STRATUS: A Multi-agent System for Autonomous Reliability Engineering of Modern Clouds

Yinfang Chen[◇*] **Jiaqi Pan**[◇¶*] **Jackson Clark**[◇*] **Yiming Su**[◇*]
Noah Zheutlin[†] **Bhavya Bhavya**[†] **Rohan Arora**[†] **Yu Deng**[†]
Saurabh Jha[†] **Tianyin Xu**[◇]
University of Illinois Urbana-Champaign[◇] IBM Research[†] IIIS, Tsinghua University[¶]

Abstract

In cloud-scale systems, failures are the norm. A distributed computing cluster exhibits hundreds of machine failures and thousands of disk failures; software bugs and misconfigurations are reported to be more frequent. The demand for *autonomous*, AI-driven reliability engineering continues to grow, as existing human-in-the-loop practices can hardly keep up with the scale of modern clouds. This paper presents STRATUS, an LLM-based multi-agent system for realizing autonomous Site Reliability Engineering (SRE) of cloud services. STRATUS consists of multiple specialized agents (e.g., for failure detection, diagnosis, mitigation), organized in a state machine to assist system-level safety reasoning and enforcement. We formalize a key safety specification of agentic SRE systems like STRATUS, termed *Transactional No-Regression (TNR)*, which enables safe exploration and iteration. We show that TNR can effectively improve autonomous failure mitigation. STRATUS significantly outperforms state-of-the-art SRE agents in terms of success rate of failure mitigation problems in AIOpsLab and ITBench (two SRE benchmark suites), by at least 1.5 times across various models. STRATUS shows a promising path toward practical deployment of agentic systems for cloud reliability.

1 Introduction

Cloud systems are the backbone of today's large-scale computing and end-user applications. Their reliability is crucial, with a few minutes of outages leading to significant user impacts and financial losses [9, 11]. Given the ever-growing scale, complexity, and dynamics of cloud systems, cloud reliability continues to be a grand challenge. Despite extensive efforts [32, 39, 58, 82], modern clouds still struggle to detect, localize, and mitigate system failures effectively and timely. One essential reason is the existing human-centric reliability engineering practice [12, 13]—human engineers are in the loop, responsible for decision making, e.g., which alerts to act upon, which tools to use, and
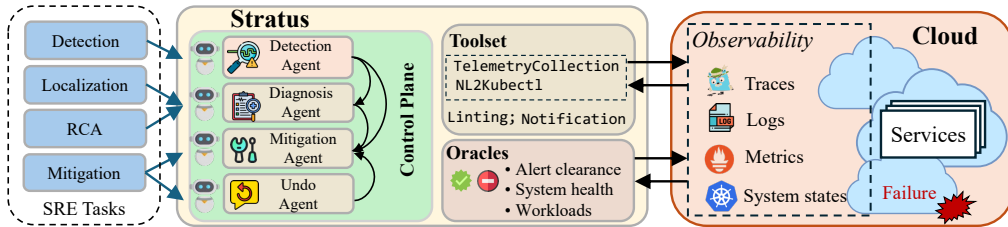

Figure 1: Overview of STRATUS, an LLM-based multi-agent system for autonomous Site Reliability Engineering (SRE) of modern cloud services.

---

[*]Co-primary authors.

which mitigation paths to choose. However, as failures are the norm in cloud-scale systems, it is increasingly costly for human-in-the-loop approaches to keep up with the scale of modern clouds. Autonomous Site Reliability Engineering (SRE), empowered by generative AI, is desired [19, 35, 62].

This paper explores an agentic AI approach for *autonomous* Site Reliability Engineering (SRE) of modern clouds. We present STRATUS, an LLM-based agentic system that realizes autonomous SRE for cloud services through failure detection, localization, root-cause analysis, and mitigation. STRATUS differs from prior work on developing AI or agentic tools that assist human engineers [20, 36, 37, 61, 73, 81] (for data collection, summarization, and root-cause prediction). STRATUS aims to govern live production systems and mitigate failures, without human intervention (see §2).

STRATUS is a multi-agent system that orchestrates specialized agents of detection, diagnosis, mitigation, etc., each responsible for one or multiple tasks (Figure 1). We chose a multi-agent design because (1) it specializes agents for SRE tasks with well-defined roles, (2) it offers customizability of individual agents and extensibility of the system, and (3) it makes it easy to reason about agent safety. STRATUS uses deterministic control-flow logic that orchestrates agents in a state machine, while leveraging LLMs in data flows that make agents intelligent and creative.

Safety is a key challenge of SRE agents (or any agentic systems that operate critical live systems). STRATUS must not worsen the state of the target system, when mitigating its failures; however, it is hard to ensure that a mitigation plan generated by the agent (or even by a human) is always successful, considering the dynamics and complexity of the system and imperfect AI models. In practice, the risk of "making things worse" is a fundamental barrier to deploying autonomous agentic techniques in high-stake systems. We formalize a safety specification of agentic SRE, termed *Transactional No-Regression* or TNR. TNR constructs transaction semantics to ensure: (1) the agent's mitigation actions, if unsuccessful, can always be "undone," and (2) the agent keeps improving the system in terms of its health state, by undoing actions that worsen it (i.e., no regression). We show that TNR enables safe agent exploration and iteration, which effectively improves failure mitigation.

We describe our implementation that realizes TNR using (1) state-machine based scheduling (for write exclusivity), (2) sandboxing (for confinement), and (3) an *Undo Agent* that realizes a system-wide undo operator [15]. We implement a stack-based undo mechanism that tracks the agent actions with regards to specific system state and reverts them in the correct order if needed. We also describe other notable implementations, e.g., agent tooling, bootstrapping, and termination.

We evaluate STRATUS on two SRE benchmark suites, AIOpsLab [19] and ITBench [35]. STRATUS significantly outperforms state-of-the-art agentic solutions in terms of success rate of failure mitigation problems in AIOpsLab and ITBench, by at least 1.5 times across various models (GPT-4o, GPT-4o-mini, and Llama3). We show that TNR, which enables the undo-and-retry practice, is key to the ability of STRATUS in solving complex mitigation problems; it allows STRATUS to safely and iteratively explore new mitigation paths, while avoiding unrecoverable error states.

Summary. This paper makes the following contributions:

- An attempt of an agentic AI system that autonomously mitigates system failures, towards fully autonomous Site Reliability Engineering of modern clouds.
- STRATUS, a multi-agent system that implements an end-to-end agentic SRE pipeline, from failure detection, localization, to mitigation, with system-wide safety properties.
- Transactional Non-Regression (TNR), a formalization of agentic SRE safety specification, and its implementation in STRATUS that enables safe agent exploration and iteration.
- STRATUS significantly outperforms state-of-the-art SRE agents in terms of success rate of various SRE tasks on two SRE benchmark suites (AIOpsLab and ITBench).
- The artifacts of STRATUS can be found at `https://github.com/xlab-uiuc/stratus`.

2 Background and Problem Definition

STRATUS aims to realize *autonomous SRE* which we define as an automated system on detecting failures, localizing failing components, analyzing the root causes, and mitigating the failures, with minimal human intervention. In other words, autonomous SRE systems like STRATUS imitate human engineers with agentic AI, and hence is fundamentally different from AI or agentic tools designed for assisting human engineers on specific SRE tasks [20, 73, 81]. A key differential feature of STRATUS

is to automatically *mitigate* failures, beyond summarizing failure information and predicting root cause categories as in prior work. Mitigation poses unique challenges for autonomous SRE: unlike detection and diagnosis tasks that only need to observe the system, mitigation needs to change the system state. How to ensure safety of failure mitigation is a key challenge—mitigation actions must not cause new (nested) failures and must not drive the system to a worse state (i.e., no regression).

We model a target cloud system as an *environment* $\mathcal{E}$ characterized by:

- A set of states $S = \{s\}$, including a crash state $\perp$ where the system is completely unavailable;
- A severity metric $\mu(s^e) : S \cup \{\perp\} \to \mathbb{N} \cup \{\infty\}$ that represents the severity of an error state s^e.

An error state can be caused by different types of faults (root causes) [10], including software bugs [27], misconfigurations [75], and hardware issues [29, 44, 46, 47]. The severity of the error for a given state s is measured as a weighted sum of the sets of alerts (A), violations of service-level agreement (SLA) (V), and system capacity loss (unhealthy nodes) (L); it is formally defined as: $\mu(s) = w_1 \cdot |A| + w_2 \cdot |V| + w_3 \cdot |L|$ where $w_i > 0, \mu(\perp) = \infty$. If the entire system is unavailable, then it is in the state $\perp$, where $\mu(\perp) = \infty$. Just like human SRE engineers [12], STRATUS addresses following SRE tasks in order to resolve the error in the system state:

- **Detection.** SRE must promptly detect production failures via logs, traces, and other telemetry data; detecting failures is the first step to prevent incidents.
- **Localization.** SRE must localize the faulty components so that they can be isolated to minimize the blast radius and impacts.
- **Root Cause Analysis (RCA).** SRE shall identify the root causes of the failures in order to repair the faulty components (e.g., fixing bugs and misconfigurations).
- **Mitigation.** SRE must mitigate the failures to prevent their propagations that lead to production incidents and service outages.

Note that RCA is not on the critical path of mitigation. In practice, RCA is typically done offline, while mitigation directly determines service availability [69]. A successful mitigation may not need to know the failure root causes. For example, mitigating a faulty machine can be done by migrating deployed jobs to a healthy machine [70], without knowing the faulty component; mitigating a faulty software component can be done by rebooting it [17, 56] or reverting recent changes [14].

3 STRATUS Design

STRATUS uses a multi-agent system design that orchestrates multiple specialized agents $\mathcal{A} = \{\alpha_i\}$ (α_i refers to an agent) to interact with $\mathcal{E}$ and mitigate any system failure represented by an error state s^e. Our implementation currently uses four agents $\mathcal{A} = \{\alpha_D, \alpha_G, \alpha_M, \alpha_U\}$:

- α_D (**Detection agent**): Observes $\mathcal{E}$ to identify failures and establishes the initial error state s_0^e.
- α_G (**Diagnosis agent**): Observes $\mathcal{E}$ (s_0^e and telemetry) to determine the root cause(s) of the detected failure; its output is primarily analytical. α_G is responsible for both localization and RCA.
- α_M (**Mitigation agent**): Takes the diagnostic output from α_D and α_G. It is responsible for (1) devising a high-level mitigation plan, (2) decomposing the plan into a sequence of concrete mitigation actions, and (3) executing each action by issuing commands through agent tools with Agent-Computer Interfaces (ACI) [77] (see §4).
- α_U (**Undo Agent**): Executes an undo sequence on $\mathcal{E}$ if a transaction executed by α_M is aborted.

Agents interact with the environment using actions from a defined *Action Space* through ACI:

- A_{read}: Read-only commands that do not change the state of $\mathcal{E}$ (e.g., kubectl get[2]).
- A_{write}: Write commands that can mutate the state of $\mathcal{E}$ (e.g., kubectl apply).
- A_{undo}: A specialized sequence of commands executed by α_U to undo the effects of write commands.

Agents α_D, α_G are restricted to A_{read}. Agent α_M executes actions from A_{write}, as well as A_{read} for reading system states and checking pre/post conditions. Agent α_U executes actions effectively in A_{undo} (which internally uses A_{write} commands to restore a prior state).

---

[2]kubectl is the command-line tool of Kubernetes, a *de facto* cloud platform that manages cloud systems [16].

The system state of $\mathcal{E}$ changes based on *State-Transition Relations* (see [26]), $R : S \times A_{\text{write}} \to S \cup \{\bot\}$. The undo is performed by an *operator* $U : S \to S$, realized by α_U.

3.1 Safety Specification

STRATUS must guarantee safety. However, an agent's action, whether due to an imperfect understanding, a planning flaw, or hallucinations of the generative model, could inadvertently worsen an already degraded system state ($\mu(s_0^e) > 0$). In our experience, this risk of "making things worse" is a main barrier to deploying autonomous agentic techniques in high-stake systems. How can we grant autonomous agents the authority to execute sequences of potentially state-changing commands (e.g., via `kubectl`) for critical tasks like failure mitigation, without risking worse outcomes?

We formalize a safety specialization of agentic SRE systems like STRATUS, termed Transactional Non-Regression (TNR), atop the classic transition abstraction [28]. TNR ensures that (1) the agent's mitigation actions, if unsuccessful, can always be "undone", and (2) the agent keeps improving the system state (i.e., the severity metric decreases monotonically, with no regression).

3.1.1 Assumptions

We state the following assumptions, which are enforced by our implementation (§4):

A1 **Writer Exclusivity (Agent-Lock or A-Lock).** At most one writer agent (α_M or α_U) is scheduled to execute commands that can mutate system states at a time and have exclusive access; reader agents (α_D, α_G) do not mutate state relevant to μ. A-Lock is a readers-writer lock.

A2 **Faithful Undo.** The undo operator U, when invoked by α_U on s_{post} after an abort decision, restores the checkpointed s_{pre} exactly, i.e., $U(s_{\text{post}}) = s_{\text{pre}}$.

A3 **Bounded Risk Window.** The length k (the number of commands) of any transaction executed by α_M is bounded by a system-wide threshold K ($1 \leq k \leq K$). This limits the duration the A-Lock is held and the complexity of any single transaction.

3.1.2 Transaction Semantics

We construct transaction semantics atop the above assumptions, where a transaction of length k ($1 \leq k \leq K$) is a sequence of read or write commands $T = (a_1, \ldots, a_k) \in (A_{\text{write}}, A_{\text{read}})^k$. A transaction encodes a mitigation plan executed by α_M while holding the A-Lock (see A1):

R1 **Checkpoint**: α_M records the entry state s_{pre} before a_1 is executed.

R2 **Execute**: α_M runs $a_1 \ldots a_k$ sequentially. Let s_{post} be the state after a_k completes (or $\bot$ if any a_j ($1 \leq j \leq k$) causes a crash).

R3 **Commit/Abort rule**: α_M evaluates: *Commit* if $s_{\text{post}} \neq \bot$ and $\mu(s_{\text{post}}) \leq \mu(s_{\text{pre}})$; otherwise, *abort* by instructing α_U to invoke U *once*. Per A2, this restores s_{pre} exactly.

An aborted transaction leaves *no trace* on externally visible state transitions. We call the internal sequence of states visited during transaction execution, $s_{\text{pre}} \xrightarrow{a_1} s^{(1)} \xrightarrow{a_2} \ldots \xrightarrow{a_k} s_{\text{post}}$, and their corresponding severity metrics the *hidden μ-path*. Only $\mu(s_{\text{pre}})$ and, if T commits, $\mu(s_{\text{post}})$ are part of the externally *visible* state transitions that define the system's trajectory viewed by other agents.

3.1.3 Transactional Non-Regression (TNR)

Let s_0^e be the error state observed by α_D when an failure occurs, with an *initial severity* $b = \mu(s_0^e)$.

Lemma 3.1. *Every state s in externally visible state transitions (i.e., $s = s_0^e$, or s is a state immediately following a read-only action by α_D or α_G, or s is a state following the completion (commit or abort) of a transaction by α_M or α_U) satisfies $\mu(s) \leq b$.*

Proof Sketch. The proof is by induction on the sequence of externally visible states, where an externally visible state is one observed outside of any active transaction T.

Base Case. For the initial state s_0^e (established by α_D when the failure is detected), $\mu(s_0^e) = b$ by definition. Thus, the invariant $I(s) \equiv \mu(s) \leq b$ holds for s_0^e.

4

Inductive Step. Assume that the invariant $I(s_i) \equiv \mu(s_i) \leq b$ holds for an externally visible state s_i. We show it holds for the next externally visible state s_{i+1}.

- If s_{i+1} results from a *read* action (by α_D or α_G) on s_i: These agents execute actions from A_{read} which do not mutate state. Thus, $s_{i+1} = s_i$, and $\mu(s_{i+1}) = \mu(s_i) \leq b$. The invariant holds.
- If s_{i+1} results from the completion of T executed by α_M, starting from s_i (so $s_{\text{pre}} = s_i$):
  - The T's execution $(a_1, \ldots, a_k)$ and its commit/abort decision occur atomically with respect to other writer agents due to *A-Lock (A1)*.
  - If the T *commits*, the new externally visible state is $s_{i+1} = s_{\text{post}}$. The commit condition requires $\mu(s_{\text{post}}) \leq \mu(s_{\text{pre}})$. Since $s_{\text{pre}} = s_i$ and $\mu(s_i) \leq b$ by the inductive hypothesis, it follows that $\mu(s_{\text{post}}) \leq b$. The invariant holds for s_{i+1}.
  - If the T *aborts*, α_U restores s_{pre} with *Faithful Undo (A2)*. The externally visible state is $s_{i+1} = s_{\text{pre}}$ (which is s_i). Since $\mu(s_i) \leq b$ by the inductive hypothesis, the invariant holds for s_{i+1}. Note that the effects of $a_1, \ldots, a_k$ by α_M are not part of the externally visible sequence.

Thus, any state in the externally visible sequence satisfies $\mu(s) \leq b$. $\qquad\square$

Table 1 gives an example that contrasts the *hidden* and *visible* severity trajectories for four common mitigation plans, as orchestrated by STRATUS.

Table 1: The TNR safety guarantee under four failure mitigation plans.

Mitigation	TNR actions (by α_M)	Hidden μ-path[3]	Commit?	Visible μ
Node drain/rebalance	cordon, evict, scale	$12 \to 18 \to 9$	✓	$12 \to 9$
Rolling upgrade	scale 0, patch, scale 3	$15 \to 22 \to 11$	✓	$15 \to 11$
Bad image attempted	scale 0, patch(bad), scale 3	$15 \to 24 \to 30$	✗	$15 \to 15$
Single hot-fix ($K{=}1$)	apply hotfix	$15 \to x$	✓ if $x \leq 15$ ✗ otherwise	≤ 15

3.2 Implications

TNR, which states that "severity never increases over the initial baseline b in the observable trace of states," is an instance of the *Alpern-Schneider safety property* [6, 8]. This safety property is not merely a theoretical construct; it forms a foundation for building trustworthy and effective AI agents that interact with critical live systems. For any agentic system employing an ACI to enact changes, especially those using LLMs for complex reasoning and planning (like STRATUS's α_M), the risk of generating sub-optimal or flawed action sequences is non-trivial.

By ensuring that the potential "damage" of an agent is capped (it cannot make the system observably worse than s_0), TNR provides a powerful tool for *safe exploration and iteration*. This allows the agent to attempt ambitious, complex repairs, learn from outcomes (via observing μ post-transaction), and adapt its strategy without the risk of digging a deeper hole.

Moreover, preventing agent-driven escalations beyond the baseline b makes the *liveness* property of eventually reaching a healthy state tenable [7, 8]. It empowers robust, reliable AI agents capable of sophisticated, autonomous interventions in the real world, fostering the development of AI systems that can not only act, but also act *safely* through *recoverability*.

3.3 Extension

TNR currently assumes no concurrent writer agents and has no concurrency control. Writer agents are strictly serialized with the readers-writer lock (A1). Our current deployment model uses one STRATUS instance per cloud system (each system managed in a separate namespace).

STRATUS can be extended to support multiple concurrent writers by evolving A-Lock into a fine-grained, resource-aware concurrency control system. The idea is to allow multiple mitigation agents to execute transactions concurrently, provided their intended actions do not conflict over shared resources. This can be achieved through a *Coordination Controller* that functions as a distributed lock manager. Each mitigation agent first comes up with a plan P_i, which is associated with a set

---

[3]Hidden μ values arise *inside* the transaction during execution by α_M and are not visible to other agents, nor do they form part of the externally visible state sequence against which the lemma is proven.

of resources, $R(P_i)$, containing the unique identifiers of all system components its write actions will mutate. The controller maintains a global set of locked resources, R_{locked}, which is the union of resource sets for all currently executing plans, $R_{\text{locked}} = \bigcup_{j \in \text{executing}} R(P_j)$. A newly submitted plan P_i is admitted for execution if and only if its resource set is disjoint from the global lock set: $\text{Admit}(P_i) \iff R(P_i) \cap R_{\text{locked}} = \emptyset$.

Upon admission, the plan's resources are atomically added to the global lock set ($R_{\text{locked}} \leftarrow R_{\text{locked}} \cup R(P_i)$), effectively acquiring a distributed lock on them. The agent then executes its transaction under the original TNR semantics, guaranteeing that the severity metric does not increase: $\mu(s_{\text{post}}) \leq \mu(s_{\text{pre}})$. Once the transaction completes (via commit or abort), its resources are released: $R_{\text{locked}} \leftarrow R_{\text{locked}} \setminus R(P_i)$. This mechanism upholds the "Writer Exclusivity" (A1) at the resource level, ensuring that the safety properties of TNR hold for each concurrent transaction. Furthermore, this strict resource isolation can prevent read-after-write hazards, as an agent is guaranteed a consistent view of any resource not explicitly locked by another transaction.

4 Implementation

STRATUS is implemented using the CrewAI multi-agent framework [1]. The agents are orchestrated via state-machine based control-flow logic (Figure 2). STRATUS is an extensible framework. The agents are loosely coupled and can be independently developed and customized. We currently use off-the-shelf LLMs such as GPT and Llama models without fune-tuning. One can also add new agents or replace existing agents. The control plane can also be customized (e.g., using a different state machine) as it is decoupled from the data plane of the agents (each agent's data-flow logic).
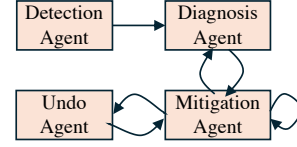
Figure 2: The state machine based control-flow logic.

4.1 Realizing TNR in STRATUS

In STRATUS, the assumptions A1–A3 are enforced as safety invariants with the following mechanisms.

Writer Exclusivity. First, sandboxing-based confinement is implemented for each agent. The detection and diagnosis agents are not allowed to issue any commands that may mutate system states (the confinement rules are detailed in the appendix). Second, the state machine ensures mutual exclusion when a writer agent (the mitigation agent or the undo agent) is scheduled to run, i.e., the sequences of the actions of the writer agents are strictly serialized. We also instruct the agent to dry-run the commands (`kubectl --dry-run`) whenever applicable to simulate command execution and identify errors without altering the system state.

Faithful Undo. STRATUS ensures that every agent action has a corresponding undo operator; otherwise, the action is not allowed. STRATUS rejects destructive actions which cannot be recovered, or turns them into recoverable actions through the agent tools (e.g., a file deletion is replaced by moving the file to a backup volume, which is recoverable). A common pattern of the undo operator is system-state rollback. Modern cloud platforms such as Kubernetes [16], Borg [71], Twine [68], and ECS [45] are implemented based on *the state-reconciliation principle* [65, 66], with declarative interfaces [26, 67]. Rollback in state-reconciliation systems can be implemented by recording the state changes and later reconciling to the recorded states.

STRATUS implemented a stack-based rollback mechanism, as exemplified by Figure 3. The stack-based mechanism allows fine-grained rollback of the related system resources (represented as state objects in Kubernetes) instead of the system-wide state. Our current undo agent α_U mechanically walks over the stack and performs undo, and thus is arguably not intelligent. The rationale of keeping α_U as an independent agent is to integrate more advanced undo policies and mechanisms. For example, we are exploring a learning-based rollback policy where the LLMs are given the choice of undoing a subset of the actions in the stack (instead of always rolling back to the start state).
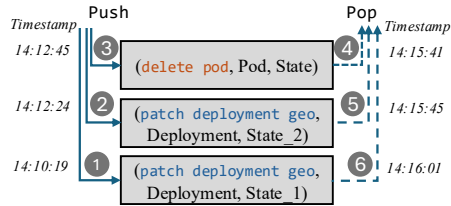
Figure 3: An example of the action stack used for reconciliation-based undo.

Bounded Risk Window. The implementation is trivial. However, the window size has an impact on mitigation effectiveness. We set K to 20 based on an empirical analysis (in appendix).

Limitations. From an implementation perspective, realizing perfect undo for all conceivable state changes in complex environments like cloud systems remains a practical challenge (e.g., involving application-specific states and external interactions). Certain operations may require more sophisticated compensation logic not covered by a simple $U(s_{\text{post}}) = s_{\text{pre}}$. Fortunately, modern cloud-native systems are typically equipped with an operator program that follows the state-reconciliation principle [21, 57], which can be leveraged by STRATUS. Moreover, our rule-based confinement may not be perfect to capture all destructive actions. Recent work on advanced guardrails can help improve the confinement (see Related Work in §6).

The principle of segmenting agent interventions into transactionally bounded stages, each adhering to non-regression with respect to a measurable system health metric (μ), significantly de-risks autonomous operation. It transforms the problem from hoping an entire complex plan is perfect to verifying the safety of smaller, manageable steps.

4.2 Other Notable Implementations

Agent tools. We develop tools that enable agents to interact with the environment using natural languages, following the Agent-Computer Interface (ACI) principles [77]. Specifically, we developed two sets of tools: (1) *observability tools* for agents to query different kinds of observability data such as state objects (maintained by the cloud platform like Kubernetes), system logs, distributed traces, performance metrics, etc; We use LLMs to summarize the observability data instead of directly feeding a large volume of data to the agents, as suggested by recent work [20]. (2) *command-line tools* that allow agents to execute commands and change system states using natural language, e.g., `NL2Kubectl` takes descriptions in natural language and generates concrete `kubectl` commands. The confinement described in §4.1 is implemented as the command-line tool wrappers. STRATUS's design supports hybrid operations where writer agents can request human approval before execution, with interceptable action stacks and tool interfaces.

Bootstrapping (Where shall the diagnosis agent start?) The multi-agent design makes it easy to specialize agents with different roles. We boost the diagnosis agent α_G with a bootstrapping technique that helps α_G localize the failure region. In cloud systems, failures are propagated along request-response paths [40, 69], which is reflected by distributed traces [38, 64]. STRATUS localizes the faults by constructing the call graph and establishes an initial fault localization hypothesis to guide the diagnostic process. Despite the simple idea, the fault-localization based bootstrapping is important for large-scale systems and massive volumes of observability data.

Termination (When shall the agents stop?) Agents need to correctly determine whether the target failure is successfully resolved to stop its actions; a perpetual agent could cause unintended effects, destabilize the system, or waste computing resources. STRATUS integrates a structured validation-and-termination approach. After each diagnosis or mitigation procedure (a series of actions), STRATUS assesses system health based on three kinds of oracles: (1) *alerts*: whether the alert that reports the target failure is cleared, (2) *user requests*: whether user requests can be successfully returned (e.g., no `5XX` or `4XX` HTTP responses), and (3) *system health*: whether system components (e.g., pods and volumes) are running in healthy states. These oracles act as weak oracles individually, each capturing a partial view of system health. STRATUS combines multiple oracles to form a stronger oracle. If all oracles pass, α_M concludes a successful mitigation and terminates.

5 Evaluation

We evaluate STRATUS on two state-of-the-art benchmarks, AIOpsLab [19] and ITBench [35]. Both benchmarks provide a live, arena-like environment, where AI agents are asked to resolve problems; each problem encodes a failure in emulated cloud systems [3, 25, 84], as exemplified by Figure 4. We focus on STRATUS's ability to mitigate the failures (which has dependencies with detection and localization); we also report results of the other tasks, including detection, localization, and RCA. Each benchmark takes STRATUS 15–20 machine hours to finish.

Figure 4: An example problem.

Baseline Agents. We compare STRATUS with the reference agents offered by AIOpsLab and ITBench, referred to as AOL-
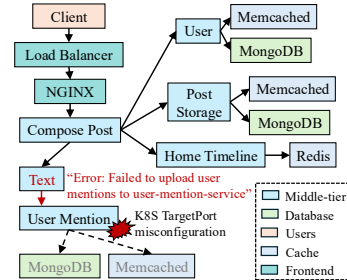
agent and ITB-agent respectively. For AIOpsLab, we also compare STRATUS with two other agents, ReAct and Flash, included in the benchmark [19]. We fuel these agents with different LLMs, including GPT-4o, GPT-4o-mini, and Llama 3.3. The LLM temperature is set to 0 to make the results more deterministic.

We measure agentic systems using metrics: Success Rate (the percentage of problems being successfully solved), average time (in seconds), the number of steps to solve a problem, and monetary cost in US dollars (calculated based on token consumptions). We also profile the memory footprint for one heavy problem across different agents in STRATUS.

Failure Mitigation Effectiveness. As shown in Table 2, STRATUS (GPT-4o) significantly outperforms state-of-the-art SRE agents on mitigation problems in both AIOpsLab and ITBench. Concretely, it successfully solves 69.2% (9/13) and 50.0% (9/18) mitigation problems in AIOpsLab and ITBench, respectively. This leads to improvements on the success rate by 1.5X and 5.4X over the second best performing solution (AOL-agent and ITB-agent), respectively. The advantage of STRATUS over the other SRE agents is consistent with different models, including both GPT-4o-mini and Llama 3.3.

We carefully inspect the problem-solving trajectories of the evaluated agents. For the mitigation problems solved by STRATUS in AIOpsLab, STRATUS correctly mitigated the triggering conditions or the root causes of the failures. In eight ITBench problems, STRATUS exploits an observation that the injected faults do not persist, after the failing pods restarted (the fault injector cannot recognize the original pod); thus, STRATUS tends to restart the failure pods one by one and thus solves the problem. Such a mitigation strategy may not work in reality, especially for failures caused by persistent faults (e.g., misconfigurations and hardware defects).

Table 2: **Effectiveness of STRATUS in solving mitigation problems of AIOpsLab [19] and ITBench [35].** ITB-agent numbers are taken from [35] ("-": unknown). The "$" column reports the average cost (in USD) of running the agent over all the mitigation tasks in each benchmark.

<table>
<tr><td colspan="5">(a) AIOpsLab (13 Mitigation Problems)</td><td colspan="5">(b) ITBench (18 Mitigation Problems)</td></tr>
<tr><td>Agent</td><td>Succ.</td><td>Time (s)</td><td>Steps</td><td>$</td><td>Agent</td><td>Succ.</td><td>Time (s)</td><td>Steps</td><td>$</td></tr>
<tr><td>ReAct (4o)</td><td>23.1%</td><td>46.0</td><td>23.0</td><td>0.112</td><td>ITB-agent (4o)</td><td>9.2%</td><td>251.7</td><td>-</td><td>-</td></tr>
<tr><td>Flash (4o)</td><td>38.5%</td><td>154.0</td><td>23.1</td><td>0.150</td><td>ITB-agent (llama)</td><td>5.7%</td><td>440.8</td><td>-</td><td>-</td></tr>
<tr><td>AOL-agent (4o)</td><td>46.2%</td><td>223.3</td><td>21.7</td><td>0.206</td><td>STRATUS (4o)</td><td>50.0%</td><td>1720.8</td><td>115.7</td><td>6.11</td></tr>
<tr><td>AOL-agent (mini)</td><td>7.7%</td><td>58.9</td><td>22.7</td><td>0.003</td><td>STRATUS (mini)</td><td>19.4%</td><td>3874.9</td><td>468.9</td><td>9.38</td></tr>
<tr><td>AOL-agent (llama)</td><td>15.4%</td><td>98.2</td><td>13.0</td><td>0.037</td><td>STRATUS (llama)</td><td>28.0%</td><td>2566.6</td><td>160.3</td><td>0.76</td></tr>
<tr><td>STRATUS (4o)</td><td>69.2%</td><td>811.9</td><td>46.3</td><td>0.877</td><td colspan="5">• "4o" refers to GPT-4o (gpt-4o-2024-08-06)</td></tr>
<tr><td>STRATUS (mini)</td><td>23.1%</td><td>3557.9</td><td>125.7</td><td>0.036</td><td colspan="5">• "mini" refers to GPT-4o-mini (gpt-4o-mini-2024-07-18)</td></tr>
<tr><td>STRATUS (llama)</td><td>23.1%</td><td>1486.9</td><td>71.8</td><td>0.360</td><td colspan="5">• "llama" refers to Llama 3.3 (llama-3-3-70b-instruct)</td></tr>
</table>

STRATUS overall takes longer time to solve problems; for the hard problems, STRATUS retries multiple times to solve them—*the retry is enabled by the Transactional No-Regression safety guarantee.* In comparison, the other agents do not have the safe retry capability. In fact, they provide no safety guarantee over their mitigation actions—for the failures they fail to mitigate, these agents can leave the system in worse states than the original error state. For the same reason, STRATUS spends more in dollar amount, but the amount is significantly cheaper than human cost.

Effectiveness of TNR-based Undo-and-Retry. The TNR safety is crucial to STRATUS's effectiveness in solving complex mitigation problems. Table 3 shows this point through an ablation study of STRATUS (GPT-4o) (more detailed ablation analysis can be found in the appendix). We compare STRATUS (GPT-4o) with two variants: (1) *No retry:* Only attempting one mitigation path, and (2) *Naïve retry w/o undo:* exploring a new mitigation path if the previous mitigation path was not successful; however, the new exploration is conducted directly from the ending state of the last attempt, instead of rolling back to the original state as in STRATUS.

We see that (1) the ability of retrying a different mitigation plan is important—only in two problems, the first mitigation plan generated by STRATUS successfully mitigates the failure, and (2) the ability to undo and rollback the system state is critical—new mitigation from an error state left by a failed attempt can hardly succeed (only one new problem is solved in this way). Intuitively, failed mitigations further worsen the system states, making the system harder and harder to save. TNR prevents such

failure patterns and thus is essential to mitigation. Figure 5 shows the probability density function of the number of retries (per problem) done by the STRATUS (GPT-4o) in AIOpsLab mitigation problems. STRATUS sets a limit of nine retries. In 80+% of the problems, STRATUS retries at least once; in 30+% of the problems, STRATUS retries no less than five times.

Table 3: **Ablation analysis on TNR-enabled undo-and-retry of STRATUS (GPT-4o)** using the 13 mitigation problems in AIOpsLab, which shows the importance of TNR safety.

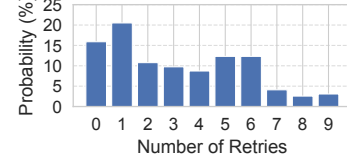| Ablation	Succ. Rate	Time (s)	Cost ($)
STRATUS (4o)	69.2%	811.9	0.877
— No retry	15.4%	72.6	0.163
— Naïve retry w/o undo	23.1%	1221.5	0.929

Figure 5: Probability density of the retry times per problem.

Effectiveness on Detection, Localization, and Root Cause Analysis (RCA). We also evaluated the effectiveness of STRATUS on other SRE problems, in comparison to other SRE agents. Table 4 shows the results for problems in AIOpsLab. Note that the number of problems in the categories are different. STRATUS outperforms the reference agents in both detection and localization problems. Specifically, STRATUS with Llama and GPT-4o achieves over 90% of success rate for detection. Localization and RCA problems, which require the agents to pinpoint the root-cause components (e.g., the pod) and the fault type (e.g., misconfiguration or bug), present greater challenges to agentic solutions; STRATUS (GPT-4o) has the highest success rate for localization problems and is second highest for RCA problems. Note that RCA is not a prerequisite of failure mitigation (see §2).

Table 4: **Effectiveness of STRATUS in solving other SRE problems in AIOpsLab.** The problems in AIOpsLab are independent; there are in total 86 problems of four SRE types (§2).

Agent	Detection (32 Problems)			Localization (28 Problems)			RCA (26 Problems)		
	Succ.	Time (s)	$	Succ.	Time (s)	$	Succ.	Time (s)	$
ReAct (4o)	87.5%	33.2	0.086	26.8%	59.6	0.328	23.1%	28.5	0.065
Flash (4o)	59.4%	30.7	0.013	39.3%	165.0	0.190	26.9%	30.5	0.019
AOL-agent (4o)	62.5%	14.4	0.061	46.9%	34.8	0.083	38.5%	12.3	0.061
AOL-agent (mini)	25.0%	43.0	0.002	9.5%	34.1	0.001	7.7%	57.7	0.003
AOL-agent (llama)	84.4%	19.8	0.019	32.1%	40.5	0.018	30.8%	13.8	0.014
STRATUS (4o)	90.6%	48.4	0.118	51.2%	65.3	0.126	34.6%	39.6	0.068
STRATUS (mini)	78.1%	34.4	0.010	25.0%	37.2	0.013	30.8%	279.0	0.007
STRATUS (llama)	93.8%	50.0	0.111	36.3%	90.5	0.112	26.9%	60.2	0.095

Note that the relatively low RCA success rate across other tasks for STRATUS (and all the other agents) is largely attributed to the ambiguous evaluation metrics used by AIOpsLab. AIOpsLab measures RCA success by matching the agent's output against pre-defined category labels; however, some labels are ambiguous and not mutually exclusive. For example, for a port misconfiguration problem, AIOpsLab defines the root cause to be "Virtualization" (as the layer) and "Misconfiguration" (as the fault type). STRATUS answers the root cause to be "Application" (as the layer) and "Network/Storage Issue" (as the fault type). Arguably, STRATUS's answer is more precise in describing the root cause—a port is more of an application-level issue than "Virtualization", since there is no traditional hypervisor-like virtualization layer in AIOpsLab. However, STRATUS is considered incorrect for this problem in AIOpsLab due to such label mismatch. As ongoing work, we are actively investigating new metrics designs to improve the benchmarks for SRE agents.

Memory usage. To understand the memory usage of STRATUS, we profile STRATUS running a heavy mitigation problem (`misconfig_app_hotel_res-mitigation-1` in AIOpsLab), which takes 13 steps to resolve. The dominant memory cost (over 530 MB) comes from loading the agent runtime and its dependencies (e.g., CrewAI and benchmark tooling). The Detection agent uses the most memory as it collects and analyzes raw telemetry data (e.g., using `pandas` for trace analysis). The Mitigation agent consumes the largest context memory (31.03 KB) as it accumulates trajectories across steps, whereas the Undo agent uses the least (5.79 KB) due to its simple rollback operations.

6 Related Work

AI and Agentic Research for SRE. Applying AI/ML techniques to solve SRE problems has a long history of research, as AI/ML has the potential to accommodate the large scale of modern systems and infrastructures (e.g., the clouds) [48, 84], as well as massive volumes of observability data [38]. Specialized AI/ML techniques are designed for concrete SRE tasks, such as failure detection [52, 54, 79], triage [59, 60], and diagnosis [24, 33, 42, 49]. Recently, generative AI models, as well as agentic approaches, are increasingly used to further generalize and automate these SRE tasks [5, 20, 36, 78]. However, the aforementioned efforts focus on building tools to assist human engineering (see §2). Differently, we aim for autonomous SRE. A differential feature is *failure mitigation*. Prior work mostly focuses on helping human engineers to detect, understand the failure or recommend potential mitigation documents, where mitigation is the main goal of STRATUS

Safety of Agentic AI Systems. Safety is an emerging concern of agentic AI systems [22, 31, 50, 63]. Prior work focuses on implementing safety guardrails [2, 34, 51, 72], which aim to prevent agents from producing or acting upon harmful inputs and outputs, often through prompt-level or static analysis. However, such preventative approaches are limited in environments like cloud systems, where the system state evolves over time and side effects emerge dynamically only after actions are executed. For example, AutoSafeCoder [51] mitigates code vulnerabilities via static checks, but such methods cannot anticipate runtime failures or cascading effects that depend on temporal system dynamics. In addition, few works provide guarantees on recovery. For example, TrustAgent [31] proposes the Agent-Constitution framework that address agent safety. However, it does not provide guarantees on the safety of the agent action or any recovery from unsafe actions. For SRE, this could allow failures to regress into a worse situation due to unsafe agent actions.

STRATUS addresses this challenge by enabling safe exploration with Transactional Non-Regression (TNR). TNR allows agents to explore multiple action sequences safely and iteratively, giving the agent more opportunities to mitigate failures. Concurrent work has also started integrating transactional primitives into LLM-based frameworks [18]. STRATUS extends this direction by enabling rollback of both the agent and the system state, combined with a continuous severity metric to assess action outcomes. This dynamic and fine-grained safety mechanism is critical for real-world deployment of autonomous agents in evolving operational contexts like SRE.

Multi-agent Systems. Multi-agent has become a common design pattern, with the development of many LLM-based multi-agent frameworks [1, 4, 30, 41, 74] and use cases [55, 76, 83, 85]. Recent work proposes multi-agent conversations [74] and debates [23, 43] that encourage diverse thinking among agents. However, our experience shows that conversations and debates are not suitable for SRE tasks which require safety reasoning and timeliness of problem solving. Hence, STRATUS uses a deterministic state machine to coordinate the agents. Recent work also reports common failure modes of multi-agent systems [53, 80]. We detail how STRATUS addresses them in the appendix.

7 Concluding Remarks

With generative AI and agentic technologies, autonomous SRE for modern clouds may no longer be a pipedream. While we can expect new AI models and fine-tuning to further improve problem-solving abilities of our agents, we assert that safety is the key challenge for agentic SRE systems to operate live production clouds. We also assert that agent safety has to be rigorously defined and enforced as a first-class system design principle. This paper makes a first attempt to formalize TNR as a useful safety specification of agentic SRE and realize it in our multi-agent SRE system. We envision and hope the work to inspire stronger safety properties and system-level support for agentic SRE systems.

Acknowledgment

References

[1] Crew AI. `https://www.crewai.com/`.

[2] Guardrails AI. `https://www.guardrailsai.com/`.

[3] OpenTelemetry Astronomy Shop Demo. `https://opentelemetry.io/docs/demo/`.

[4] Swarm: Scalable infrastructure for multi-agent coordination. `https://github.com/openai/swarm`, 2024.

[5] AHMED, T., GHOSH, S., BANSAL, C., ZIMMERMANN, T., ZHANG, X., AND RAJMOHAN, S. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)* (Jul 2023).

[6] ALPERN, B., DEMERS, A. J., AND SCHNEIDER, F. B. Safety without stuttering. *Information Processing Letters 23*, 4 (1986), 177–180.

[7] ALPERN, B., AND SCHNEIDER, F. B. Defining Liveness. *Information Processing Letters 21* (1985), 181–185.

[8] ALPERN, B., AND SCHNEIDER, F. B. Recognizing safety and liveness. *Distributed Computing 2*, 3 (Sept. 1987), 117–126.

[9] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the Clouds: A Berkeley View of Cloud Computing. Tech. rep., University of California at Berkeley, Feb. 2009.

[10] AVIZIENIS, A., LAPRIE, J.-C., AND RANDELL, B. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing (TDSC) 1*, 1 (Jan 2004), 1–23.

[11] BARROSO, L. A., HÖLZLE, U., AND RANGANATHAN, P. *The Datacenter as a Computer: Designing Warehouse-Scale Machines*. 2018.

[12] BEYER, B., JONES, C., PETOFF, J., AND MURPHY, N. R. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Inc., Mar. 2016.

[13] BEYER, B., MURPHY, N. R., RENSIN, D. K., KAWAHARA, K., AND THORNE, S. *Site Reliability Workbook: Practical Ways to Implement SRE*. O'Reilly Media Inc., Aug. 2018.

[14] BHAGWAN, R., KUMAR, R., MADDILA, C. S., AND PHILIP, A. A. Orca: Differential Bug Localization in Large-Scale Services. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)* (Oct. 2018).

[15] BROWN, A. B., AND PATTERSON, D. A. Undo for Operators: Building an Undoable E-mail Store. In *Proceedings of the 2003 USENIX Annual Technical Conference (ATC'03)* (June 2003).

[16] BURNS, B., GRANT, B., OPPENHEIMER, D., BREWER, E., AND WILKES, J. Borg, Omega, and Kubernetes. *Communications of the ACM 59*, 5 (May 2016), 50–57.

[17] CANDEA, G., KAWAMOTO, S., FUJIKI, Y., FRIEDMAN, G., AND FOX, A. Microreboot – A Technique for Cheap Recovery. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation (OSDI'04)* (Dec. 2004).

[18] CHANG, E. Y., AND GENG, L. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning. *arXiv:2503.11951* (Mar 2025).

[19] CHEN, Y., SHETTY, M., SOMASHEKAR, G., MA, M., SIMMHAN, Y., MACE, J., BANSAL, C., WANG, R., AND RAJMOHAN, S. AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds. In *Proceedings of the Conference on Machine Learning and Systems (MLSys'25)* (May 2025).

[20] CHEN, Y., XIE, H., MA, M., KANG, Y., GAO, X., SHI, L., CAO, Y., GAO, X., FAN, H., WEN, M., ZENG, J., GHOSH, S., ZHANG, X., ZHANG, C., LIN, Q., RAJMOHAN, S., ZHANG, D., AND XU, T. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the 19th European Conference on Computer Systems (EuroSys'24)* (Apr. 2024).

[21] DOBIES, J., AND WOOD, J. *Kubernetes Operators: Automating the Container Orchestration Platform.* O'Reilly Media, Inc., Feb. 2020.

[22] DONG, Y., MU, R., ZHANG, Y., SUN, S., ZHANG, T., WU, C., JIN, G., QI, Y., HU, J., MENG, J., BENSALEM, S., AND HUANG, X. Safeguarding Large Language Models: A Survey. *arXiv:2406.02622* (Jun 2024).

[23] DU, Y., LI, S., TORRALBA, A., TENENBAUM, J. B., AND MORDATCH, I. Improving Factuality and Reasoning in Language Models through Multiagent Debate. *arXiv:2305.14325* (May 2023).

[24] GAN, Y., LIANG, M., DEV, S., LO, D., AND DELIMITROU, C. Sage: Practical & Scalable ML-Driven Performance Debugging in Microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)* (Apr 2021).

[25] GAN, Y., ZHANG, Y., CHENG, D., SHETTY, A., RATHI, P., KATARKI, N., BRUNO, A., HU, J., RITCHKEN, B., JACKSON, B., HU, K., PANCHOLI, M., HE, Y., CLANCY, B., COLEN, C., WEN, F., LEUNG, C., WANG, S., ZARUVINSKY, L., ESPINOSA, M., LIN, R., LIU, Z., PADILLA, J., AND DELIMITROU, C. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)* (Apr 2019).

[26] GU, J. T., SUN, X., ZHANG, W., JIANG, Y., WANG, C., VAZIRI, M., LEGUNSEN, O., AND XU, T. Acto: Automatic End-to-End Testing for Operation Correctness of Cloud System Management. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP'23)* (Oct 2023).

[27] GUNAWI, H. S., HAO, M., LEESATAPORNWONGSA, T., PATANA-ANAKE, T., DO, T., ADITYATAMA, J., ELIAZAR, K. J., LAKSONO, A., LUKMAN, J. F., MARTIN, V., AND SATRIA, A. D. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC'14)* (Nov. 2014).

[28] HAERDER, T., AND REUTER, A. Principles of transaction-oriented database recovery. *ACM Computing Surveys 15*, 4 (Dec. 1983), 287–317.

[29] HOCHSCHILD, P. H., TURNER, P., MOGUL, J. C., GOVINDARAJU, R., RANGANATHAN, P., CULLER, D. E., AND VAHDAT, A. Cores that don't count. In *Proceedings of the 18th Workshop on Hot Topics in Operating Systems (HotOS'21)* (June 2021).

[30] HONG, S., ZHUGE, M., CHEN, J., ZHENG, X., CHENG, Y., WANG, J., ZHANG, C., WANG, Z., YAU, S. K. S., LIN, Z., ZHOU, L., RAN, C., XIAO, L., WU, C., AND SCHMIDHUBER, J. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *Proceedings of the International Conference on Machine Learning (ICML'24)* (Jan 2024).

[31] HUA, W., YANG, X., LI, Z., WEI, C., AND ZHANG, Y. TrustAgent: Towards Safe and Trustworthy LLM-based Agents through Agent Constitution. *arXiv:2402.01586* (Feb. 2024).

[32] HUANG, P., GUO, C., LORCH, J. R., ZHOU, L., AND DANG, Y. Capturing and Enhancing In Situ System Observability for Failure Detection. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)* (Oct. 2018).

[33] IKRAM, A., CHAKRABORTY, S., MITRA, S., SAINI, S., BAGCHI, S., AND KOCAOGLU, M. Root Cause Analysis of Failures in Microservices through Causal Discovery. In *Proceedings of The Thirty-Fifth Annual Conference on Neural Information Processing Systems (NeurIPS'22)* (Oct. 2022).

[34] INAN, H., UPASANI, K., CHI, J., RUNGTA, R., IYER, K., MAO, Y., TONTCHEV, M., HU, Q., FULLER, B., TESTUGGINE, D., AND KHABSA, M. Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations. *arXiv:2312.06674* (Dec 2023).

[35] JHA, S., ARORA, R., WATANABE, Y., YANAGAWA, T., CHEN, Y., CLARK, J., BHAVYA, B., VERMA, M., KUMAR, H., KITAHARA, H., ZHEUTLIN, N., TAKANO, S., PATHAK, D., GEORGE, F., WU, X., TURKKAN, B. O., VANLOO, G., NIDD, M., DAI, T., CHATTERJEE, O., GUPTA, P., SAMANTA, S., AGGARWAL, P., LEE, R., MURALI, P., WOOK AHN, J., KAR, D., RAHANE, A., FONSECA, C., PARADKAR, A., DENG, Y., MOOGI, P., MOHAPATRA, P., ABE, N., NARAYANASWAMI, C., XU, T., VARSHNEY, L. R., MAHINDRU, R., SAILER,

A., SHWARTZ, L., SOW, D., FULLER, N. C. M., AND PURI, R. ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks. In *Proceedings of the International Conference on Machine Learning (ICML'25)* (July 2025).

[36] JIANG, Y., ZHANG, C., HE, S., YANG, Z., MA, M., QIN, S., KANG, Y., DANG, Y., RAJMOHAN, S., LIN, Q., AND ZHANG, D. Xpert: Empowering Incident Management with Query Recommendations via Large Language Models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE'24)* (Apr. 2024).

[37] JIN, P., ZHANG, S., MA, M., LI, H., KANG, Y., LI, L., LIU, Y., QIAO, B., ZHANG, C., ZHAO, P., HE, S., SARRO, F., DANG, Y., RAJMOHAN, S., LIN, Q., AND ZHANG, D. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2023).

[38] KALDOR, J., MACE, J., BEJDA, M., GAO, E., KUROPATWA, W., O'NEILL, J., ONG, K. W., SCHALLER, B., SHAN, P., VISCOMI, B., VENKATARAMAN, V., VEERARAGHAVAN, K., AND SONG, Y. J. Canopy: An End-to-End Performance Tracing And Analysis System. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)* (Oct. 2017).

[39] LENERS, J. B., WU, H., HUNG, W.-L., AGUILERA, M. K., AND WALFISH, M. Detecting Failures in Distributed Systems with the Falcon Spy Network. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)* (Oct. 2011).

[40] LI, A., LU, S., NATH, S., PADHYE, R., AND SEKAR, V. ExChain: Exception Dependency Analysis for Root Cause Diagnosis. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)* (Apr. 2024).

[41] LI, G., HAMMOUD, H., ITANI, H., KHIZBULLIN, D., AND GHANEM, B. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Proceedings of The Thirty-Sixth Annual Conference on Neural Information Processing Systems (NeurIPS'23)* (Sept. 2023).

[42] LI, M., LI, Z., YIN, K., NIE, X., ZHANG, W., SUI, K., AND PEI, D. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. In *Proceedings of the 28th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'22)* (Aug. 2022).

[43] LIANG, T., HE, Z., JIAO, W., WANG, X., WANG, Y., WANG, R., YANG, Y., SHI, S., AND TU, Z. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP'24)* (Nov. 2024).

[44] MANEAS, S., MAHDAVIANI, K., EMAMI, T., AND SCHROEDER, B. A Study of SSD Reliability in Large Scale Enterprise Storage Deployments. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)* (Feb. 2020).

[45] MELISSARIS, T., NABAR, K., RADUT, R., REHMTULLA, S., SHI, A., CHANDRASHEKAR, S., AND PAPAPANAGIOTOU, I. Elastic Cloud Services: Scaling Snowflake's Control Plane. In *Proceedings of the 13th ACM Symposium on Cloud Computing (SOCC'22)* (Nov. 2022).

[46] MEZA, J., WU, Q., KUMAR, S., AND MUTLU, O. A Large-Scale Study of Flash Memory Failures in the Field. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'15)* (June 2015).

[47] MEZA, J., XU, T., VEERARAGHAVAN, K., AND SONG, Y. J. A Large-Scale Study of Data Center Network Reliability. In *Proceedings of the 18th ACM Internet Measurement Conference (IMC'18)* (Oct. 2018).

[48] MIAO, G., MOSER, L. E., YAN, X., TAO, S., CHEN, Y., AND ANEROUSIS, N. Generative Models for Ticket Resolution in Expert Networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)* (July 2010).

[49] MICKENS, J., SZUMMER, M., AND NARAYANAN, D. Snitch: Interactive Decision Trees for Troubleshooting Misconfigurations. In *Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SYSML'07)* (Apr. 2007).

[50] MIEHLING, E., RAMAMURTHY, K. N., VARSHNEY, K. R., RIEMER, M., BOUNEFFOUF, D., RICHARDS, J. T., DHURANDHAR, A., DALY, E. M., HIND, M., SATTIGERI, P.,

WEI, D., RAWAT, A., GAJCIN, J., AND GEYER, W. Agentic AI Needs a Systems Theory. *arXiv:2503.00237* (Feb. 2025).

[51] NUNEZ, A., ISLAM, N. T., JHA, S. K., AND NAJAFIRAD, P. AutoSafeCoder: A Multi-Agent Framework for Securing LLM Code Generation through Static Analysis and Fuzz Testing. *arXiv:2409.10737* (Sept. 2024).

[52] PALATIN, N., LEIZAROWITZ, A., SCHUSTER, A., AND WOLFF, R. Mining for Misconfigured Machines in Grid Systems. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)* (Aug. 2006).

[53] PAN, M. Z., CEMRI, M., AGRAWAL, L. A., YANG, S., CHOPRA, B., TIWARI, R., KEUTZER, K., PARAMESWARAN, A., RAMCHANDRAN, K., KLEIN, D., GONZALEZ, J. E., ZAHARIA, M., AND STOICA, I. Why Do Multiagent Systems Fail? In *Proceedings of The Thirteenth International Conference on Learning Representations (ICLR'25)* (Mar. 2025).

[54] POTHARAJU, R., CHAN, J., HU, L., NITA-ROTARU, C., WANG, M., ZHANG, L., AND JAIN, N. ConfSeer: Leveraging Customer Support Knowledge Bases for Automated Misconfiguration Detection. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'15)* (Aug. 2015).

[55] QIAN, C., LIU, W., LIU, H., CHEN, N., DANG, Y., LI, J., YANG, C., CHEN, W., SU, Y., CONG, X., XU, J., LI, D., LIU, Z., AND SUN, M. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL'24)* (Aug. 2024).

[56] QIN, F., TUCEK, J., SUNDARESAN, J., AND ZHOU, Y. Rx: Treating Bugs As Allergies—A Safe Method to Survive Software Failure. In *Proceedings of the 20th Symposium on Operating System Principles (SOSP'05)* (Oct. 2005).

[57] RATIS, P. Lessons Learned Using the Operator Pattern to Build a Kubernetes Platform. In *USENIX SREcon* (Oct. 2021).

[58] REN, X. J., WANG, S., JIN, Z., LION, D., CHIU, A., XU, T., AND YUAN, D. Relational Debugging — Pinpointing Root Causes of Performance Problems. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)* (July 2023).

[59] SHAO, Q., CHEN, Y., TAO, S., YAN, X., AND ANEROUSIS, N. EasyTicket: a Ticket Routing Recommendation Engine for Enterprise Problem Resolution. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB'08)* (Aug. 2008).

[60] SHAO, Q., CHEN, Y., TAO, S., YAN, X., AND ANEROUSIS, N. Efficient Ticket Routing by Resolution Sequence Mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)* (Aug. 2008).

[61] SHETTY, M., BANSAL, C., UPADHYAYULA, S. P., RADHAKRISHNA, A., AND GUPTA, A. AutoTSG: Learning and Synthesis for Incident Troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)* (Nov. 2022).

[62] SHETTY, M., CHEN, Y., SOMASHEKAR, G., MA, M., SIMMHAN, Y., ZHANG, X., MACE, J., VANDEVOORDE, D., LAS-CASAS, P., GUPTA, S. M., NATH, S., BANSAL, C., AND RAJMOHAN, S. Building AI Agents for Autonomous Clouds: Challenges and Design Principles. In *Proceedings of 15th ACM Symposium on Cloud Computing (SoCC'24)* (Nov. 2024).

[63] SHINN, N., CASSANO, F., BERMAN, E., GOPINATH, A., NARASIMHAN, K., AND YAO, S. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv:2303.11366* (Mar. 2023).

[64] SIGELMAN, B. H., BARROSO, L. A., BURROWS, M., STEPHENSON, P., PLAKAL, M., BEAVER, D., JASPAN, S., AND SHANBHAG, C. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. `http://research.google.com/archive/papers/dapper-2010-1.pdf`, Apr. 2010.

[65] SUN, X., LUO, W., GU, J. T., GANESAN, A., ALAGAPPAN, R., GASCH, M., SURESH, L., AND XU, T. Automatic Reliability Testing for Cluster Management Controllers. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)* (July 2022).

14

[66] SUN, X., SURESH, L., GANESAN, A., ALAGAPPAN, R., GASCH, M., TANG, L., AND XU, T. Reasoning About Modern Datacenter Infrastructures Using Partial Histories. In *Proceedings of the 18th Workshop on Hot Topics in Operating Systems (HotOS'21)* (May 2021).

[67] SURESH, L., AO LOFF, J., KALIM, F., JYOTHI, S. A., NARODYTSKA, N., RYZHYK, L., GAMAGE, S., OKI, B., JAIN, P., AND GASCH, M. Building Scalable and Flexible Cluster Managers Using Declarative Programming. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)* (Nov. 2020).

[68] TANG, C., YU, K., VEERARAGHAVAN, K., KALDOR, J., MICHELSON, S., KOOBURAT, T., ANBUDURAI, A., CLARK, M., GOGIA, K., CHENG, L., CHRISTENSEN, B., GARTRELL, A., KHUTORNENKO, M., KULKARNI, S., PAWLOWSKI, M., PELKONEN, T., RODRIGUES, A., TIBREWAL, R., VENKATESAN, V., AND ZHANG, P. Twine: A Unified Cluster Management System for Shared Infrastructure. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)* (Nov. 2020).

[69] TREYNOR, B., DAHLIN, M., RAU, V., AND BEYER, B. The Calculus of Service Availability. *Communications of the ACM* (Aug. 2017).

[70] VEERARAGHAVAN, K., MEZA, J., MICHELSON, S., PANNEERSELVAM, S., GYORI, A., CHOU, D., MARGULIS, S., OBENSHAIN, D., PADMANABHA, S., SHAH, A., SONG, Y. J., AND XU, T. Maelstrom: Mitigating Datacenter-level Disasters by Draining Interdependent Traffic Safely and Efficiently. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)* (Oct. 2018).

[71] VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-Scale Cluster Management at Google with Borg. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys'15)* (Apr. 2015).

[72] WANG, H., POSKITT, C. M., AND SUN, J. AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. *arXiv:2503.18666* (Mar. 2025).

[73] WANG, Z., LIU, Z., ZHANG, Y., ZHONG, A., FAN, L., WU, L., AND WEN, Q. RCAgent: Cloud Root Cause Analysis by Autonomous Agents with Tool-Augmented Large Language Models. *arXiv:2310.16340* (Oct. 2023).

[74] WU, Q., BANSAL, G., ZHANG, J., WU, Y., LI, B., ZHU, E., JIANG, L., ZHANG, X., ZHANG, S., LIU, J., AWADALLAH, A. H., WHITE, R. W., BURGER, D., AND WANG, C. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. In *Proceedings of the 1st Conference on Language Modeling (COLM'24)* (Oct. 2024).

[75] XU, T., ZHANG, J., HUANG, P., ZHENG, J., SHENG, T., YUAN, D., ZHOU, Y., AND PASUPATHY, S. Do Not Blame Users for Misconfigurations. In *Proceedings of the 24th Symposium on Operating System Principles (SOSP'13)* (Nov. 2013).

[76] XU, T., ZHANG, Y.-F., CHU, Z., WANG, S., AND WEN, Q. AI-Driven Virtual Teacher for Enhanced Educational Efficiency: Leveraging Large Pretrain Models for Autonomous Error Analysis and Correction. *arXiv:2409.09403* (Sept. 2024).

[77] YANG, J., JIMENEZ, C. E., WETTIG, A., LIERET, K., YAO, S., NARASIMHAN, K., AND PRESS, O. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Proceedings of the Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS'24)* (Sept. 2024).

[78] YU, Z., MA, M., ZHANG, C., QIN, S., KANG, Y., BANSAL, C., RAJMOHAN, S., DANG, Y., PEI, C., PEI, D., LIN, Q., AND ZHANG, D. MonitorAssistant: Simplifying Cloud Service Monitoring via Large Language Models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE'24)* (July 2024).

[79] ZHANG, J., RENGANARAYANA, L., ZHANG, X., GE, N., BALA, V., XU, T., AND ZHOU, Y. EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection. In *Proceedings of the 19th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'14)* (Mar. 2014).

[80] ZHANG, S., YIN, M., ZHANG, J., LIU, J., HAN, Z., ZHANG, J., LI, B., WANG, C., WANG, H., CHEN, Y., AND WU, Q. Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems. In *Proceedings of the International Conference on Machine Learning (ICML'25)* (2025).

[81] ZHANG, X., MITTAL, T., BANSAL, C., WANG, R., MA, M., REN, Z., HUANG, H., AND RAJMOHAN, S. FLASH: A Workflow Automation Agent for Diagnosing Recurring Incidents. `https://www.microsoft.com/en-us/research/publication/flash-a-workflow-automation-agent-for-diagnosing-recurring-incidents/`.

[82] ZHANG, Y., MAKAROV, S., REN, X., LION, D., AND YUAN, D. Pensieve: Non-Intrusive Failure Reproduction for Distributed Systems using the Event Chaining Approach. In *Proceedings of the 26th ACM Symposium on Operating System Principles (SOSP'17)* (Oct. 2017).

[83] ZHAO, J., ZU, C., HAO, X., LU, Y., HE, W., DING, Y., GUI, T., ZHANG, Q., AND HUANG, X. LONGAGENT: Achieving Question Answering for 128k-Token-Long Documents through Multi-Agent Collaboration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP'24)* (Nov. 2024).

[84] ZHOU, X., PENG, X., XIE, T., SUN, J., JI, C., LI, W., AND DING, D. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Transactions on Software Engineering 47* (Dec. 2021), 243–260.

[85] ZHOU, Z., LIN, Y., JIN, D., AND LI, Y. Large Language Model for Participatory Urban Planning. *arXiv:2402.17161* (Feb. 2024).

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: We list our main contributions in the end of the Introduction, and the contributions are analyzed in Section 3 and evaluated in Section 5.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Limitations are discussed in context (see Section 3.2 and Section 4).

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We state our assumptions in Section 3.1.1 and write down our proofs in Section 3.1.3.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4 discusses our implementation. The artifacts and instructions can be found at `https://github.com/xlab-uiuc/stratus`.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
    1. If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
    2. If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
    3. If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
    4. We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Section 4 discusses our implementation. The artifacts and instructions can be found at `https://github.com/xlab-uiuc/stratus`.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We explain our experimental setting and details in Section 5 and appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The reported results are statistically significant. The experiments are run multiple times and the median numbers are reported in the tables.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We document the compute resources used in the appendix. We list the cost of using LLMs in our evaluation section.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: All experiments are conducted on publicly available benchmarks. We do not anticipate any negative social impacts. Our agent is designed specifically for Site Reliability Engineering.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss broader societal impacts in the appendix.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: We do not release any data or models, so there is no potential for misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The foundation models used in our experiments are cited properly. Only open-source benchmark suites are used.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Our only new asset is our code, can be found at `https://github.com/xlab-uiuc/stratus`. We also provide scripts and a README file containing documentation in the repository.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: This paper does not involve crowdsourcing or research with any human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This paper does not involve crowdsourcing or research with any human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: STRATUS is fundamentally an LLM-based multi-agent system for autonomous Site Reliability Engineering (SRE) of cloud services. The LLMs are central to the system's planning, reasoning, and interaction with the environment through natural-language interfaces, making their usage both original and essential to the system's methodology.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendix

In the appendix, we provide additional analysis, detailed results, and more discussions.

Table of Contents

A Confinement Rules

To ensure safety and predictability, each STRATUS agent operates under sandboxed confinement rules. As detailed in Table 5, detection and diagnosis agents are restricted to a Read-Only role and are only permitted to *observe* the system using commands such as `TelemetryCollection` tools. These agents are explicitly prohibited from executing any operation that alters the state of the system.

To perform mitigation, STRATUS will need to use the Writer role, as changing the system state is essential for mitigation. For state-changing tasks, STRATUS performs command-level validation to verify that generated actions are syntactically correct and safe to run in production. Risky operations such as deleting namespaces are prohibited.

Certain `kubectl` commands, such as `kubectl exec -it` and `kubectl edit`, launch interactive shells or editors. These commands are prevented because STRATUS cannot participate in interactive sessions, which would cause execution to hang. Additionally, granting arbitrary shell access significantly increases the potential for unintended and unsafe behaviors. Therefore, we explicitly disallow all interactive commands.

Table 5: Agent confinement rules in STRATUS. ▮ shows the allowed sandboxing rules according to the agent's role, while ▮ shows the blocked actions enforced by STRATUS.

Category	Subcategory	Confinement	Example(s)
Agent Role-Based	Read-Only Agents	Non-mutating commands only	`kubectl get`, `kubectl describe` `kubectl logs`, `ls`, `cat` `TelemetryCollection`
	Writer Agents	Sequential execution only	Only serialized state modifications are allowed
Command-Level Confinement (Blocked for all agents):			
Kubectl Subcommands	Destructive Operations	Namespace deletion	`kubectl delete namespace my-ns`
	Interactive Edit	debug, edit	`kubectl debug pod/foo` `kubectl edit deployment/bar`
	Input from stdin	`-f -` pattern	`kubectl apply -f -`
Kubectl Flags	Interactive Terminal	`-stdin`, `-tty`, `exec -it`	`kubectl attach -tty pod/foo` `kubectl exec -it pod/bash`
Shell Syntax	Pipelines	Pipe operator	`kubectl get pods | grep Running`
	Compound Commands	`&&`, `||`, `;`	`kubectl get pods && echo success`
	Command Substitution	`$(...)`, backticks	`kubectl delete pod $(kubectl get pods -o name)`
	Flow Control	if, for, while, until, case	`if [ ... ]; then ...; fi` `for pod in ...; do ...; done`
	Shell Functions	Function definitions	`myfunc() { ... }`

Confinement Examples

Example 1 (Input from stdin):

1. STRATUS:	`kubectl apply -f -`
2. Confinement:	`"Stdin redirection is not allowed."`

Example 2 (Interactive Terminal):

1. AOL-agent:	`kubectl exec -it mongodb-rate-bfbcf4587-md6xl -n` `test-hotel-reservation -- mongo`
2. AOL-agent:	Hang there until timeout.
1. STRATUS:	`kubectl exec -it mongodb-rate-bfbcf4587-j7lsf -n` `test-hotel-reservation -- mongo`
2. Confinement:	`"Interactive flag detected: -it.  Such commands are not supported."`

To further enforce safety, STRATUS uses static linting to validate command syntax before execution. This validation includes rejecting shell pipes, compound commands (e.g., using `&&`, `||`, or semicolons), and other complex shell constructs. These are disallowed for two reasons: (1) there are exponentially many arbitrary shell expressions that STRATUS may generate; comprehensively verifying the safety

of all possible shell expressions is not realistic. On the contrary, individual `kubectl` invocations can be easily checked. (2) we require the agent to operate incrementally, issuing and validating one command at a time, to work effectively with our safety infrastructure and confinement layers.

When supported, STRATUS employs Kubernetes' dry-run mechanism to produce the effect of a command without persisting changes. In this mode, the Kubernetes API server processes the request and returns the result without modifying the system. This simulated output is presented to the agent, which can then decide whether to proceed, adjust the command, or cancel the operation entirely. This approach allows the mitigation agent to preview the impact of state-changing actions and apply corrections before execution.

The examples above show the effectiveness of command-level confinement in the mitigation agent of STRATUS, particularly for preventing execution failures or system hangs due to unsafe operations. In Example 1, the agent attempts to execute the command `kubectl apply -f -`, which will make the agent hang due to missing input via stdin. However, it is prevented by STRATUS with feedback. In Example 2, the agent issues an interactive command using the `-it` flag, attempting to launch a terminal session within a running pod. Without confinement, this results in the execution hanging until a timeout occurs. STRATUS's confinement mechanism blocks such interactive terminal invocations, returning a clear message to the agent: `"Interactive flag detected: -it. Such commands are not supported."`

In our evaluation, 10.5% of commands executed by STRATUS (GPT-4o) and 9.7% by STRATUS (LLaMA3) would result in system hangs, in the absence of our confinement mechanisms.

B Bounded Risk Window

A key safety mechanism in STRATUS is the *bounded risk window*, which specifies the number of actions that the mitigation agent (α_M) can execute in a single transaction. In this section, we empirically investigate how the total step limit impacts STRATUS's success rate on AIOpsLab problems (detection, localization, root-cause analysis, and mitigation). For each problem, we run the evaluated agents with total step limits setting to different values (3, 5, 10, 15, 20, and 30). Results are presented in Figure 6.

STRATUS (GPT-4o) improves from 0% to 53.53%, after switching the step limit from 3 to 10. This shows that allowing STRATUS to take more steps can significantly boost its effectiveness. We also find that among all evaluated agents, agent performance plateaus after 15 steps, indicating that taking more steps after a certain threshold does not yield better performance significantly.

The effectiveness gain can be observed on STRATUS with other LLM backends, but with lower overall accuracy, relatively scaling with the LLM's capability. Based on these observations, we set the default bounded risk window to $K = 20$.
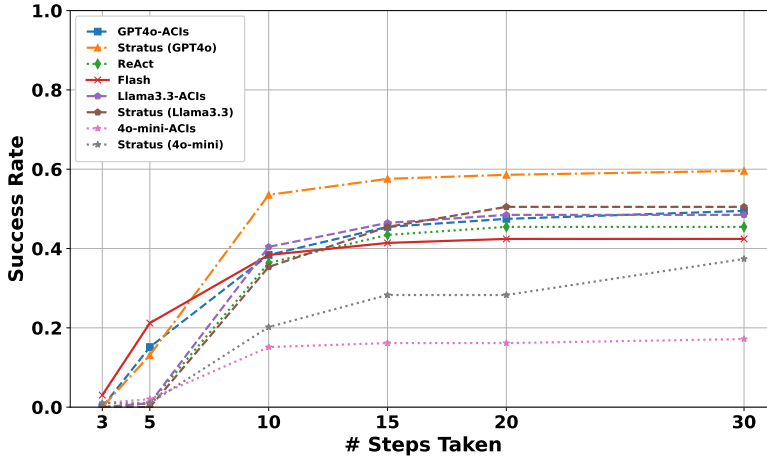

Figure 6: Impact of step limit on success rate of different agents in AIOpsLab.

C Ablation Study on ITBench

In §5, we presented an ablation study of STRATUS (GPT-4o) on AIOpsLab, demonstrating the effectiveness of its retry and undo mechanisms. We now examine STRATUS's behavior under ablation on the ITBench benchmark, as shown in Table 6. We observe that (1) disabling retry substantially drops the success rate to 11.1%. Note that this configuration implicitly disables the undo agent, as there are no prior actions to undo. Without retries, the agent must determine the root cause and devise a mitigation plan correctly with one attempt, which is significantly challenging. (2) STRATUS achieves the same task-level success rate (solving 9/18 problems) when the undo agent is disabled. As mentioned in §5, STRATUS exploits the observation that in 8 out of 18 problems, restarting the target pods clears the incident alerts. Since ITBench evaluates correctness primarily based on the absence of immediate alerts, this allows the agent to resolve the incident without trying more destructive, state-changing actions, where our undo mechanism will be most effective.

To better understand this, we analyzed these 8 problems in detail. Among all mitigation actions issued by the agent, approximately 15.2% are pod deletion operations managed by Kubernetes Deployment, which can automatically recreate the pods. Furthermore, a significant portion (44.2%) of state-changing commands, e.g., changing a container image to an image with a hallucinated name, are easily detected and fixed by STRATUS in the subsequent run, since STRATUS has a reflection mechanism, which can help the agent to realize what went wrong in the previous run. The rest of the commands are a mix of direct restart or redeployment of the pods, which again lifts the underlying fault directly. In this regard, the undo mechanism does not improve the success rate in ITBench. However, the absence of undo capabilities can still compromise the cloud over time. Latent or non-critical side effects from agent actions, such as misconfigurations or performance degradations that do not immediately trigger alerts, may go undetected. Without TNR, there is no guarantee ensuring that such side effects are identified and reverted.

Table 6: **Ablation analysis of STRATUS (GPT-4o) on mitigation problems in ITBench.**

Ablation	Succ. Rate	Time (s)	Cost ($)
STRATUS (4o)	50%	1720.8	6.11
− No Retry	11.1%	267.2	0.72
− Naïve retry w/o undo	50%	1711.9	5.13

D Tool Usage

We develop a set of tools that enable agents to interact with the cloud environment through natural language. We describe the implemented tools in §D.1, and analyze their usage distribution across execution steps in §D.1.

D.1 List of Tools

In order to do complex reasoning and safe system manipulation, STRATUS has a comprehensive suite of tools, including observability tool and command-line tools. We describe some of the tools here.

Observability Tool. STRATUS has tools to observe the clouds, through collecting telemetry data (`TelemetryCollection`), or retrieving system states.

- `GetLogsTool`: Retrieves logs from specific component, e.g., application logs from a given pod.

- `GetTracesTool` and `ReadTracesTool`: Retrieve traces if the system is traced, e.g., traces from Jaeger. `ReadTracesTool` handles large trace files stored on disk.

- `GetBootstrapAnalysisTool` and `ReadBootstrapAnalysisTool`: Support trace-based boot-strapping by identifying likely fault locations. Instead of digesting the entire trace data, the agent will only read the traces with errors (e.g., `500` HTTP responses) as initial analysis results.

- `OracleTools`: After each iteration of diagnosis or mitigation, oracles can be used to validate system health (e.g., alert clearance, workload throughput, cluster health). These oracles determine whether the agent should continue, retry, or terminate.

Note that STRATUS disables direct metrics consumption, as raw numerical data can be difficult for the agent to interpret effectively. Instead, such information is surfaced in logs in a more pre-processed form, e.g., as filtered, aggregated, or human-readable summaries. These tools abstract away raw observability data, enabling the agent to reason over high-level, interpretable signals, and to determine its termination.

Command-line Tools. On the action side, STRATUS has specialized tools to construct the command to change the system states and perform the failthful undo.

- `NL2Kubectl`: Translates natural language instructions generated by the agent into Kubernetes commands. This decouples high-level reasoning from low-level execution syntax.
- `LintingTool`: Acts as an internal tool to check the correctness of the command, e.g., the commands generated by `NL2Kubectl`; if the command is not correct or allowed, the tool will return a message to ask the agent to improve the command.
- `RollbackTool`: Supports the undo agent to perform failthful undo actions.
- `NotificationTool`: Whenever the agents feel confident that the task is finished, it can use the `NotificationTool` to notify human of task completion, or escalation. Notification can be made in different troubleshooting phases e.g., detection phase should notify human whether there is a nomaly in the system; while the agent should notify human whether the mitigation is successful or complete.

D.2 Distribution of Tool Usage

To analyze the tool invocation patterns in STRATUS, we log the tool usage of the STRATUS (GPT-4o) across all problems in the AIOpsLab. For each execution step (i.e., each iteration within the retry loop), we record which tools that interact with the clouds are invoked. In Figure 7, we show the distribution of external tool usage (including the `TelemetryCollection`, `NL2Kubectl`, `NotificationTool`, and `RollbackTool`) by the STRATUS (GPT-4o) at different steps in our AIOpsLab experiments.

STRATUS exhibits distinct phases of tool usage across execution steps. In the initial stage (Iterations 1–2), the `RollbackTool` is predominantly used by the undo agent, and bootstrapping tools are used to perform initial analysis, which are specified in the control flow. In early steps (Iterations 3–8), the agent primarily engages `TelemetryCollection` tools to gather observability data from logs and traces. As execution progresses (Iterations 9–23), `NL2KubectlTool` dominates, indicating the agent's transition from information gathering using provided tools to more customized command generation for diagnosis and mitigation. Additionally, we observe a shift in preference from trace retrieval to log retrieval over time.

E Oracle Usage

In §4.2, we describe how STRATUS determines termination based on the oracles. Here, we use an example (Problem ID: `wrong_bin_usage-mitigation-1` in AIOpslab) to illustrate the usage of our oracles in STRATUS. In this problem, a microservice is misconfigured to use an incorrect binary during container initialization. As a result, while the container is able to start and all pods appear healthy at the orchestration level, the application has functional failures. Specifically, the service responds with a high rate of HTTP errors (non-2xx/3xx responses), indicating that the application is not working as expected.

As illustrated in Figure 8, during the initial validation phase, the system health oracle confirms that all pods are running normally. However, the workload oracle detects a number of erroneous responses (115), revealing that the issue has not been fully resolved. Based on this feedback, STRATUS further troubleshoots the issue, and later correctly identifies the root cause as an incorrect binary and proceeds to patch the deployment with the appropriate version. A second round of oracle validation shows that all checks pass: the cluster remains healthy and the workload oracle confirms the elimination

Figure 7: Distribution of tool usage by STRATUS (GPT-4o) across steps. The figure shows a histogram of tool invocations grouped by type and step index (Iteration).

of HTTP errors (zero non-2xx/3xx responses). With successful validation, STRATUS notifies the benchmark, marking the resolution of the issue.

Figure 8: Oracle usage when STRATUS solving the `wrong_bin_usage-mitigation-1` task. In the first validation round, the workload oracle detects 115 failed responses, indicating the issue is unresolved. STRATUS continues troubleshooting and patches the deployment with the correct binary. In the second round, both oracles confirm success.

F   Hyperparameter Sweep

We perform a hyperparameter sweep over the temperature parameter in STRATUS (GPT-4o) using all 13 mitigation tasks from AIOpsLab. Table 7 shows the results. We can see that the temperature of 0.0 yields the best results. Setting temperature to 0 allows the LLM to output deterministically with a given input. Setting a non-zero temperature allows "creativity" in LLM's output. However, in the SRE context, this may not be desirable. A non-zero temperature permits the LLM to output non-deterministic results given the same input. For example, given a microservice application trace that clearly indicates the root cause of a problem, we would like STRATUS to confidently take *one* correct mitigation action. In addition, setting overly high temperature can directly affect the tool's ability in generating the correct commands, which harms STRATUS's performance critically.

Table 7: **Effect of Temperature on Mitigation Success Rate.**

Temperature	Succ.	Time (s)	Step	$
0.0	69.2%	811.9	46.3	0.877
0.2	51.3%	961.9	71.6	0.647
0.4	46.2%	950.58	67.9	0.493
0.6	56.4%	836.8	55.1	0.388
0.8	33.3%	1209.51	77.2	0.533

G   Result Details

We present detailed evaluation results for STRATUS (GPT-4o) in AIOpsLab. Table 8 shows the results of detection, localization, and RCA tasks. Table 9 shows the mitigation results.

Table 8: Detailed results of Detection, Localization, and RCA for STRATUS (GPT-4o) in AIOpsLab. We use the Pass@3 metric to calculate localization accuracy. Non-existent tasks are denoted as –.

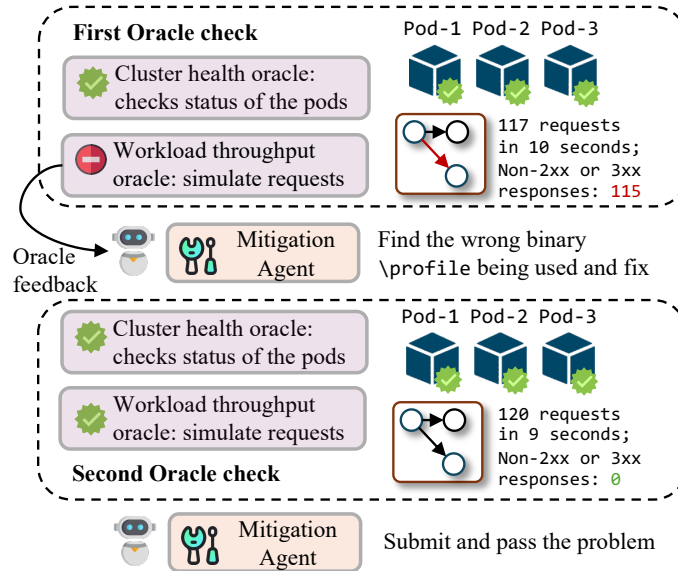| Problem ID	Detection			Localization				RCA		
	Time (s)	Pass	$	Time (s)	Pass@3	Succ.	$	Time (s)	Succ.	$
k8s_target_port-misconfig-1	39.04	T	0.100	29.90	T	100	0.093	49.33	0%	0.073
k8s_target_port-misconfig-2	49.46	T	0.176	27.97	T	100	0.088	28.97	50%	0.089
k8s_target_port-misconfig-3	42.46	T	0.141	41.19	T	100	0.147	41.00	50%	0.075
auth_miss_mongodb	42.26	T	0.117	28.70	T	50	0.115	43.30	0%	0.083
revoke_auth_mongodb-1	42.98	T	0.066	58.26	T	100	0.050	37.05	50%	0.042
revoke_auth_mongodb-2	45.04	T	0.089	33.27	T	100	0.041	45.45	50%	0.056
user_unregistered_mongodb-1	35.63	T	0.043	34.47	T	100	0.041	34.10	50%	0.042
user_unregistered_mongodb-2	38.21	T	0.044	34.26	T	100	0.042	35.72	50%	0.042
misconfig_app_hotel_res	45.74	T	0.095	36.91	T	100	0.029	45.97	100%	0.058
scale_pod_zero_social_net	72.04	T	0.113	35.15	T	100	0.110	42.39	0%	0.076
assign_to_non_existent_node_social_net	28.62	T	0.045	28.91	T	100	0.045	28.54	0%	0.046
container_kill	43.77	T	0.042	39.72	T	100	0.063	–	–	–
pod_failure_hotel_res	68.94	T	0.050	42.97	T	100	0.064	–	–	–
pod_kill_hotel_res	55.75	T	0.072	275.83	F	0	0.120	–	–	–
network_loss_hotel_res	43.17	T	0.061	58.97	F	0	0.070	–	–	–
network_delay_hotel_res	38.83	T	0.082	35.87	F	0	0.081	–	–	–
noop_detection_hotel_reservation	38.62	F	0.101	–	–	–	–	–	–	–
noop_detection_social_network	38.10	F	0.052	–	–	–	–	–	–	–
astronomy_shop_ad_service_failure	87.84	T	0.094	67.51	F	0	0.067	–	–	–
astronomy_shop_ad_service_high_cpu	40.44	T	0.060	67.16	T	50	0.071	–	–	–
astronomy_shop_ad_service_manual_gc	42.17	T	0.086	64.73	F	0	0.055	–	–	–
astronomy_shop_cart_service_failure	38.37	T	0.092	60.96	T	100	0.111	–	–	–
astronomy_shop_image_slow_load	51.59	T	0.072	76.70	T	33.33	0.209	–	–	–
astronomy_shop_kafka_queue_problems	66.00	T	0.126	65.48	F	0	0.059	–	–	–
astronomy_shop_payment_service_failure	65.14	T	0.054	67.03	F	0	0.085	–	–	–
astronomy_shop_payment_service_unreachable	52.18	T	0.065	66.44	F	0	0.045	–	–	–
astronomy_shop_product_catalog_service_failure	36.90	T	0.084	62.56	F	0	0.046	–	–	–
astronomy_shop_recommendation_service_cache_failure	78.68	T	0.101	54.93	F	0	0.090	–	–	–
astronomy_shop_loadgenerator_flood_homepage	29.59	T	1.183	35.45	F	0	1.345	–	–	–
redeploy_without_PV	40.72	T	0.044	–	–	–	–	42.74	0%	0.070
wrong_bin_usage	40.32	T	0.170	298.09	F	0	0.149	40.86	50%	0.137
noop_detection_astronomy_shop	68.54	F	0.056	–	–	–	–	–	–	–
Average	48.35	29/32	0.118	65.34	16/28	51.19	0.126	39.65	34.62%	0.068

30

Table 9: Detailed results of Mitigation tasks for STRATUS (GPT-4o) in AIOpsLab.

Problem ID	Time (s)	Step	Succ.	$
k8s_target_port-misconfig-mitigation-1	124.59	13	T	0.116
k8s_target_port-misconfig-mitigation-2	210.08	23	T	0.338
k8s_target_port-misconfig-mitigation-3	148.49	11	T	0.084
assign_to_non_existent_node_social_net-mitigation-1	115.42	12	T	0.051
scale_pod_zero_social_net-mitigation-1	139.03	88	T	0.232
user_unregistered_mongodb-mitigation-1	2,007.58	123	T	0.330
user_unregistered_mongodb-mitigation-2	2,090.04	46	F	0.955
revoke_auth_mongodb-mitigation-1	1,405.48	121	F	0.436
revoke_auth_mongodb-mitigation-2	1,321.47	108	F	0.644
auth_miss_mongodb-mitigation-1	136.60	20	T	0.148
misconfig_app_hotel_res-mitigation-1	1,367.99	10	F	0.805
redeploy_without_PV-mitigation-1	839.00	152	T	6.704
wrong_bin_usage-mitigation-1	649.44	51	T	0.562
Average	811.94	59.85	9/13	0.877

H   Heuristic Evaluation of STRATUS's Multi-agent Design

STRATUS adopts a multi-agent design. Recent work [53] categorizes common failure modes of multi-agent systems, including *Specification Issues*, *Inter-Agent Misalignment*, and *Task Verification*. We present a heuristic evaluation of the STRATUS design and how the design addresses common failure modes. Table 10 provides a summary.

Table 10: Failure modes from [53] and STRATUS techniques that mitigate them. Abbreviations: DCF=*Deterministic Control Flow*, CRB=*Confinement (role-based)*, CCL=*Confinement (command-level)*, ORC=*Oracles*, STS=*Specialized Toolset*, UDO=*Undo*, LTR=*Linter*, BTS=*Bootstrapping*, ATD=*Agent Thought Dropout*, RWR=*Reflection with Retries*. Each row shows how the technique conceptually addresses the failure.

Failure Mode	STRATUS Technique(s)	Description and Examples
Specification Issues (System Design)		
Disobey Task Specification	CRB, CCL	Role-based confinement assigns specific tasks for different troubleshooting stages; command-level confinement will validate each action;
Disobey Role Specification	CRB, STS	Each agent has clear roles (diagnosis, mitigation, etc) and is given tools corresponding to the agent type. This allows for agent-specific confinement where each agent is only given the necessary tools to complete the task it's been assigned.
Step Repetition	ATD, BTS	Agent thought dropout prevents agents from being trapped in previous misleading thoughts; bootstrapping gives focused start.
Loss of Conversation History	RWR	Persistent agent thoughts retain context across retries.
Unaware of Termination Conditions	ORC	Oracles identify task completion and provide extra context.
Inter-Agent Misalignment (Agent Coordination)		
Conversation Reset	DCF, UDO	Dataflow direction between agents is the same as control flow; undo restores state on incorrect or destructive actions.
Fail to Ask for Clarification	RWR	A new retry round can be initiated if the agent fails the task.
Task Derailment	BTS, ATD, UDO	Bootstrapping orients initial step (i.e., where the agent starts); contaminated thought can be dropped; undo can be used to restore the state.
Information Withholding	DCF	Agents will be forced to give a final answer (a part of CrewAI's design) to determine its success or to notify of failure.
Ignored Other Agent's Input	DCF, RWR	Agents consume its predecessor's output (including output of the previous round in retrying) in the dataflow until the task is completed.
Reasoning-Action Mismatch	–	Probably related more to the model's own capabilities.
Task Verification (Quality Control)		
Premature Termination	ORC, DCF	Task ends only after all agents finish and oracles confirm completion.
No or Incomplete Verification	ORC, CRB, CCL	Task completion and single action are both verified.
Incorrect Verification	ORC	Multiple oracles provide validation to reduce false positives.

Specification Issues. A common failure pattern arises from incomplete or ambiguous role/task definitions, or insufficient system design encoding in agent prompts [53]. STRATUS avoids this failure pattern in multiple aspects (see Table 10). Role-based confinement (CRB) is used to confine the agent's behavior according to the role and task specifications; command-level confinement

(CCL) will validate each action. Also, STRATUS has specialized toolset (STS) for different agents, to retrieve the relevant information from the system, which can facilitate their task completion. **Inter-Agent Misalignment.** The Inter-Agent Misalignment category includes coordination failures among agents, such as one agent's incorrect output misleading downstream agents [53]. Take our context as the example. If the detection agent identifies the wrong error location, the mitigation agent may inadvertently repair the wrong service. STRATUS addresses such issues by employing deterministic control flow (DCF), which coordinates the agents to work together and communicate clearly with each other. STRATUS also uses reflection with retries (RWR), which enables the system to reassess and relaunch a task (within the bounded risk window K). In addition, the undo capability (UDO) allows the system to revert to a safe state after erroneous executions. Furthermore, agent thought dropout (ATD) prevents potentially corrupted or misleading context propagated from previous agents. **Task Verification.** Inadequate verification that fails to detect or correct errors or premature termination of tasks can make multi-agent systems fail [53]. STRATUS deploys three oracles (ORC) including alert oracle, user request oracle, and system health oracle, which periodically assess system states and validate whether termination conditions have been met, to ensure the agent is terminating appropriately. Complementing these, STRATUS has confinement (CRB, CCL, LTR), e.g., command dry-run and syntax check, to ensure that commands are semantically and syntactically checked prior to execution.

I An Example of STRATUS's Trajectory

We show the trajectory of STRATUS when solving a mitigation task (Problem ID: `redeploy_without_PV-mitigation-1`) in AIOpsLab. In this problem, the storageclass of the persistent volumes (PVs) is mistakenly removed during redeployment. As a result, the persistent volumes claims (PVCs) will not be able to find the corresponding storageclass in the new deployment, and most of the pods will be in an erroneous state.

We show the key actions taken by STRATUS in "Gold Action." Statements marked as "Note" are not part of the trajectory but are added for clarity.

Phase: Initialization　　　　　　　　　　　　　　　　　　　　　　　　　　　　　**Run #1**

```
[21:34:50] All pods in namespace 'test-hotel-reservation' are ready.
== Fault Injection ==
[21:34:51] Waiting for namespace 'test-hotel-reservation' to be deleted...
[21:35:18] Namespace 'test-hotel-reservation' has been deleted.
Namespace 'test-hotel-reservation' deleted successfully.
Deleting namespace test-hotel-reservation without deleting the PV.
Redeploying test-hotel-reservation.
Namespace test-hotel-reservation not found. Creating namespace.
Namespace test-hotel-reservation created successfully: namespace/test-hotel-reservation created
```

Task: Bootstrapping　　　　　　　　　　　　　　　　　　　　　　　　　　　**Diagnosis Agent**

```
Analyze the traces to localize which part of the system, e.g., a service, a component, a pod, a node, etc., has
↪  anomalies. Always use the get_traces tool first; then use a read_traces tool to help you with the localization.
↪  Note that it may not be helpful to use the tools too many times. You are working on the namespace
↪  'test-hotel-reservation', please use this namespace when using the tools.
```

Thought & Action　　　　　　　　　　　　　　　　　　　　　　　　　　　　　**Diagnosis Agent**

```
To analyze the traces and localize anomalies in the 'test-hotel-reservation' namespace, I will first use the
get_traces tool to fetch the traces for a specified duration.  Then, I will use the read_traces tool to read and
analyze the traces.
```

```
Call get_traces with {"namespace": "test-hotel-reservation", "duration": 30}
```

Tool Usage: get_traces　　　　　　　　　　　　　　　　　　　　　　　　　　**Diagnosis Agent**

```
Traces data exported to: /eval/xagent/trace_output/traces_1747186602.csv
Please take the next action
```

Thought & Action	**Diagnosis Agent**

Call read_traces with {"file_path": "/eval/xagent/trace_output/traces_1747186602.csv"}

Tool Usage: read_traces	**Diagnosis Agent**

The following list of services are the last service the request reached before returning an error, these services
↪ may not be the exact place of the root cause, so you may need to dig deeper. These services correspond to a
↪ list of operations happened there, which is the service relying on and could be where the root cause comes
↪ from. Here is the list:
1: {service: "search", operation: "geo"}
2: {service: "frontend", operation: "recommendation"}

Task Bootstrapping Finished	**Diagnosis Agent**

- List of entities with anomalies:
  1. Service: "search", Operation: "geo"
  2. Service: "frontend", Operation: "recommendation"

- Possible root cause entities for further analysis:
  - Service: "search", Operation: "geo"
  - Service: "frontend", Operation: "recommendation"

Task: Mitigation	**Mitigation Agent**

The problem description is '''You are an expert DevOps engineer assigned to mitigate anomalies in a deployed
↪ service...

Thought & Action	**Mitigation Agent**

To begin the mitigation process, I need to gather more information about the services with anomalies. The anomalies
are present in the "search" service with the "geo" operation and the "frontend" service with the "recommendation"
operation. I will start by fetching the logs for these services to understand the issues better.
First, I will fetch the logs for the "search" service in the "test-hotel-reservation" namespace.

Call get_logs with {"namespace": "test-hotel-reservation", "service": "search"}

Tool Usage: get_logs	**Mitigation Agent**

```
{"level":"info","time":"2025-05-14T01:36:03Z","message":"TLS disabled."}
{"level":"info","time":"2025-05-14T01:36:03Z","message":"Set global log level: info"}
{"level":"info","time":"2025-05-14T01:36:03Z","message":"Tune: setGCPercent to 100"}
2025-05-14T01:36:03Z INF cmd/search/main.go:24 > Reading config...
2025-05-14T01:36:03Z INF cmd/search/main.go:40 > Read target port: 8082
2025-05-14T01:36:03Z INF cmd/search/main.go:41 > Read consul address: consul:8500
2025-05-14T01:36:03Z INF cmd/search/main.go:42 > Read jaeger address: jaeger:6831
2025-05-14T01:36:03Z INF cmd/search/main.go:51 > Initializing jaeger agent [service name: search | host:
↪ jaeger:6831]...
2025-05-14T01:36:03Z INF tracing/tracer.go:26 > Jaeger client: adjusted sample ratio 1.000000
2025-05-14T01:36:03Z INF cmd/search/main.go:56 > Jaeger agent initialized
2025-05-14T01:36:03Z INF cmd/search/main.go:58 > Initializing consul agent [host: consul:8500]...
2025-05-14T01:36:03Z INF cmd/search/main.go:63 > Consul agent initialized
2025-05-14T01:36:03Z INF cmd/search/main.go:74 > Starting server...
2025-05-14T01:36:03Z INF registry/registry.go:91 > Trying to register service [ name: srv-search, id:
↪ 254379a6-bcf4-478c-a828-268a443478fb, address: 10.244.2.21:8082 ]
2025-05-14T01:36:03Z INF services/search/server.go:100 > Successfully registered in consul
```

Please take the next action

Thought & Action	**Mitigation Agent**

The logs for the "search" service do not indicate any errors or anomalies. The service seems to be initializing and
registering successfully. Next, I will fetch the logs for the "frontend" service in the "test-hotel-reservation"
namespace to investigate the "recommendation" operation anomaly.

Call get_logs with {"namespace": "test-hotel-reservation", "service": "frontend"}

Tool Usage: get_logs — Mitigation Agent

{"level":"info","time":"2025-05-14T01:35:46Z","message":"TLS disabled."}
{"level":"info","time":"2025-05-14T01:35:46Z","message":"Set global log level: info"}
{"level":"info","time":"2025-05-14T01:35:46Z","message":"Tune: setGCPercent to 100"}
2025-05-14T01:35:46Z INF cmd/frontend/main.go:23 > Reading config...
...
2025/05/14 01:35:46 grpc/lb/consul: error retrieving instances from Consul: Get
↪ "http://consul:8500/v1/health/service/srv-search?passing=1": dial tcp 10.96.73.138:8500: connect: connection
↪ refused
...
Please take the next action

Thought & Action — Mitigation Agent

The logs for the "frontend" service indicate a connection issue with the Consul service. The error message "dial tcp 10.96.73.138:8500: connect: connection refused" suggests that the frontend service is unable to connect to the Consul service, which is likely causing the anomaly in the "recommendation" operation.
To address this issue, I will check the status of the Consul service in the "test-hotel-reservation" namespace to ensure it is running and accessible.

Call NL2Kubectl Tool with {"nl_query": "get pods -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

```
NAME                                    READY   STATUS    RESTARTS      AGE
consul-b84877fb4-v9sgv                  1/1     Running   0             74s
frontend-69d95967fb-cgrjc               1/1     Running   0             73s
geo-99d8c58c-5wrbn                      0/1     Error     2 (32s ago)   73s
jaeger-67445b6dd6-7kbpd                 1/1     Running   0             71s
memcached-profile-66bb9c8b9d-86bpf      1/1     Running   0             70s
memcached-rate-68585796cb-gtpg6         1/1     Running   0             67s
memcached-reserve-646c7dcd6b-vwz8t      1/1     Running   0             62s
mongodb-geo-5ff578bcb8-bg6cr            0/1     Pending   0             71s
mongodb-profile-758cb77f9f-thlpl        0/1     Pending   0             70s
mongodb-rate-56cc8659c9-7dwkw           0/1     Pending   0             66s
mongodb-recommendation-55699465f7-f9hng 0/1     Pending   0             64s
mongodb-reservation-5994859869-zvvmt    0/1     Pending   0             60s
mongodb-user-969c6c449-86tnh            0/1     Pending   0             58s
profile-676fcc86d5-v7c55                1/1     Running   2 (26s ago)   69s
rate-c9bc58c85-klcq7                    1/1     Running   2 (24s ago)   65s
recommendation-8675c6c9ff-hqbl8         1/1     Running   2 (22s ago)   64s
reservation-7699b8fdd4-756lh            0/1     Error     1 (31s ago)   60s
search-65df96d8df-6lnw2                 1/1     Running   0             59s
user-8477d787d8-d8d45                   0/1     Error     1 (29s ago)   57s
```

Thought & Action — Mitigation Agent

The Consul pod is running, which suggests that the issue might not be with the Consul service itself but rather with network connectivity or configuration. The error message "connection refused" typically indicates that the service is not reachable at the specified address or port.
Given that the Consul pod is running, I will now check the service configuration for Consul to ensure it is correctly set up to expose the necessary ports.

Call NL2Kubectl Tool with {"nl_query": "describe service consul -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

```
Name:              consul
...
Port:              8300  8300/TCP
TargetPort:        8300/TCP
Endpoints:         10.244.2.17:8300
Port:              8400  8400/TCP
TargetPort:        8400/TCP
Endpoints:         10.244.2.17:8400
Port:              8500  8500/TCP
TargetPort:        8500/TCP
Endpoints:         10.244.2.17:8500
Port:              8600  8600/UDP
TargetPort:        53/UDP
Endpoints:         10.244.2.17:53
...
```

Thought & Action	**Mitigation Agent**

The Consul service is correctly configured to expose port 8500, which is the port the frontend service is trying
to connect to. The IP address and port are correctly set, and the endpoints are available. This suggests that
the issue might be related to network policies or firewall rules that are preventing the frontend service from
connecting to the Consul service.
To further investigate, I will check if there are any network policies in place that might be affecting connectivity
between the frontend and Consul services.

Call NL2Kubectl Tool with {"nl_query": "get networkpolicy -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	**Mitigation Agent**

Thought & Action	**Mitigation Agent**

There are no network policies listed in the "test-hotel-reservation" namespace, which means there are no
restrictions on network traffic between the services. This suggests that the issue might be related to the
configuration of the frontend service itself or a misconfiguration in the Consul service.
Given the information gathered, the root cause of the anomaly in the "frontend" service's "recommendation" operation
appears to be a connectivity issue with the Consul service. The error message "connection refused" indicates that
the frontend service is unable to reach the Consul service at the specified address and port.
Mitigation Plan:
1. Verify the network configuration and ensure that the frontend service can reach the Consul service on port 8500.
2. Check the firewall settings and ensure that there are no rules blocking traffic between the frontend and Consul
services.
3. Review the configuration of the frontend service to ensure it is correctly set up to connect to the Consul
service.
4. If necessary, restart the Consul service to ensure it is running correctly and accepting connections.
Thought: I have identified the root cause of the anomaly and outlined a mitigation plan. I will now submit the
solution.

Call submit with {}

Phase: Submission Validation	**Run #1**

Submission triggered. Validating...
Waiting 15 seconds for changes to take effect...
Testing workload generator...
...
  117 requests in 10.00s, 30.62KB read
  Non-2xx or 3xx responses: 117
Requests/sec: 11.70
Transfer/sec: 3.06KB

Tool Usage: submit	**Mitigation Agent**

The system is not in a valid state. Please resolve all issues before submitting.

Thought & Action	**Mitigation Agent**

The submission failed because the system is not in a valid state, indicating unresolved issues. I need to further
investigate and resolve any remaining issues before submitting the solution.
The "geo" operation in the "search" service also has an anomaly. I will check the logs for the "geo" pod to gather
more information about the error status.

Call get_logs with {"namespace": "test-hotel-reservation", "service": "geo"}

Tool Usage: get_logs	**Mitigation Agent**

...
2025-05-14T01:37:15Z INF cmd/geo/main.go:37 > Initializing DB connection...
2025-05-14T01:37:27Z PNC cmd/geo/db.go:26 > no reachable servers
panic: no reachable servers

goroutine 1 [running]:
github.com/rs/zerolog.(*Logger).Panic.func1({0x96e211, 0x0})
...
Please take the next action

Thought & Action   **Mitigation Agent**

```
The logs for the "geo" service indicate a panic due to "no reachable servers," which suggests a connectivity issue
with the MongoDB database.  The error occurs during the initialization of the database connection, indicating that
the "geo" service is unable to connect to the MongoDB instance at "mongodb-geo:27017."
To address this issue, I will check the status of the MongoDB pod for the "geo" service to ensure it is running and
accessible.
```

```
Call NL2Kubectl Tool with {"nl_query":  "describe pod mongodb-geo-5ff578bcb8-bg6cr -n test-hotel-reservation"}
```

Tool Usage: NL2Kubectl Tool   **Mitigation Agent**

```
Name:           mongodb-geo-5ff578bcb8-bg6cr
Namespace:      test-hotel-reservation
Priority:       0
Service Account: default
Node:           <none>
Labels:         io.kompose.service=mongodb-geo
                pod-template-hash=5ff578bcb8
...
Events:
  Type      Reason            Age    From             Message
  ----      ------            ----   ----             -------
  Warning   FailedScheduling  2m     default-scheduler  0/3 nodes are available: pod has unbound immediate
  ↪   PersistentVolumeClaims. preemption: 0/3 nodes are available: 3 Preemption is not helpful for scheduling.
```

Thought & Action   **Mitigation Agent**

```
The "mongodb-geo" pod is in a "Pending" state due to a failed scheduling event.  The message "pod has unbound
immediate PersistentVolumeClaims" indicates that the pod is unable to start because it cannot bind to the required
PersistentVolumeClaim (PVC) named "geo-pvc."
To resolve this issue, I need to check the status of the "geo-pvc" to ensure it is correctly configured and
available.
```

```
Call NL2Kubectl Tool with {"nl_query":  "describe pvc geo-pvc -n test-hotel-reservation"}
```

Tool Usage: NL2Kubectl Tool   **Mitigation Agent**

```
Name:          geo-pvc
Namespace:     test-hotel-reservation
StorageClass:  geo-storage
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       mongodb-geo-5ff578bcb8-bg6cr
Events:
  Type      Reason             Age              From                     Message
  ----      ------             ----             ----                     -------
  Warning   ProvisioningFailed  4s (x9 over 2m4s)  persistentvolume-controller  storageclass.storage.k8s.io
  ↪   "geo-storage" not found
```

Thought & Action   **Mitigation Agent**

```
The "geo-pvc" is in a "Pending" state because the storage class "geo-storage" is not found.  This is preventing
the PersistentVolumeClaim from being bound, which in turn is causing the "mongodb-geo" pod to remain in a "Pending"
state.
To resolve this issue, I need to create or correct the storage class "geo-storage" so that the PVC can be bound and
the pod can start.
Mitigation Plan:
1.  Create a storage class named "geo-storage" with the appropriate configuration to support the "geo-pvc"
PersistentVolumeClaim.
2.  Ensure that the storage class is available and correctly configured to allow the PVC to bind.
3.  Once the storage class is created, the "geo-pvc" should bind successfully, allowing the "mongodb-geo" pod to
start.
I will now proceed to create the storage class "geo-storage."
```

```
Call NL2Kubectl Tool with {"nl_query":  "create storageclass geo-storage --provisioner=kubernetes.io/aws-ebs
--parameters type=gp2"}
```

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
Dry-run failed. Potentially it's an invalid command. stderr: error: unknown flag: --provisioner
See 'kubectl create --help' for usage.
```

Thought & Action	Mitigation Agent

It seems the command to create the storage class was incorrect. I will correct the command and attempt to create the storage class "geo-storage" using a valid configuration.

Call NL2Kubectl Tool with {"nl_query": "apply -f - «EOF\napiVersion: storage.k8s.io/v1\nkind: StorageClass\nmetadata:\n name: geo-storage\nprovisioner: kubernetes.io/aws-ebs\nparameters:\n type: gp2\nEOF"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
storageclass.storage.k8s.io/geo-storage created
```

Thought & Action	Mitigation Agent

The storage class "geo-storage" has been successfully created. I will now check the status of the "geo-pvc" to ensure it is bound and the "mongodb-geo" pod can start.

Call NL2Kubectl Tool with {"nl_query": "describe pvc geo-pvc -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
Name:          geo-pvc
Namespace:     test-hotel-reservation
StorageClass:  geo-storage
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       mongodb-geo-5ff578bcb8-bg6cr
Events:
  Type     Reason              Age                   From                        Message
  ----     ------              ----                  ----                        -------
  Warning  ProvisioningFailed  14s (x9 over 2m14s)   persistentvolume-controller  storageclass.storage.k8s.io
  ↪  "geo-storage" not found
```

Thought & Action	Mitigation Agent

The "geo-pvc" is still in a "Pending" state, and the error message indicates that the storage class "geo-storage" is not found. This suggests that there might be an issue with the creation of the storage class or its recognition by the persistent volume controller.
To resolve this, I will verify the existence of the storage class "geo-storage" and ensure it is correctly configured.

Call NL2Kubectl Tool with {"nl_query": "get storageclass geo-storage"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
NAME          PROVISIONER            RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
geo-storage   kubernetes.io/aws-ebs  Delete          Immediate          false                  6s
```

Thought & Action	Mitigation Agent

The storage class "geo-storage" exists and is correctly configured with the provisioner "kubernetes.io/aws-ebs."
However, the persistent volume controller is still unable to find it, which suggests a potential issue with the cluster configuration or the controller itself.
Given the information gathered, the root cause of the anomaly in the "geo" operation of the "search" service is the inability to bind the "geo-pvc" due to the persistent volume controller not recognizing the storage class "geo-storage."
Mitigation Plan:
1. Verify the cluster configuration to ensure the persistent volume controller is correctly set up to recognize storage classes.
2. Check the logs of the persistent volume controller for any errors or misconfigurations.

```
3.  If necessary, restart the persistent volume controller to ensure it is running correctly and can recognize the
storage class.
Thought:  I have identified the root cause of the anomaly and outlined a mitigation plan.  I will now submit the
solution.

─────────────────────────────────────────────────────────────────────────────

Call submit with {}
```

Phase: Submission Validation   Run #1

```
Submission triggered. Validating...
Waiting 15 seconds for changes to take effect...
Testing workload generator...
...
  117 requests in 10.00s, 30.62KB read
  Non-2xx or 3xx responses: 117
Requests/sec:    11.70
Transfer/sec:     3.06KB
```

Tool Usage: submit   Mitigation Agent

```
The system is not in a valid state. Please resolve all issues before submitting.
```

```
...

A series of observation and submission.
...
```

Task Mitigation Finished   Mitigation Agent

```
Root Cause Analysis:
1. The "frontend" service's "recommendation" operation anomaly is due to a connectivity issue with the Consul
↪   service. The error message "connection refused" indicates that the frontend service is unable to reach the
↪   Consul service at the specified address and port.
2. The "geo" operation in the "search" service is experiencing a connectivity issue with the MongoDB database. The
↪   error "no reachable servers" suggests that the "geo" service is unable to connect to the MongoDB instance due
↪   to the "geo-pvc" being in a "Pending" state.
3. The "geo-pvc" is unable to bind because the persistent volume controller is not recognizing the storage class
↪   "geo-storage," despite it being correctly configured.

Mitigation Plan:
1. For the "frontend" service:
   - Verify the network configuration and ensure that the frontend service can reach the Consul service on port
↪    8500.
   - Check firewall settings to ensure there are no rules blocking traffic between the frontend and Consul services.
   - Review the configuration of the frontend service to ensure it is correctly set up to connect to the Consul
↪    service.
   - Restart the Consul service if necessary to ensure it is running correctly and accepting connections.

2. For the "geo" operation:
   - Verify the cluster configuration to ensure the persistent volume controller is correctly set up to recognize
↪    storage classes.
   - Check the cluster setup to ensure all necessary components are installed and configured.
   - Restart the persistent volume controller if necessary to ensure it is running correctly and can recognize the
↪    storage class "geo-storage."

These steps should help resolve the anomalies in the "frontend" and "search" services.
```

Phase: Validation & Retry   Run #1

```
-----------------------------------------------------------------------
  117 requests in 10.00s, 44.57KB read
  Non-2xx or 3xx responses: 63
Requests/sec:    11.70
Transfer/sec:     4.46KB

Adding issues ['  Non-2xx or 3xx responses: 117']
Issues found with pod geo-99d8c58c-5wrbn:
  - Container hotel-reserv-geo is in CrashLoopBackOff
...
Issues found with pod user-8477d787d8-d8d45:
  - Container hotel-reserv-user is in CrashLoopBackOff
Found 12 issues in the cluster.
Adding issues ['Container hotel-reserv-geo is in CrashLoopBackOff', ...]
Validation result: {'success': False, 'issues': ['  Non-2xx or 3xx responses: 117', 'Container hotel-reserv-geo is
↪   in CrashLoopBackOff', ...]}
Output written to: eval/05-13_21-28-46-redeploy_without_PV-mitigation-1/lumyn_output/agent_output_0.json
Analyzing previous run results for improvement...
Reflection generated successfully.
```

Task: Rollback   **Rollback Agent**

```
You are given a system and a rollback tool. The system is either in a initial faulty state, or in the state you
↪  left it incompletely repaired after the previous round of mitigation. You need to rollback the system to the
↪  initial state since the operations you performed in the previous round may negatively impact your current round
↪  of mitigation.
You should run the rollback tool as many times as you can, since this rollback tool undos the last operation
↪  performed on the system. You should run the rollback tool until it returns that there are no more operations to
↪  undo.
```

Thought & Action   **Rollback Agent**

```
To rollback the system to its initial state, I need to use the rollback tool repeatedly until it indicates that
there are no more operations to undo.
```

```
Call rollback_tool with {"argument":  "rollback the last operation"}
```

Tool Usage: rollback_tool   **Rollback Agent**

```
Rolled back the previous command: kubectl apply -f - <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: geo-storage
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
EOF, using rollback:kubectl delete storageclass.storage.k8s.io geo-storage -n test-hotel-reservation
```

Thought & Action   **Rollback Agent**

```
Thought:  I need to provide a different argument to the rollback tool to continue the rollback process.
```

```
Call rollback_tool with {"argument":  "undo the previous operation"}
```

Tool Usage: rollback_tool   **Rollback Agent**

```
No more actions to rollback.
```

Task Rollback Finished   **Rollback Agent**

```
None.
```

Task: Bootstrapping   **Diagnosis Agent**

```
Analyze the traces to localize which part of the system, e.g., a service, a component, a pod, a node, etc., has
↪  anomalies. Always use the get_traces tool first; then use a read_traces tool to help you with the localization.
↪  Note that it may not be helpful to use the tools too many times. You are working on the namespace
↪  'test-hotel-reservation', please use this namespace when using the tools.
```

Thought & Action   **Diagnosis Agent**

I need to fetch traces from the 'test-hotel-reservation' namespace to analyze and identify any anomalies in the
system. I'll start by using the get_traces tool to retrieve the necessary data.

Call get_traces with {"namespace": "test-hotel-reservation", "duration": 60}

Tool Usage: get_traces   **Diagnosis Agent**

Traces data exported to: /eval/xagent/trace_output/traces_1747186964.csv
Please take the next action

Thought & Action   **Diagnosis Agent**

Thought: I have successfully fetched the traces data. Now, I need to read the traces from the file to analyze and
identify any anomalies in the system.

Call read_traces with {"file_path": "/eval/xagent/trace_output/traces_1747186964.csv"}

Tool Usage: read_traces   **Diagnosis Agent**

The following list of services are the last service the request reached before returning an error, these services
↪   may not be the exact place of the root cause, so you may need to dig deeper. These services correspond to a
↪   list of operations happened there, which is the service relying on and could be where the root cause comes
↪   from. Here is the list:
1: {service: "frontend", operation: "on/GetRecommendations"}
2: {service: "search", operation: "geo"}
3: {service: "frontend", operation: "user"}

Task Bootstrapping Finished   **Diagnosis Agent**

- List of entities with anomalies:
   1. Service: "frontend", Operation: "on/GetRecommendations"
   2. Service: "search", Operation: "geo"
   3. Service: "frontend", Operation: "user"

- Possible root cause entities:
   - The anomalies may originate from the "frontend" service, particularly in the "on/GetRecommendations" and "user"
   ↪   operations, as well as the "search" service in the "geo" operation. Further analysis can start from these
   ↪   root cause entities.

Task: Mitigation   **Mitigation Agent**

The problem description is '''You are an expert DevOps engineer assigned to mitigate anomalies in a deployed
↪   service...

Reflection on the previous run: Root Cause:
1. The anomaly in the "frontend" service's "recommendation" operation is due to a connectivity issue with the
↪   Consul service, indicated by the error message "connection refused."
2. The anomaly in the "geo" operation of the "search" service is due to the inability to bind the "geo-pvc" because
↪   the persistent volume controller is not recognizing the storage class "geo-storage."

Mitigation Plan:
1. For the "frontend" service:
    - Verify the network configuration and ensure that the frontend service can reach the Consul service on port
    ↪   8500.
    - Check the firewall settings and ensure that there are no rules blocking traffic between the frontend and
    ↪   Consul services.
    - Review the configuration of the frontend service to ensure it is correctly set up to connect to the Consul
    ↪   service.
    - If necessary, restart the Consul service to ensure it is running correctly and accepting connections.

2. For the "geo" operation in the "search" service:
    - Verify the cluster configuration to ensure the persistent volume controller is correctly set up to recognize
    ↪   storage classes.
    - Check the cluster setup to ensure all necessary components are installed and configured.
    - If necessary, restart the persistent volume controller to ensure it is running correctly and can recognize the
    ↪   storage class..

The following issues were found in the cluster:
-   Non-2xx or 3xx responses: 117
- Container hotel-reserv-geo is in CrashLoopBackOff
- Pod mongodb-geo-5ff578bcb8-bg6cr is in Pending state
- Pod mongodb-profile-758cb77f9f-thlpl is in Pending state
- ...

Thought & Action	**Mitigation Agent**

The previous run has already identified the root causes and provided a mitigation plan. I will follow the plan to resolve the issues. The plan involves two main tasks: fixing the connectivity issue with the Consul service for the "frontend" service and addressing the persistent volume controller issue for the "geo" operation in the "search" service.
First, I will verify the network configuration and ensure that the frontend service can reach the Consul service on port 8500. I will check the firewall settings and review the configuration of the frontend service. If necessary, I will restart the Consul service.
Next, I will verify the cluster configuration to ensure the persistent volume controller is correctly set up to recognize storage classes. I will check the cluster setup and restart the persistent volume controller if necessary.
Let's start by checking the network configuration for the "frontend" service to ensure it can reach the Consul service.

Call NL2Kubectl Tool with {"nl_query": "describe service frontend -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	**Mitigation Agent**

```
Name:                   frontend
Namespace:              test-hotel-reservation
Labels:                 io.kompose.service=frontend
Annotations:            kompose.cmd: kompose convert
                        kompose.version: 1.22.0 (955b78124)
Selector:               io.kompose.service=frontend
Type:                   ClusterIP
IP Family Policy:       SingleStack
IP Families:            IPv4
IP:                     10.96.54.225
IPs:                    10.96.54.225
Port:                   5000  5000/TCP
TargetPort:             5000/TCP
...
```

Thought & Action	**Mitigation Agent**

The "frontend" service is configured with a ClusterIP and is listening on port 5000. However, the issue is related to connectivity with the Consul service on port 8500. I need to check if the Consul service is running and accessible on the expected port.
Next, I will describe the Consul service to verify its configuration and ensure it is running correctly.

Call NL2Kubectl Tool with {"nl_query": "describe service consul -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	**Mitigation Agent**

```
Name:                   consul
...
Port:                   8300  8300/TCP
TargetPort:             8300/TCP
Endpoints:              10.244.2.17:8300
Port:                   8400  8400/TCP
TargetPort:             8400/TCP
Endpoints:              10.244.2.17:8400
Port:                   8500  8500/TCP
TargetPort:             8500/TCP
Endpoints:              10.244.2.17:8500
Port:                   8600  8600/UDP
TargetPort:             53/UDP
Endpoints:              10.244.2.17:53
...
```

Thought & Action	**Mitigation Agent**

The Consul service is correctly configured with a ClusterIP and is listening on port 8500, which matches the expected configuration. The endpoints are also correctly set up. This suggests that the issue might be related to firewall settings or the frontend service configuration.
I will now check the firewall settings to ensure there are no rules blocking traffic between the frontend and Consul services.

Call NL2Kubectl Tool with {"nl_query": "get networkpolicy -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	**Mitigation Agent**

Thought & Action	Mitigation Agent

There are no network policies defined in the "test-hotel-reservation" namespace, which means there are no firewall rules blocking traffic between the frontend and Consul services. The issue might be related to the frontend service configuration.
I will now check the configuration of the frontend service to ensure it is correctly set up to connect to the Consul service.

Call NL2Kubectl Tool with {"nl_query": "describe deployment frontend -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
Name:                   frontend
Namespace:              test-hotel-reservation
CreationTimestamp:      Tue, 13 May 2025 21:35:37 -0400
Labels:                 io.kompose.service=frontend
Annotations:            deployment.kubernetes.io/revision: 1
                        kompose.cmd: kompose convert
                        kompose.version: 1.22.0 (955b78124)
Selector:               io.kompose.service=frontend
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
...
```

Thought & Action	Mitigation Agent

The frontend deployment is configured correctly, and there are no apparent issues with the environment variables or command setup. The deployment is running and available, which suggests that the connectivity issue might be transient or related to the Consul service itself.
I will proceed to restart the Consul service to ensure it is running correctly and accepting connections.

Call NL2Kubectl Tool with {"nl_query": "rollout restart deployment consul -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
deployment.apps/consul restarted
```

Thought & Action	Mitigation Agent

The Consul deployment has been restarted. This should help resolve any transient connectivity issues between the frontend service and the Consul service.
Next, I will address the issue with the "geo" operation in the "search" service, which is related to the persistent volume controller not recognizing the storage class "geo-storage." I will start by checking the persistent volume claims to verify their status.

Call NL2Kubectl Tool with {"nl_query": "get pvc -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
NAME                STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS           VOLUMEATTRIBUTESCLASS
↪   AGE
geo-pvc             Pending                                     geo-storage            <unset>
↪   7m23s
profile-pvc         Pending                                     profile-storage        <unset>
↪   7m20s
rate-pvc            Pending                                     rate-storage           <unset>
↪   7m16s
recommendation-pvc  Pending                                     recommendation-storage <unset>
↪   7m14s
reservation-pvc     Pending                                     reservation-storage    <unset>
↪   7m11s
user-pvc            Pending                                     user-storage           <unset>
↪   7m8s
```

Thought & Action	Mitigation Agent

The persistent volume claims (PVCs) are all in a "Pending" state, indicating that they are not being bound to any persistent volumes. This is likely due to the storage classes not being recognized by the persistent volume controller.
I will now check the storage classes to ensure they are correctly configured and recognized by the cluster.

Call NL2Kubectl Tool with {"nl_query": "get storageclass"}

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
NAME                              PROVISIONER          RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
↪   AGE
openebs-device                    openebs.io/local     Delete          WaitForFirstConsumer   false
↪   13m
openebs-hostpath (default)        openebs.io/local     Delete          WaitForFirstConsumer   false
↪   13m
standard (default)                rancher.io/local-path Delete         WaitForFirstConsumer   false
↪   14m
```

Thought & Action	Mitigation Agent

```
The storage classes "geo-storage", "profile-storage", "rate-storage", "recommendation-storage",
"reservation-storage", and "user-storage" are not listed among the available storage classes.  This indicates
that these storage classes are not defined in the cluster, which is why the PVCs are in a "Pending" state.
To resolve this issue, I need to create the missing storage classes.  I will start by creating the "geo-storage"
class.
```

```
Call NL2Kubectl Tool with {"nl_query":  "create storageclass geo-storage --provisioner=rancher.io/local-path
--reclaim-policy=Delete --volume-binding-mode=WaitForFirstConsumer"}
```

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
Dry-run failed. Potentially it's an invalid command. stderr: error: unknown flag: --provisioner
See 'kubectl create --help' for usage.
```

Gold Action 1	Mitigation Agent

```
It seems the command to create a storage class was incorrect.  I will use the correct syntax to create the
"geo-storage" storage class using a YAML file.
```

```
Call NL2Kubectl Tool with {"nl_query":  "apply -f - «EOF\napiVersion:  storage.k8s.io/v1\nkind:
StorageClass\nmetadata:\n name:  geo-storage\nprovisioner:  rancher.io/local-path\nreclaimPolicy:
Delete\nvolumeBindingMode:  WaitForFirstConsumer\nEOF"}
```

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
storageclass.storage.k8s.io/geo-storage created
```

Gold Action 2	Mitigation Agent

```
The "geo-storage" storage class has been successfully created.  I will proceed to create the other missing storage
classes:  "profile-storage", "rate-storage", "recommendation-storage", "reservation-storage", and "user-storage".
```

```
Call NL2Kubectl Tool with {"nl_query":  "apply -f - «EOF\napiVersion:  storage.k8s.io/v1\nkind:
StorageClass\nmetadata:\n name:  profile-storage\nprovisioner:  rancher.io/local-path\nreclaimPolicy:
Delete\nvolumeBindingMode:  WaitForFirstConsumer\nEOF"}
```

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
storageclass.storage.k8s.io/profile-storage created
```

Gold Action 3	Mitigation Agent

```
The "profile-storage" storage class has been successfully created.  I will continue to create the remaining storage
classes.
```

```
Call NL2Kubectl Tool with {"nl_query":  "apply -f - «EOF\napiVersion:  storage.k8s.io/v1\nkind:
StorageClass\nmetadata:\n name:  rate-storage\nprovisioner:  rancher.io/local-path\nreclaimPolicy:
Delete\nvolumeBindingMode:  WaitForFirstConsumer\nEOF"}
```

Tool Usage: NL2Kubectl Tool	Mitigation Agent

```
storageclass.storage.k8s.io/rate-storage created
```

Gold Action 4 — Mitigation Agent

The "rate-storage" storage class has been successfully created. I will proceed to create the "recommendation-storage" storage class.

Call NL2Kubectl Tool with {"nl_query": "apply -f - «EOF\napiVersion: storage.k8s.io/v1\nkind: StorageClass\nmetadata:\n name: recommendation-storage\nprovisioner: rancher.io/local-path\nreclaimPolicy: Delete\nvolumeBindingMode: WaitForFirstConsumer\nEOF"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

storageclass.storage.k8s.io/recommendation-storage created

Gold Action 5 — Mitigation Agent

The "recommendation-storage" storage class has been successfully created. I will proceed to create the "reservation-storage" storage class.

Call NL2Kubectl Tool with {"nl_query": "apply -f - «EOF\napiVersion: storage.k8s.io/v1\nkind: StorageClass\nmetadata:\n name: reservation-storage\nprovisioner: rancher.io/local-path\nreclaimPolicy: Delete\nvolumeBindingMode: WaitForFirstConsumer\nEOF"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

storageclass.storage.k8s.io/reservation-storage created

Gold Action 6 — Mitigation Agent

The "reservation-storage" storage class has been successfully created. I will proceed to create the "user-storage" storage class.

Call NL2Kubectl Tool with {"nl_query": "apply -f - «EOF\napiVersion: storage.k8s.io/v1\nkind: StorageClass\nmetadata:\n name: user-storage\nprovisioner: rancher.io/local-path\nreclaimPolicy: Delete\nvolumeBindingMode: WaitForFirstConsumer\nEOF"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

storageclass.storage.k8s.io/user-storage created

Thought & Action — Mitigation Agent

The "user-storage" storage class has been successfully created. All the necessary storage classes are now in place. This should resolve the issue with the persistent volume claims being in a "Pending" state.
I will now check the status of the persistent volume claims to ensure they are bound correctly.

Call NL2Kubectl Tool with {"nl_query": "get pvc -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

```
NAME                STATUS    VOLUME                                       CAPACITY   ACCESS MODES   STORAGECLASS
↪   VOLUMEATTRIBUTESCLASS    AGE
geo-pvc             Bound     pvc-2937753a-cc60-4904-9371-d89b0db18ecd     1Gi        RWO            geo-storage
↪   <unset>                  7m53s
profile-pvc         Bound     pvc-a4f1d897-89ca-4cdf-8f94-0958942aa243     1Gi        RWO            profile-storage
↪   <unset>                  7m50s
rate-pvc            Pending                                                                          rate-storage
↪   <unset>                  7m46s
recommendation-pvc  Pending
↪   recommendation-storage   <unset>                 7m44s
reservation-pvc     Pending
↪   reservation-storage      <unset>                 7m41s
user-pvc            Pending                                                                          user-storage
↪   <unset>                  7m38s
```

Thought & Action   Mitigation Agent

The "geo-pvc" and "profile-pvc" are now bound, indicating that the storage classes are being recognized. However,
the other PVCs are still in a "Pending" state. This might be due to resource constraints or scheduling issues.
I will check the events for one of the pending PVCs to understand why it is not being bound.

Call NL2Kubectl Tool with {"nl_query": "describe pvc rate-pvc -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool   Mitigation Agent

```
Name:          rate-pvc
Namespace:     test-hotel-reservation
StorageClass:  rate-storage
Status:        Pending
Volume:
Labels:        <none>
Annotations:   volume.beta.kubernetes.io/storage-provisioner: rancher.io/local-path
               volume.kubernetes.io/selected-node: kind-worker2
               volume.kubernetes.io/storage-provisioner: rancher.io/local-path
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       mongodb-rate-56cc8659c9-7dwkw
Events:
  Type    Reason             Age                    From
  ↪  Message
  ----    ------             ----                   ----
  ↪  -------
  Warning ProvisioningFailed    100s (x26 over 7m48s)  persistentvolume-controller
  ↪  storageclass.storage.k8s.io "rate-storage" not found
  Normal   WaitForPodScheduled  10s                    persistentvolume-controller
  ↪  waiting for pod mongodb-rate-56cc8659c9-7dwkw to be scheduled
  Normal   ExternalProvisioning  5s                    persistentvolume-controller
  ↪  Waiting for a volume to be created either by the external provisioner 'rancher.io/local-path' or manually by
  ↪  the system administrator. If volume creation is delayed, please verify that the provisioner is running and
  ↪  correctly registered.
  Normal   Provisioning          5s
  ↪  rancher.io/local-path_local-path-provisioner-58cc7856b6-n29rr_bf374c2e-efd5-4d8b-a53f-0ac317a46e1a  External
  ↪  provisioner is provisioning volume for claim "test-hotel-reservation/rate-pvc"
```

Thought & Action   Mitigation Agent

The "rate-pvc" is pending due to the storage class "rate-storage" not being found initially, but it seems the
provisioner is now attempting to provision the volume. This indicates that the issue might resolve itself as the
provisioner completes its task.
I will check the status of the persistent volume claims again to see if they have been bound.

Call NL2Kubectl Tool with {"nl_query": "get pvc -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool   Mitigation Agent

```
NAME              STATUS   VOLUME                                    CAPACITY  ACCESS MODES  STORAGECLASS
↪  VOLUMEATTRIBUTESCLASS   AGE
geo-pvc           Bound    pvc-2937753a-cc60-4904-9371-d89b0db18ecd  1Gi       RWO           geo-storage
↪  <unset>                 7m57s
profile-pvc       Bound    pvc-a4f1d897-89ca-4cdf-8f94-0958942aa243  1Gi       RWO           profile-storage
↪  <unset>                 7m54s
rate-pvc          Pending                                                                    rate-storage
↪  <unset>                 7m50s
recommendation-pvc Pending
↪  recommendation-storage  <unset>                 7m48s
reservation-pvc   Bound    pvc-b7dfe091-9243-41ba-8175-92d350c81f95  1Gi       RWO
↪  reservation-storage     <unset>                 7m45s
user-pvc          Pending                                                                    user-storage
↪  <unset>                 7m42s
```

Thought & Action   Mitigation Agent

The "reservation-pvc" has been successfully bound, but "rate-pvc", "recommendation-pvc", and "user-pvc" are still
pending. The provisioner is working on these claims, and they may resolve shortly.
I will now check the logs of the "hotel-reserv-geo" container to see if there are any errors related to the
CrashLoopBackOff state.

Call get_logs with {"namespace": "test-hotel-reservation", "service": "hotel-reserv-geo"}

Tool Usage: get_logs — Mitigation Agent

```
Error: Your service/namespace does not exist. Use kubectl to check.
Please take the next action
```

Thought & Action — Mitigation Agent

It seems there was an error retrieving logs for the "hotel-reserv-geo" container, possibly due to a misconfiguration or the service not existing. I will check the pods in the "test-hotel-reservation" namespace to verify their status and identify any issues.

Call NL2Kubectl Tool with {"nl_query": "get pods -n test-hotel-reservation"}

Tool Usage: NL2Kubectl Tool — Mitigation Agent

```
NAME                                      READY   STATUS             RESTARTS         AGE
consul-65844d4555-5vpkm                   1/1     Running            0                40s
frontend-69d95967fb-cgrjc                 1/1     Running            0                8m3s
geo-99d8c58c-5wrbn                        0/1     CrashLoopBackOff   6 (41s ago)      8m3s
jaeger-67445b6dd6-7kbpd                   1/1     Running            0                8m1s
memcached-profile-66bb9c8b9d-86bpf        1/1     Running            0                8m
memcached-rate-68585796cb-gtpg6           1/1     Running            0                7m57s
memcached-reserve-646c7dcd6b-vwz8t        1/1     Running            0                7m52s
mongodb-geo-5ff578bcb8-bg6cr              1/1     Running            0                8m1s
mongodb-profile-758cb77f9f-thlpl          1/1     Running            0                8m
mongodb-rate-56cc8659c9-7dwkw             0/1     Pending            0                7m56s
mongodb-recommendation-55699465f7-f9hng   0/1     Pending            0                7m54s
mongodb-reservation-5994859869-zvvmt      1/1     Running            0                7m50s
mongodb-user-969c6c449-86tnh              0/1     Pending            0                7m48s
profile-676fcc86d5-v7c55                  0/1     Error              6 (3m16s ago)    7m59s
rate-c9bc58c85-klcq7                      0/1     CrashLoopBackOff   6 (25s ago)      7m55s
recommendation-8675c6c9ff-hqbl8           0/1     CrashLoopBackOff   6 (26s ago)      7m54s
reservation-7699b8fdd4-756lh              0/1     CrashLoopBackOff   5 (2m58s ago)    7m50s
search-65df96d8df-6lnw2                   1/1     Running            0                7m49s
user-8477d787d8-d8d45                     0/1     Error              6 (3m1s ago)     7m47s
```

Thought & Action — Mitigation Agent

The "geo" pod is in a CrashLoopBackOff state, and several other pods are pending or experiencing errors. This suggests there may be underlying issues with the application configuration or resource constraints.
I will check the logs of the "geo" pod to identify any specific errors causing the CrashLoopBackOff state.

Call get_logs with {"namespace": "test-hotel-reservation", "service": "geo"}

Tool Usage: get_logs — Mitigation Agent

```
{"level":"info","time":"2025-05-14T01:42:48Z","message":"TLS disabled."}
{"level":"info","time":"2025-05-14T01:42:48Z","message":"Set global log level: info"}
{"level":"info","time":"2025-05-14T01:42:48Z","message":"Tune: setGCPercent to 100"}
2025-05-14T01:42:48Z INF cmd/geo/main.go:23 > Reading config...
2025-05-14T01:42:48Z INF cmd/geo/main.go:36 > Read database URL: mongodb-geo:27017
2025-05-14T01:42:48Z INF cmd/geo/main.go:37 > Initializing DB connection...
2025-05-14T01:43:00Z PNC cmd/geo/db.go:26 > no reachable servers
panic: no reachable servers

goroutine 1 [running]:
github.com/rs/zerolog.(*Logger).Panic.func1({0x96e211, 0x0})
        /go/src/github.com/harlow/go-micro-services/vendor/github.com/rs/zerolog/log.go:405 +0x2d
github.com/rs/zerolog.(*Event).msg(0xc00019e150, {0x96e211, 0x14})
        /go/src/github.com/harlow/go-micro-services/vendor/github.com/rs/zerolog/event.go:158 +0x2b8
github.com/rs/zerolog.(*Event).Msg(...)
        /go/src/github.com/harlow/go-micro-services/vendor/github.com/rs/zerolog/event.go:110
main.initializeDatabase({0xc000028288, 0xc0000fddd0})
        /go/src/github.com/harlow/go-micro-services/cmd/geo/db.go:26 +0x167
main.main()
        /go/src/github.com/harlow/go-micro-services/cmd/geo/main.go:38 +0x877

Please take the next action
```

Task Mitigation Finished — Mitigation Agent

```
Root Cause:
1. The "frontend" service's "recommendation" operation is experiencing connectivity issues with the Consul service,
↪  indicated by the error message "connection refused." The Consul service was restarted to resolve any transient
↪  connectivity issues.
```

```
   2. The "geo" operation in the "search" service is unable to bind the "geo-pvc" due to the persistent volume
↪   controller not recognizing the storage class "geo-storage." The missing storage classes were created to resolve
↪   this issue.
   3. The "geo" pod is in a CrashLoopBackOff state due to a database connection error, specifically "no reachable
↪   servers." This indicates a problem with the MongoDB connection.

   Mitigation Plan:
   1. For the "frontend" service:
      - Ensure the frontend service is correctly configured to connect to the Consul service on port 8500.
      - Verify there are no network policies or firewall rules blocking traffic between the frontend and Consul
      ↪  services.
      - Monitor the Consul service to ensure it remains operational and accepting connections.

   2. For the "geo" operation in the "search" service:
      - Ensure the persistent volume claims are bound correctly now that the storage classes have been created.
      - Investigate the MongoDB connection issue causing the "geo" pod to crash. Verify the MongoDB service is running
      ↪  and accessible at the expected endpoint.

   3. For the pending PVCs:
      - Monitor the provisioner to ensure it completes the task of binding the remaining PVCs.
      - Check for any resource constraints or scheduling issues that may be preventing the PVCs from being bound.

   Conclusion:
   The primary issues have been addressed by restarting the Consul service and creating the missing storage classes.
↪   The remaining tasks involve monitoring the system to ensure the changes take effect and investigating the
↪   MongoDB connection issue for the "geo" pod.
```

Note (not part of the trajectory)

Although not all the pods are ready in the last observation, this doesn't means the agent failed this attempt. The operations it performed are correct, but the system is still in the process of stabilizing. Kubernetes will attempt to restart faulty pods periodically. After a few minutes of waiting, all pods will be ready and the workloads will be able to run without issues.

Phase: Validation & Retry   Run #2

```
Waiting 300.0 seconds for changes to take effect...
...
-----------------------------------------------------------------------
  117 requests in 10.00s, 48.95KB read
Requests/sec:     11.70
Transfer/sec:      4.89KB
```

Phase: AIOpsLab Validation   Run #2

```
== Evaluation ==
Results:
{'success': True}
```