A benchmark for root cause analysis of microservice systems with telemetry data. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 777–780, 2025. doi: 10.1145/3701716.3715290.

[11] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/d1f9e4a9f109b6e8b75ed362736f22ec-Abstract-Conference.html.

[12] Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, et al. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=jP59rz1bZk.

[13] Benjamin Barton. How we built a real-world evaluation platform for autonomous SRE agents at scale. Datadog blog, https://www.datadoghq.com/blog/engineering/bits-ai-eval-platform/, 2026.

[14] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

[15] Yinfang Chen, Jiaqi Pan, Jackson Clark, Yiming Su, Noah Zheutlin, Bhavya Bhavya, Rohan Arora, Yu Deng, Saurabh Jha, and Tianyin Xu. STRATUS: A multi-agent system for autonomous reliability engineering of modern clouds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=fYW1PKawwJ.

[16] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016. doi: 10.1109/MS.2016.60.

[17] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016. URL https://arxiv.org/abs/1606.06565.

[18] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/3d719fee332caa23d5038b8a90e81796-Abstract-Conference.html.

[19] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024. URL https://arxiv.org/abs/2409.12122.

[20] Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous RLHF: Faster and more efficient off-policy RL for language models. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=FhTAG591Ve.

[21] Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=X9diEuva9R.

[22] GLM-5 Team. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

[23] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1407–1416, 2018. URL https://proceedings.mlr.press/v80/espeholt18a.html.

[24] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

[25] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=5PAF7PAY2Y.

[26] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

[27] MiniMax. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025. URL https://arxiv.org/abs/2506.13585.

[28] Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025. URL https://arxiv.org/abs/2510.11370.

[29] Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=c1bTcrDmt4.

[30] Model Context Protocol. Model context protocol specification (version 2026-07-28). https://modelcontextprotocol.io/specification/2026-07-28, 2026.

[31] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. doi: 10.1109/IROS.2017.8202133.

[32] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1737–1749, 2023. doi: 10.1109/ICSE48619.2023.00149.

[33] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys)*, pages 674–688, 2024. doi: 10.1145/3627703.3629553.

[34] The PostgreSQL Global Development Group. PostgreSQL documentation: Routine vacuuming. https://www.postgresql.org/docs/current/routine-vacuuming.html, 2026. Accessed 2026-10-13.

[35] The Kubernetes Authors. Safely drain a node. https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/, 2026. Accessed 2026-10-13.

[36] The PostgreSQL Global Development Group. PostgreSQL documentation: Replication slots. https://www.postgresql.org/docs/current/warm-standby.html#STREAMING-REPLICATION-SLOTS, 2026. Accessed 2026-10-13.

[37] Ben Cohen, Emaad Khwaja, Youssef Doubli, Salahidine Lemaachi, Chris Lettieri, Charles Masson, Hugo Miccinilli, Elise Ramé, Qiqi Ren, Afshin Rostamizadeh, Jean Ogier du Terrail, Anna-Monica Toon, Kan Wang, Stephan Xie, Zongzhe Xu, Viktoriya Zhukova, David Asker, Ameet Talwalkar, and Othmane Abou-Amal. This time is different: An observability perspective on time series foundation models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2505.14766.

[38] Emaad Khwaja, Chris Lettieri, Gerald Woo, Eden Belouadah, Marc Cenac, Guillaume Jarry, Enguerrand Paquin, Xunyi Zhao, Viktoriya Zhukova, Othmane Abou-Amal, Chenghao Liu, Ameet Talwalkar, and David Asker. Toto 2.0: Time series forecasting enters the scaling era. Technical Report arXiv:2605.20119, Datadog AI Research, 2026. URL https://arxiv.org/abs/2605.20119.

[39] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pages 611–626, 2023. doi: 10.1145/3600006.3613165.

[40] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=VqkAKQibpq.

[41] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. CheckFreq: Frequent, fine-grained DNN checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST)*, pages 203–216, 2021. URL https://www.usenix.org/conference/fast21/presentation/mohan.

[42] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. GEMINI: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pages 364–381, 2023. doi: 10.1145/3600006.3613145.

[43] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing billion-scale model training. In *USENIX Annual Technical Conference (USENIX ATC)*, 2021. URL https://www.usenix.org/conference/atc21/presentation/ren-jie.

[44] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. URL https://arxiv.org/abs/1604.06174.

[45] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. URL https://arxiv.org/abs/1909.08053.

[46] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2020. doi: 10.1109/SC41405.2020.00024.

[47] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1701.06538.

[48] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

© Datadog 2026

Appendix

A RL Infrastructure Enhancements

In this appendix, we document some of what we learned and the engineering enhancements we made in order to scale up our SkyRL-based training stack to production training runs.

A.1 Balancing Generation and Training

Keeping the trainer fed is a balancing act. Generate too slowly and the training GPUs sit idle waiting for data; generate too fast and the queue fills, generation blocks, and the inference GPUs sit idle instead. The balance point also moves during training: as rollout length changes, investigations hold generation workers for longer or shorter periods, and the bottleneck moves with it (Figure 12).

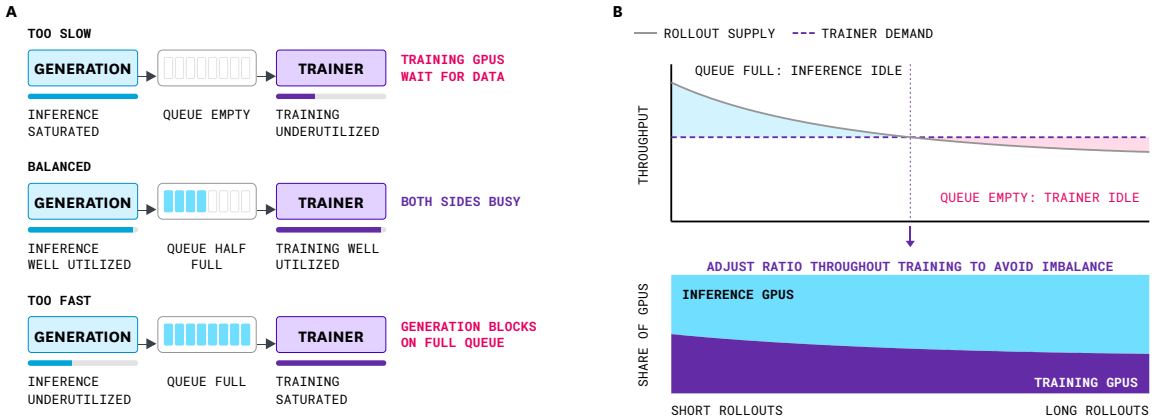

Figure 12 We balance generation against training to keep both sets of GPUs busy. A. Too much capacity on one side starves the other side's workload. **B.** The balance point moves during training as rollouts become longer or shorter.

Evaluation also runs continuously and asynchronously alongside generation and training. Instead of pausing training for a separate evaluation phase, we evaluate selected policy versions in the background while new rollouts and model updates continue (Figure 13). This gives us frequent quality signals without creating stop-the-world evaluation windows. However, evaluation also consumes inference, tool, and judging capacity, so we account for that load without allowing it to starve training rollouts.

We track generation time, tool call execution time, judge time, queue depth, and data age to find the limiting stage. Those measurements separate a shortage of generation capacity from slow telemetry queries or delayed judging, and we scale generation workers up or down to match.

Caching adds another constraint. Multi-turn investigations repeat long conversation prefixes, so inference caching can reuse that computation [39, 40], but only while rollout concurrency stays inside the KV cache capacity of the inference engines. Exceed it and the engines thrash on evictions, which slows generation and increases staleness.

This is why inference and training run in separate compute pools, with local query work scaled independently of model computation (Figure 14). CPU-only workers handle simulation, investigation coordination, checkpoint archival, and monitoring; GPU workers serve inference and training. Each pool is sized for the work it does, and resized as the bottleneck moves.

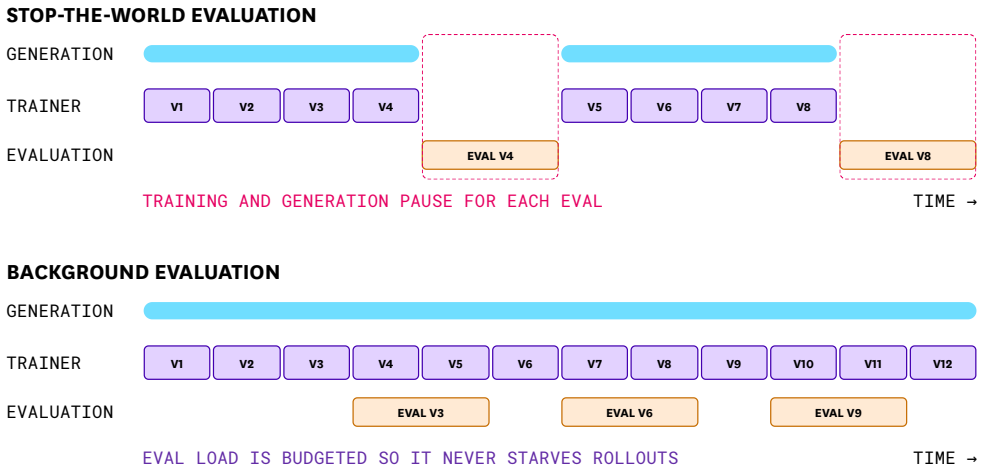

Figure 13 Evaluation runs alongside generation and training. Background evaluation removes the stop-the-world evaluation windows. Its load is part of the compute budget, so it does not starve training rollouts.

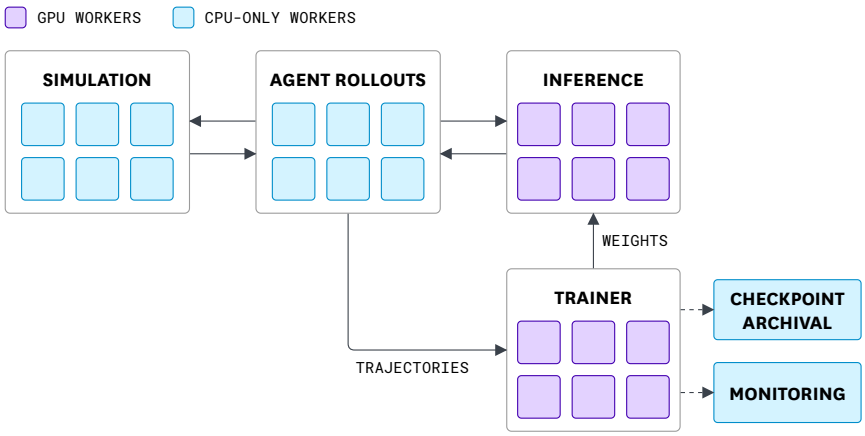

Figure 14 Heterogeneous compute lets worker pools scale independently on GPU or CPU-only hardware. CPU workers handle simulation, investigation coordination, checkpoint archival, and monitoring. GPU workers serve model inference and training.

A.2 Making Experiments Observable and Resumable

Reliable experiments require visibility into the complete investigation path. We instrument our framework with Datadog APM and Agent Observability to connect generation, tool calls, judge decisions, rewards, and the optimizer step for each rollout. Traces, logs, and GPU Monitoring help us to optimize our workloads and to determine whether a low reward came from poor agent behavior or an infrastructure failure. This differentiation is critical for computing a high-signal, low-noise reward.

In order to improve the resiliency of our pipeline, we also put significant effort into reducing the cost of starting and resuming RL runs. We initialize the generator and trainer concurrently (Figure 15) because they run on separate GPU pools and can prepare independently. While the generator produces the first trajectories, the trainer runs synthetic forward and backward passes across representative sequence-length buckets. This allows the trainer to warm up its compilation caches before training begins, preventing the first real batches from paying the cost of kernel JIT compilation.

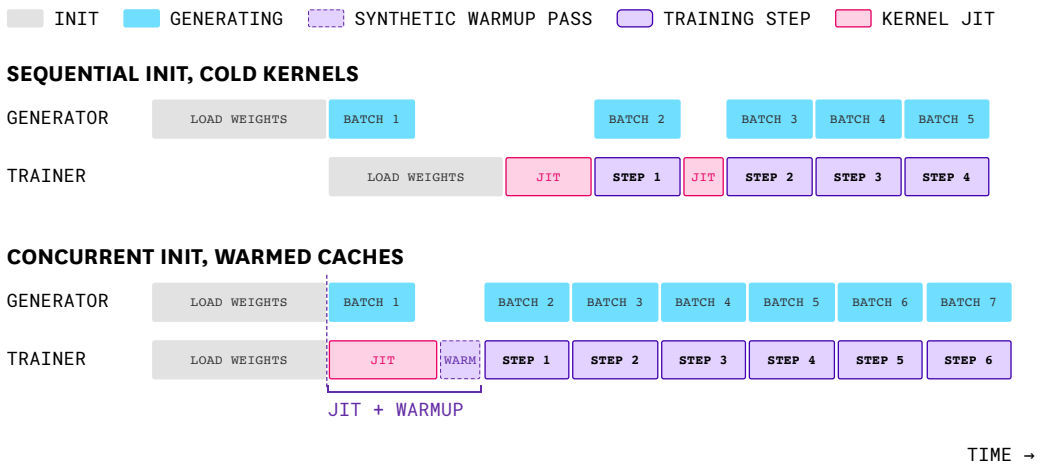

Figure 15 The trainer warms up while the generator produces its first trajectories. With concurrent initialization, the trainer compiles its kernels on synthetic passes, so the first real batches do not pay the JIT cost.

To prevent startups from creating a burst of requests, we introduce a gradual generation ramp whenever the rollout queue starts empty (Figure 16). Without this, every generation worker would start at once, creating a thundering herd of inference and tool requests. This can lead to significant KV cache eviction thrashing. The ramp gradually increases the number of generation groups allowed to run concurrently as trajectories complete, smoothing demand before reaching full concurrency. By smoothing out the generation workload, we significantly reduce the number of policy staleness violations that occur.

Checkpointing can be another major bottleneck. In some of our experiments, a single checkpoint could exceed 10 TB, and transferring that much state to durable storage outside the training cluster could block training for hours. To mitigate this, we adopt a tiered, asynchronous checkpointing approach that separates local persistence from remote archival, related to prior work on overlapped and in-memory checkpointing [41, 42]. Each rank writes its shard to cluster-local storage; once a complete local recovery point is available, training continues while the system archives the shards to durable object storage in the background.

A.3 Working Within System Constraints

Large-scale post-training is constrained by both GPU high-bandwidth memory (HBM) and host RAM. Keeping activations and optimizer state on the GPU is fastest, but long sequences can exceed available HBM. Offloading data to CPU makes larger workloads fit, but it consumes host memory and adds transfer overhead between the CPU and GPU. At sufficient scale, offloading can merely shift the bottleneck to a different part

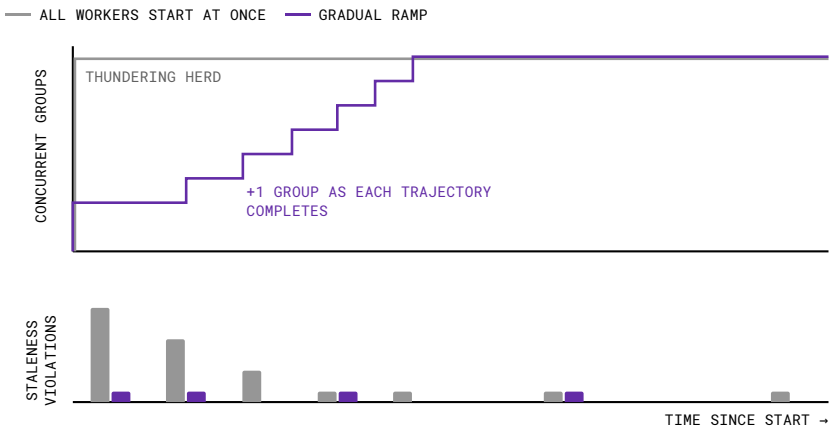

Figure 16 A gradual ramp avoids a thundering herd and cuts staleness violations. Concurrency grows by one group as each trajectory completes. This avoids the KV cache evictions that slow generation and increase staleness.

of the system, rather than removing it. Batch size, optimizer offloading [43], activation checkpointing [44], and parallelism strategies [45, 46] are among the dials we tune together, and since we train across different clusters and architectures, the balance point shifts with the HBM, host memory, and interconnect available.

Rollout Routing Replay, or R3 [28], creates another memory tradeoff. In a mixture-of-experts model, each token routes through only a subset of the model's experts [47, 48]. The inference and training systems can choose different experts for the same token, creating a mismatch between the policy that generated a trajectory and the policy update that learns from it. R3 records the expert choices made during generation and replays them during training. In some experiments, this reduced the average rollout-to-training log-probability mismatch by 30%.

However, these routing records can be large because they contain expert choices for every token and mixture-of-experts layer. A naive implementation expands them into nested Python objects and copies them into every training worker, creating a CPU-memory and serialization bottleneck. To keep memory use manageable in our production implementation, we transport the records in a custom binary wire protocol, storing expert IDs as compact `uint8` arrays. We also shard the router indices, sharing one route chunk per data-parallel rank across its model-parallel workers. Workers then transfer only the active microbatch to the GPU. This preserves the benefit of R3 without allowing its metadata to dominate host memory or training time.

B Bits Investigation Interface

Figure 17 shows the conclusion produced by Scout for the investigation presented in the case study (Section 6) as it appears in the Bits Investigation interface.

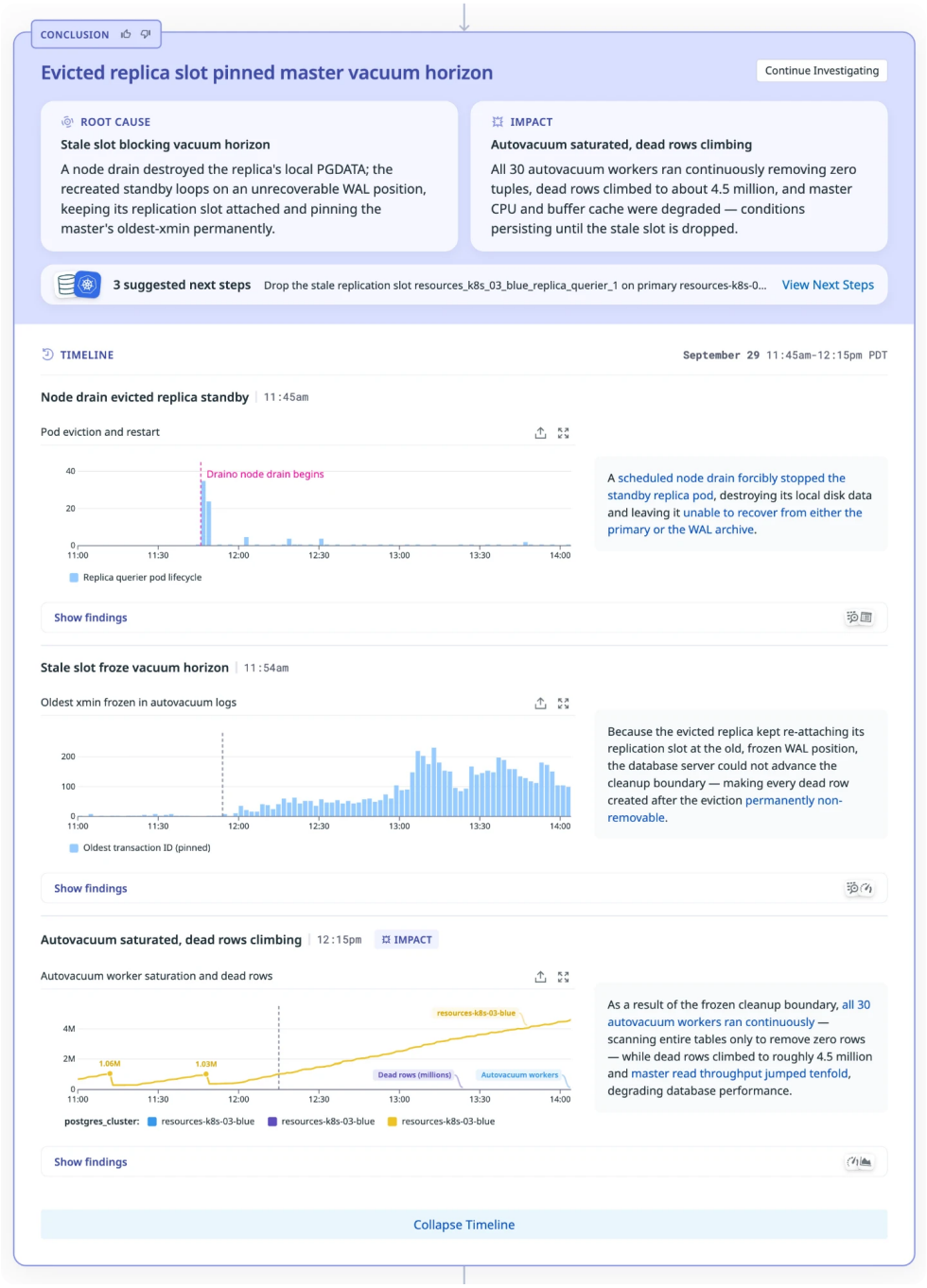

Figure 17 Scout's conclusion in the Bits Investigation interface. Scout produced this investigation for the autovacuum incident of Section 6.