ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks

Saurabh Jha [*1] **Rohan Arora** [*1] **Yuji Watanabe** [*1] **Takumi Yanagawa** [1] **Yinfang Chen** [2] **Jackson Clark** [2] **Bhavya Bhavya** [1] **Mudit Verma** [1] **Harshit Kumar** [1] **Hirokuni Kitahara** [1] **Noah Zheutlin** [1] **Saki Takano** [1] **Divya Pathak** [1] **Felix George** [1] **Xinbo Wu** [2] **Bekir O Turkkan** [1] **Gerard Vanloo** [1] **Michael Nidd** [1] **Ting Dai** [1] **Oishik Chatterjee** [1] **Pranjal Gupta** [1] **Suranjana Samanta** [1] **Pooja Aggarwal** [1] **Rong Lee** [1] **Jae-wook Ahn** [1] **Debanjana Kar** [1] **Amit Paradkar** [1] **Yu Deng** [1] **Pratibha Moogi** [1] **Prateeti Mohapatra** [1] **Naoki Abe** [1] **Chandrasekhar Narayanaswami** [1] **Tianyin Xu** [2] **Lav R. Varshney** [2] **Ruchi Mahindru** [1] **Anca Sailer** [1] **Laura Shwartz** [1] **Daby Sow** [1] **Nicholas C. M. Fuller** [1] **Ruchir Puri** [1]

Abstract

Realizing the vision of using AI agents to automate critical IT tasks depends on the ability to measure and understand effectiveness of proposed solutions. We introduce ITBench, a framework that offers a systematic methodology for benchmarking AI agents to address real-world IT automation tasks. Our initial release targets three key areas: Site Reliability Engineering (SRE), Compliance and Security Operations (CISO), and Financial Operations (FinOps). The design enables AI researchers to understand the challenges and opportunities of AI agents for IT automation with push-button workflows and interpretable metrics. ITBench includes an initial set of 102 real-world scenarios, which can be easily extended by community contributions. Our results show that agents powered by state-of-the-art models resolve only 11.4% of SRE scenarios, 25.2% of CISO scenarios, and 25.8% of FinOps scenarios (excluding anomaly detection). For FinOps-specific anomaly detection (AD) scenarios, AI agents achieve an F1 score of 0.35. We expect ITBench to be a key enabler of AI-driven IT automation that is correct, safe, and fast. ITBench, along with a leaderboard and sample agent implementations, is available at https://github.com/ibm/itbench.

[*]Equal contribution [1]IBM [2]University of Illinois at Urbana-Champaign. Correspondence to: Saurabh Jha <Saurabh.Jha@ibm.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

1. Introduction

Modern IT systems are driving many facets of our economy. They have grown significantly in complexity with the adoption of cloud computing and agile development practices (Harvard Business Review Research Report, 2022; Trask, 2025). Effective management of these systems is becoming extremely challenging as corporations struggle to keep up with this growing complexity. Various IT personas—ranging from Chief Information Officers to Site Reliability Engineers and Security and Compliance officers—and IT engineers in general are struggling to ensure resiliency, reliability, security, and cost effective operations of IT Systems.

The recent CrowdStrike outage highlighted these challenges as it brought down our society's most critical systems—from hospital services to air travel—and was estimated to cost US Fortune 500 companies a staggering $5.4 billion (Kerner, 2024). This incident underlined the critical need for intelligent IT incident resolution, with compliance and risk management capabilities, a topic also addressed in the Digital Operational Resiliency Act (DORA) in Europe (Parliament and the Council of the European Union, 2024).

The rising popularity of AI agents and their projected ability to handle intricate tasks have increased the demand for AI agents managing IT systems (John, 2024; Miguel Carreon, 2024; Pujar et al., 2023). Given the complexity of IT tasks, a major hurdle for this research is establishing systematic methods to assess the effectiveness of AI agents prior to their production deployment (Bogin et al., 2024; Kapoor et al., 2024). Consequently, there is an urgency to develop methods for evaluation of AI agents based on real IT tasks and their corresponding environments.

This paper addresses this critical need and presents ITBench, a first-of-its-kind framework that is both comprehensive and visionary for benchmarking real-life IT automation tasks. The goal of ITBench is to measure the performance of AI

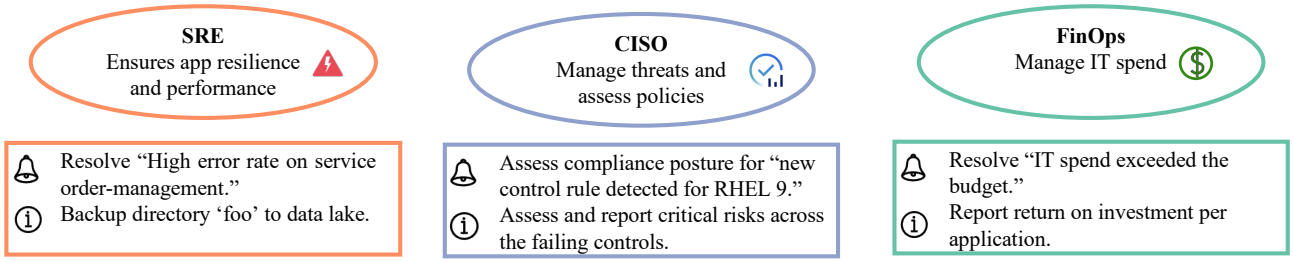


Figure 1: Sample personas and IT tasks. Bell icon represents event-triggered tasks. Information icon represents other tasks such as data analysis, preventive maintenance tasks, or continuous optimization.

agents across a wide variety of complex and real-life IT tasks across personas, including *Site Reliability Engineering (SRE)*, focusing on availability and resiliency; *Compliance and Security Operations (CISO)*, ensuring compliance and security of IT implementations; and *Financial Operations (FinOps)*, enforcing cost efficiencies and optimizing return on investment, among others (as shown in Figure 1).

ITBench aims to advance innovation and establish new standards in the field. Our contributions can be summarized along the following three axes:

- **Reflecting the real world:** ITBench addresses the IT automation requirements that are relevant and prevalent in production settings. SRE scenarios are based on real-world incidents observed in our own SaaS products. CISO scenarios are based on CIS benchmark (for Internet Security, CIS). FinOps scenarios are identified by the FinOps Foundation (Foundation, 2025a) through key business outcomes.
- **Being open and extensible with comprehensive IT coverage:** We view ITBench as a central hub for benchmarking AI-driven solutions across diverse IT automation use cases. To support this, we provide IT benchmark suites and a framework for vertical expansion (i.e., adding more scenarios) and horizontal expansion (i.e., adding more personas), ensuring extensive coverage of IT tasks. ITBench is an open-source framework built with open-source technologies, while allowing organizations with proprietary technologies to use it for developing and benchmarking their solutions.
- **Enabling automated evaluation with partial scoring:** ITBench is designed to provide constructive feedback to drive improvements in the design of agentic solutions for IT problems. It includes a comprehensive evaluation framework and leaderboard that provide feedback to users at various stages of their agents' reasoning process.

ITBench provides push-button deployment and tooling for setting up environment, runtime agent, guardrail engine, as well as authorization and authentication. It allows developers and researchers to build novel solutions for managing complex IT systems. Currently, ITBench addresses reactive problems, including incidents diagnosis and resolution, compliance assessments in regulated environments for new controls, and cost management events. In the future, we plan to expand on benchmark evaluation capabilities and include new benchmarks for additional IT processes. Currently, ITBench comprises an initial set of 102 scenarios spanning across SRE (42), CISO (50), and FinOps (10), with respective successful scenario handling rates of 13.8%, 25.2%, and 25.8% (refer to Section 4).

We believe that, similar to the highly influential SWEBench (Jimenez et al., 2024), our new ITBench framework—which encapsulates and measures the ability of AI agents to automate complex, real-world IT tasks—will spur a comparable acceleration in the performance of real-world IT AI agents.

2. Related Work

ITBench targets a comprehensive set of tasks for a wide range of personas within IT automation. The initial release of ITBench focuses on evaluating scenarios within IT Operations (ITOps). Figure 1 illustrates currently targeted personas and exemplar tasks that they are routinely facing. There is clearly rising interest in developing benchmarks to evaluate AI and ML techniques in ITOps with specific focus on SRE, CISO, and FinOps.

TrainTicket (Zhou et al., 2018) provides 22 scenarios collected through an industrial survey of real-world incidents, using hardcoded faults in the TrainTicket application to focus on fault localization. AIOpsLab (Chen et al., 2024a) provides 10 SRE-focused scenarios (referred to as "problems") utilizing a real environment (system) integration that allows interactive access to text, time series, and tabular data. InsightBench (Sahu et al., 2024) provides 100 scenarios to analyze ticket data using static tabular data and synthetic scenarios. TSB-AD (Liu and Paparrizos, 2024a) focuses on anomaly detection with 40 synthetic scenarios.

CIS-Benchmark (CIS, 2024) provides best practices for securing IT infrastructure. Despite the name of "benchmark," it offers only recommendation policies; it provides no experimental platform. Recently, Cloud Native Compute Foun-

Table 1: Comparison of ITBench with related benchmarks

Benchmark	# of Scenarios	Personas and Tasks	Resolvable	Automated Evaluation	Environment	Leaderboard
ITBench (ours)	102	SRE: Incident Resolution, CISO: Compliance Assessment, FinOps: Cost Management	✓	✓	Real Env.	✓ (verified)
TrainTicket	22	SRE: Incident Diagnosis	✗	✗	Real Env.	✗
AIOpsLab	10	SRE: Incident Resolution	✓	✗	Real Env.	✓ (unverified)
InsightBench	100	Ticket Data Analysis	✗	✗	Synthetic	✗
TSB-AD	40	Anomaly Detection	✗	✓	Synthetic	✗
CIS	1000+	Compliance/Security Focal	✓	✗	n/a (info. only)	✗

[1] **Note:** We are not aware of related benchmarks in the FinOps domain that go beyond scorecards.

dation (CNCF) Sandbox project (OSCAL-compass, 2024) released an SDK to support the translation of the CIS human readable formats into OSCAL (OSCAL, 2024). OSCAL was developed by the National Institute of Standards and Technology for programmatic usage in compliance automation. ITBench CISO automation leverages this technology to assess policy requirements.

FinOps Foundation (Foundation, 2025a) provides benchmarks that compare cloud financial performance across organizations and departments, focusing on KPIs such as resource utilization efficiency, contract coverage, and cost apportionment. These benchmarks help assess cloud efficiency by evaluating internal and external metrics, fostering structured, collaborative approaches to cloud optimization.

While existing benchmarks are valuable resources for specific tasks and use cases, and highlight the critical need for systematic benchmarking, they are limited in reflecting real-world IT problems, covering broad IT landscape, and automating evaluation. These limitations are addressed in ITBench, as shown in Table 1.

3. ITBench

ITBench is a systematic benchmarking framework and runtime environment designed to evaluate AI agents tasked with automating IT operations, incorporating a robust architecture (see Figure 2) comprising the AI Agent, Scenario Specification and Environment, Evaluator, and Leaderboard to facilitate comprehensive performance assessment.

Here, we present a brief overview of the key components: 1) Scenario Specification and Environment, 2) AI Agents, and 3) Leaderboard. More details are in Appendix B.

3.1. Scenario Specification and Environment

The bench incorporates a collection of problems that we call *scenarios*. For example, one of the problems in ITBench is to resolve a "High error rate on service order-management" in a Kubernetes environment. Another example that is relevant for the CISO persona involves assessing the compliance posture for a "new control rule detected for RHEL 9." A fundamental challenge is to emulate such problems in a manageable testbed environment. A scenario environment is an operational testbed in which a specific problem(s) occurs.

A scenario p generally corresponds to a problem to be solved in ITBench. We formalize p as a tuple $< M, E, T, D >$, where the variables are as follows:

Scenario Specification. M represents metadata and deployment descriptors, for each scenario, which is stored in the Scenario Specs database in ITBench (see Figure 2). Exemplar metadata elements per scenario include *scenario_name*, *scenario_description*, *scenario_domain*, *scenario_class*, *scenario_complexity*, and *scenario_groundtruth* (see Table 2), which are defined below:

- *scenario_name* is name given to a scenario. For example, a scenario in ITBench has the name "Recommendation Service Cache."
- *scenario_description* describes the scenario. An example of a description of the scenario is "Recommendation Service in Astronomy Shop has a cache failure."
- *scenario_domain* represents different personas within IT automation—namely "SRE," "CISO," and "FinOps."
- *scenario_class* is used to group similar scenarios, such as "Kyverno-opa," "Kyverno-update',' "CacheFailure," "HighCPU," and "CorruptImage."
- *scenario_complexity* captures the difficulty of a problem and is defined using domain knowledge. Figure 4a shows the breakdown of SRE, CISO, and FinOps scenarios in the bench. Figure 4b, 4c, and 4d shows *scenario_complexity* distribution for SRE, CISO, and FinOps, respectively. SRE scenarios are developed based on real-world incidents observed in our own SaaS products. CISO scenarios are based on CIS benchmark (for Internet Security , CIS). FinOps scenarios were developed based on "Domains" and "Capabilities" identified by the FinOps Foundation (Foundation, 2025a) to describe key business outcomes.
- *scenario_groundtruth* records task-specific outcomes that

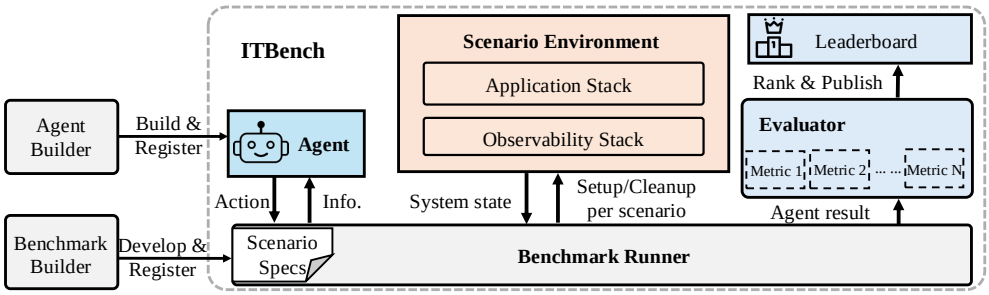


Figure 2: ITBench automation framework.

the Evaluator uses to compare against the agent's expected output. For instance, in incident resolution for SREs, the ground truth for the Diagnosis task includes a list of entities involved in the fault propagation chain, the actual fault propagation chain(s), and fault conditions, while for the Mitigation task, it captures plausible mitigation actions.

Environment. E represents an an operational testbed where the problem occurs. Components within the environment expose APIs to observe and control the environment. When the Agent Builder registers the agent for benchmarking, the Benchmark Runner (see Figure 2) randomly selects a set of scenarios, which may be optionally filtered based on the *agent_type* and *agent_level*. Next, the Benchmark Runner iterates through the set of scenarios, and for each scenario it instantiates a testbed. An example of an environment is a Kubernetes cluster installed with OpenTelemetry Astronomy Shop Demo application (Community, 2024), observability stack including Grafana (gra), Loki (lok), Jaeger (jae), and Prometheus (pro), along with mechanisms that induce problem(s) in the environment.

Triggering Events. T is a set of triggering events that occur due to manifestation of a specific problem in the environment. Tools are configured to observe the environment and raise triggering events on problematic conditions. An example of a triggering event is "High Error Rate on adservice," which may be triggered in the environment due to cache failure problem.

Desired Outcome. D defines the automation objective and represents the ultimate goal. For instance, in case of SRE incident resolution, the ultimate goal is to clear T in the E.

3.2. AI Agents

In IT automation, the different personas are focused on a specific desired outcome, which defines their automation goals. For SREs, incident resolution is the primary objective. Achieving this can involve multiple steps, such as diagnosing an incident, or a single step, like generating a diagnosis report. CISO persona focuses on the regulatory controls posture assessment process, including *Collect evidence* and

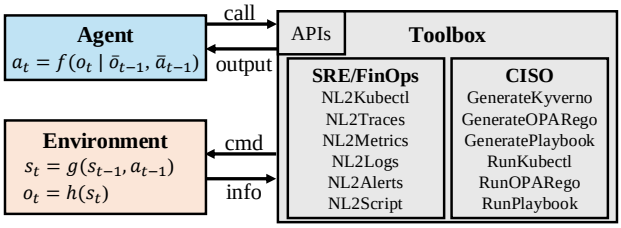


Figure 3: Agent and environment as a POMDP. Agents interact with the environment via the APIs exposed by IT-Bench's toolbox.

Scan assessment posture tasks. FinOps persona focuses on the cost management, where sample tasks include *Identify inefficiency* and *Mitigate inefficiency*. During evaluation, each step (task) is assessed independently and is measured using well defined metrics; see Table 3.

The goal of ITBench is to evaluate AI agents on a broad range of real-world IT automation tasks that are otherwise performed by SREs, FinOps, and CISO personas.

In this paper, an AI *agent* is defined as an autonomous or semi-autonomous software program that uses an LLM to plan, make decisions, interact with the target environment, and execute actions to achieve goals. An AI agent is expected to successfully handle any of the scenarios in the ITBench, by interacting with the environment.

As shown in Figure 3, agent and environment form a Partially Observed Markov Decision Process (POMDP), where the state is the snapshot of the environment. The state transitions are determined by the environment, which are then (partially) observed by the agent.

Given a scenario p instantiated in an environment E, an agent probes the environment via one of the tools and receives an observation $o_t \in \mathcal{O}$, based on which, it decides the next action:

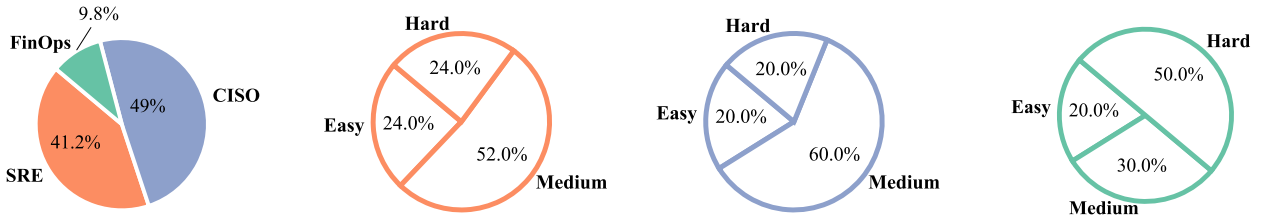
$$a_t = f(o_t|\bar{o}_{t-1}; \bar{a}_{t-1}) \tag{1}$$

Here f is the agent's decision function. $\bar{o}_{t-1}$ is the sequence of observations up to time $t-1$ and $\bar{a}_{t-1}$ is the sequence of actions taken up to $t-1$.

Table 2: Exemplar scenario classes and complexities in ITBench across 102 scenarios.

Scenario Domain	Scenario Class	Scenario Complexity	Technologies
SRE	CacheFailure: Create a memory leak due to an exponentially growing cache	Medium	K8s, Redis, MongoDB
	HighCPU: Trigger high CPU load in target service	Medium	K8, Host, Pods
	CorruptImage: Deployment uses wrong Docker image	Easy	K8s, Image registry
	HTTPRequestBodyTamperFault: Modify HTTP Post request between services	Medium	K8s, ingress/egress
	HTTPRequestAbortFault: Interrupt HTTP connection between services	Medium	K8ss, ingress/egress
	MemoryResourceLimit: Reduce memory limit on target service	Easy	K8s, Host, Pod
CISO	New K8s CIS-benchmarks on Kyverno	Easy	K8, Kyverno
	New K8s CIS-benchmarks on OPA	Medium	K8s, OPA, Kubectl
	New RHEL9 CIS-benchmarks on Ansible-OPA	Medium	RHEL9, OPA, Ansible
	Update K8s CIS-benchmarks on Kyverno	Hard	K8s, Kyverno
FinOps	CostAlertMisconfiguration: Alert threshold is too low, causing false alerts	Easy	K8s, HPA
	AutoscalerMisconfiguration: Horizontal pod autoscaler thresholds are misconfigured, creating excess pods	Hard	K8s, HPA
	Data Insights Generation: Analyze cloud bills and retrieve data based on natural language query	Easy, Medium, Hard	Natural Language to SQL
	Anomaly Detection & Ranking: Identify overspending events in a cloud bill with regard to forecasted spend amounts, and rank anomalies based on user-specified criteria	Hard	Anomaly detection, forecasting, data query from database

[1] Scenario complexity depends on the characteristics of the scenario and is independent from agent capability. See appendixes for details.
[2] K8s refers to Kubernetes (kub). [3] Here, "technologies" denotes the tools/systems a domain expert must understand to perform the task.



(a) Scenarios by persona. (b) SRE scenario complexity. (c) CISO scenario complexity. (d) FinOps scenario complexity.

Figure 4: Characterization of ITBench scenarios.

Initially, o_0 may be a triggering event showing a problematic state s_0 of the environment. Given state s_{t-1} and action a_{t-1}, the environment transitions to the next state:

$$s_t = g(s_{t-1}, a_{t-1}) \tag{2}$$

The observation o_t is determined as a function of the state and is in general a proxy for the environment state s_t, hence the formulation can be thought of as a POMDP:

$$o_t = h(s_t) \tag{3}$$

The set $\mathcal{A}$ of actions is defined as $\mathcal{Q} \bigcup \{\perp\}$, where $\mathcal{Q}$ is the set of tools and $\perp$ represents the "stop action" by the agent. We define t^* as the time when the agent stops:

$$t^* = \min\{t | a_t = \perp\} \tag{4}$$

An agent reflects on the result to guide its next action, continuing until the final goal is achieved. Given a set of scenarios that the agent works on, it targets to maximize the success defined as follows:

$$\mathbb{E}_{p \sim \pi_p}(\mathbb{I}(g(s^p_{t^*}, f(o_{t^*} | \bar{o}_{t^*-1}, \bar{a}_{t^*-1})) = s^p_G)) \tag{5}$$

where $\mathbb{I}$ is an indicator function comparing the terminating state with goal state and π is the distribution of scenarios.

3.2.1. Baseline AI Agents

We developed baseline agents SRE-Agent for SRE, Compliance Assessment Agent for CISO, and FinOps-agent for FinOps. Each of these agents uses state-of-the-art agentic techniques such as ReAct-based planning (Yao et al., 2023), reflection (Shinn et al., 2023), and disaggregation (Xu et al.,

Table 3: Personas, tasks, and metrics in ITBench.

Personas	Tasks	Metrics
SRE	Diagnosis	pass@1, Fault Localization, Fault Propagation Chain, Mean Time to Diagnosis
	Mitigation	pass@1, Mean Time to Repair
CISO	Collect evidence	pass@1
	Scan assessment posture	pass@1, Time to Process
FinOps	Identify inefficiency	pass@1
	Mitigate inefficiency	pass@1, Hourly infra cost, Efficiency
	Data Insights	pass@1, Token utilization
	Anomaly Detection	F1 Score, rank score

2023). Reflection techniques include syntax checking/linting, semantic validation (Xie et al., 2024a), and llm-as-a-judge (Zheng et al., 2023).

We use the open-source CrewAI framework (cre) to create and manage agents. The agents can be configured to use various LLMs either through watsonx, Azure, or vLLM. Each agent is initialized with a prompt that describes its goal, the context, the tasks, and the expected output format. In-context learning examples are included to guide the agent and demonstrate tool usage. Agents use tools to interact with the environment for information gathering.

Logs, traces, and metrics collected during the diagnosis process would overwhelm the context window of any LLM currently available due to large volume of data. Therefore, agents targeting the SRE or FinOps persona are equipped with specialized tools to interact with the environment (refer to Figure 3): 1) NL2Traces to extract trace data in a structured format, 2) NL2Metrics to analyze key system metrics, 3) NL2Logs to parse log data effectively, 4) NL2Kubectl to perform Kubernetes-specific operations, and a summarization tool to condense extensive data into actionable insights. For example, the agent may use the NL2Kubectl tool to "list all of the pods in the default namespace." In turn, the NL2Kubectl tool uses an LLM to transform the utterance into an executable command: "kubectl get pods -n default."

Similarly, the compliance assessment required for new regulations and technologies, with the evidence and diverse policy languages, would be overwhelming if submitted directly to LLMs. The compliance agents designed for CISO compliance assessment automation are equipped with specialized tools. These tools include capabilities to 1) generate policies such as Kyverno or OPA Rego Policy as Code starting from natural language specifications, 2) generate scripts for the collection of evidence, 3) access code repositories such as git to facilitate GitOps workflows for code management, and 4) deploy and execute the generated policies to accomplish the assessment task.

3.3. Leaderboard

ITBench includes a leaderboard to promote reproducibility and comparative analysis, following the AI common task framework (Donoho, 2019; Varshney et al., 2019). The leaderboard offers a predefined, extensible set of performance metrics designed to provide clear insights into agent performance relative to the evaluation criteria.

ITBench devises *scoring methods for partially correct solutions* to provide meaningful feedback for summative assessments. This comprehensive approach establishes a new standard for evaluating and advancing AI-driven solutions in IT automation. For each scenario that an agent works on, upon task completion, the ITBench records the final system state, which is then used at the end of all scenario runs along with the pre-defined ground truth data to validate how well the agent performed across all the scenarios.

We are open-sourcing a small subset (11 out of 102) of scenarios along with the baseline agents to help the community become familiar with ITBench through practical examples. We reserve the remaining scenarios in ITBench to benchmark and evaluate the submitted agentic solutions.

4. Results

4.1. Evaluation Setup

To understand the impact of reasoning and planning capabilities of LLMs on ITBench scenarios, we instantiate our agents using different LLM models, both for natural language reasoning and code generation. Specifically, we employ GPT-4o (checkpoint version 2024-11-20), Llama-3.3-70B-instruct, Llama-3.1-8B-instruct, and Granite-3.1-8B-instruct for tasks that rely on natural language understanding and reasoning. For code-focused use cases, we utilize GPT-4o-mini, Llama-3.1-405b-instruct, and Mixtral-8x7b-instruct. All models use a context window of 128K tokens, enabling them to process more extensive input sequences.

We conduct our experiments primarily on AWS EC2 instances (m4.xlarge), although ITBench can also be readily deployed on a consumer-grade laptop using a pseudo-cluster, thus making it easier to develop AI agents (Appendix C.4.1)

Below, we provide an overview of our baseline agents' performance across ITBench scenarios for SRE, CISO, and FinOps. Our findings indicate that both open-source and proprietary models often struggle with real-world tasks, underscoring the importance of benchmarks that push the limits of reasoning and planning in foundation models. For

Table 4: Evaluation of SRE-Agent on SRE scenarios

Models	Diagnosis				Mitigation	
	pass@1 (%)↑	FL (NTAM)↑	FPC (NTAM)↑	MTTD (s)↓	pass@1 (%)↑	MTTR (s)↓
granite-3.1-8B-instruct	3.57 ± 0.94	0.16 ± 0.02	0.19 ± 0.02	259.92 ± 65.01	0.24 ± 0.25	$845.50 \pm$ —
llama-3.1-8B-instruct	0.99 ± 0.51	0.07 ± 0.01	0.08 ± 0.01	**57.50** ± 2.05	1.98 ± 0.68	**245.13** ± 40.66
llama-3.3-70B-instruct	3.10 ± 0.84	0.16 ± 0.02	0.16 ± 0.02	191.85 ± 31.34	3.33 ± 0.90	776.27 ± 252.87
gpt-4o	**13.81** ± 1.67	**0.39** ± 0.05	**0.34** ± 0.03	72.44 ± 4.71	**11.43** ± 1.52	282.47 ± 30.04

[1] 42 scenarios (21 scenarios with traces and 21 without traces). [2] 10 runs per scenario per model. [3] pass@1 values are shown as percentages. '—' indicates missing data. [4] std error for each metric is listed. [5] **FL (NTAM)** = Normalized topology-aware metric for root cause, **FPC (NTAM)** = Normalized topology-aware metric for fault propagation chain (value between 0 and 1.0), **MTTD** = Mean time to diagnosis (seconds), **MTTR** = Mean time to repair (seconds). **Bold**: the best performance. [6] Details of NTAM are available in Appendix C.6.3

more comprehensive results and detailed scenario-level discussions, please refer to Appendix C (SRE), Appendix D (CISO), and Appendix E (FinOps).

4.2. Overall Results

Table 4, Table 5, and Table 6 show the performance of SRE-agent, CISO-agent, and FinOps-agent respectively.

SRE. We measure the efficiency of SRE-Agent on its ability to diagnose and mitigate production incidents (e.g., "a high error rate on frontend service").

Diagnosis efficiency is measured using pass@1(Chen et al., 2021) (i.e., identifying the cause as mentioned in ground truth), NTAM (Normalized Topology-Aware Metric) for root cause and fault propagation chain, and time to diagnosis.[1] Mitigation efficiency is measured in terms of pass@1 (i.e., whether the alert was cleared) and mean time to repair.

As shown in Table 4, across all SRE scenarios, GPT-4o consistently outperforms the other models, achieving the highest pass@1 scores for diagnosis (13.81%) and mitigation (11.43%), as well as the highest score on NTAM (FL and FPC) metrics. Llama-3.3-70B ranks second overall, trailing GPT-4o on most metrics. The 8B models have lower mitigation success rate. Surprisingly, Granite-3.1-8B (without any specialized finetuning) achieves higher accuracy than Llama-3.3-70B on the diagnosis task.

Removing trace data can drastically reduce success rates (see Table 20 and Table 21 in Appendix). For instance, GPT-4o's pass@1 in diagnosis falls from 13.81% with traces to 9.52% without them, and mitigation plummets to 2.86%. This highlights the critical role of system observability in SRE, which ITBench can evaluate under varying conditions. Because there is no perfect observability in practice, how to guide SRE-agents to collect new observability data and to help SRE-agents reason about failures with incomplete observability is an important but open problem.

[1] NTAM is Normalized topology-aware metric that measures the quality of the predicted root cause and fault propagation chains using a system and application topology. Refer to Appendix C.6.3.

CISO. We measure the efficacy of our agents across the four scenario classes introduced in Table 2. Each *scenario_class* imposes a distinct set of CIS-benchmarks requirements (e.g., "minimize the admission of containers wishing to share the host network namespace"), has a specific level of complexity (e.g., Easy, Medium, or Hard), and generates scenario-specific code artifacts.

The efficacy of CISO-agents is measured based on the ability to detect artifact misconfigurations (aka non-compliance, e.g., no minimum count of containers sharing namespace, or the count is above the threshold), or confirm proper configurations (aka compliance), within the varied environments of the scenario classes randomly injected with misconfigurations. Notably, GPT-based models dominate on both pass@1 and Time to Process metrics. The pass@1 is nearly two times better than second-best performing model, while the TTP shows a handling of the scenarios in the minimal time across our scenario classes.

FinOps. In addition to the standard event-driven scenarios, ITBench was extended to support non-alert-driven scenarios for the FinOps persona, demonstrating its extensibility. In particular, we added data insights and anomaly detection scenarios to ITBench. Table 6 presents our results in all FinOps usecases. We report pass@1 score for data insights, diagnosis, and mitigation tasks, and F1 score and rank score for anomaly detection. F1 score measures the precision and recall abilities of the agent to identify anomalous costs with regard to the ground truth. The rank score measures the relative ranking of the anomalies as determined by the agent with regard to the ground truth ranking. GPT-4o consistently outperforms all other models, achieving a 33% pass rate for diagnosing the origin of the cost increase alert, 29% accuracy in data insights scenarios, and 0.6 F1 score in anomaly detection. Refer to Appendix E.5 for futher details.

4.3. Impact of Scenario Complexity

SRE. We categorize scenarios as Easy, Medium, or Hard based on factors such as fault propagation chain length, number of resolution steps, and the diversity of technolo-

Table 5: Evaluation of CISO Compliance Assessment Agent on CISO scenarios

Models	Scenario pass@1 (%) ↑				O/A pass@1 (%) ↑	TTP (s) ↓
	kyverno	k8s-opa	rhel-opa	kyverno-update		
granite-3.1-8B-instruct	7.84 ± 3.84	0.00 ± 0.00	0.00 ± 0.00	1.59 ± 1.58	1.71 ± 0.76	197.03 ± 2.52
mixtral-8x7B-instruct	7.35 ± 3.19	1.43 ± 1.42	0.00 ± 0.00	1.29 ± 4.34	3.94 ± 1.03	120.63 ± 3.77
llama-3.1-8B-instruct	8.57 ± 3.37	0.00 ± 0.00	0.00 ± 0.00	7.46 ± 3.23	3.59 ± 1.07	121.49 ± 3.00
llama-3.3-70B-instruct	18.46 ± 4.94	0.00 ± 0.00	1.43 ± 2.88	8.06 ± 3.50	9.32 ± 1.67	189.61 ± 2.71
mistral-large-2	6.56 ± 3.20	22.73 ± 5.32	7.23 ± 2.88	10.45 ± 3.77	11.55 ± 1.95	167.98 ± 3.42
llama-3.1-405B-instruct	16.22 ± 4.32	20.83 ± 4.86	8.75 ± 3.26	3.17 ± 2.22	12.46 ± 1.98	178.89 ± 3.37
gpt-4o-mini	16.18 ± 4.54	**43.10** ± 6.99	**30.38** ± 5.43	9.43 ± 4.08	**25.19** ± 2.80	102.40 ± 3.70
gpt-4o	**40.28** ± 5.99	39.34 ± 6.55	7.61 ± 2.81	**17.74** ± 4.92	24.74 ± 2.64	**101.29** ± 3.81

[1] 50 scenarios. [2] 8 runs per scenario per model. [3] pass@1 values are shown as percentages. [4] TTP Time to process (seconds).
[5] **kyverno** = New K8s CIS-benchmarks on Kyverno, easy scenario class; **k8s-opa** = New K8s CIS-benchmarks on OPA, medium scenario class; **rhel-opa** = New RHEL9 CIS-benchmarks on Ansible-OPA, medium scenario class; **kyverno-update** = Update K8s CIS-benchmarks on Kyverno, hard scenario class.

Table 6: Evaluation of FinOpsAgent on FinOps scenarios.

Models	Non-Alert-Driven Scenarios			Alert-Driven Scenarios	
	Data Insights	Anomaly Detection		Diagnosis	Mitigation
	pass@1 ↑	F1 Score ↑	Ranking ↑	pass@1 (%) ↑	pass@1 (%) ↑
granite-3.1-8B-instruct	14	0.4 ± 0.07	0.3 ± 0.00	0	0
llama-3.1-8B-instruct	0	0.4 ± 0.03	0.4 ± 0.00	0	0
llama-3.3-70B-instruct	**29**	0.0	0.0 ± 0.00	16.6	0
gpt-4o	**29**	**0.6** ± 0.00	**0.5** ± 0.00	**33**	0

pass@1 values are shown as percentages. The Data-Insight evaluations exhibit zero variance because we use a fixed dataset and configure the agent's model with temperature zero. Diagnosis and Mitigation evaluations include only two scenarios, making variance negligible.

gies involved, as described in Equation (6). Our results show that success rates (pass@1) clearly decline as the *scenario_complexity* increases. Even the best performing model, GPT-4o, diagnosed only 36%, 7.73%, 5% of Easy/Medium/Hard cases (Table 18) and mitigated just 21%, 12.27%, 0% (Table 19). None of the models could mitigate the Hard scenarios, even though over 50% of Easy scenarios were mitigated. Notably, GPT-4o is the only model that successfully diagnosed multiple Hard scenarios.

CISO. The complexity of the CISO scenarios is directly mapped to scenario classes. For example, *scenario_complexity* of Kyverno scenarios is Easy, *scenario_complexity* of k8s-opa and rhel-opa is Medium, while *scenario_complexity* of Kyverno-update scenarios is Hard. All models struggle, as expected, as the difficulty of the scenarios increases from the Easy *kyverno* class to the Hard *kyverno-upadate* class.

FinOps. Currently, ITBench includes 2 Easy, 3 Medium, and 5 Hard scenarios. None of the models were able to resolve the Hard scenarios. GPT-4o performs better in anomaly detection and alert-driven scenarios, while the LLaMA-3.3-70B-Instruct model achieves comparable performance to GPT-4o in data insight scenarios.

5. A Case Study on SRE-Agent Failures

Understanding the decision process of LLM-based agents is challenging due to the complexity of agentic systems but is feasible through detailed agent trajectory logging and a structured prompting framework. We log each input and output for planning agents and tools, including the ReAct-style "Thought" step, enabling us to distinguish between high-level reasoning errors (e.g., flawed strategy) and low-level tool errors (e.g., malformed commands), enabling practical analysis and guiding design improvements.

5.1. Analyzing Lower-Level Tool Calling and Execution

Figure 5 shows tool usage and failure types across models. NL2Kubectl dominates usage, suggesting overreliance. Encouraging the agent to make a more balanced use of other tools, such as NL2Traces, could be useful, especially when kubectl commands alone are insufficient. Smaller models (e.g., granite-3.1-8B-instruct, llama-3.1-8B-instruct) show more invalid tool calls, syntax errors, and repeated invocations, indicating lower accuracy and efficiency.

5.2. Quantitative Analysis of High-Level Reasoning

We quantify reasoning by aligning each exploration path with the ground-truth fault-propagation chain. An effective agent is expected to focus its exploration around this chain, while significant deviations may signal reasoning

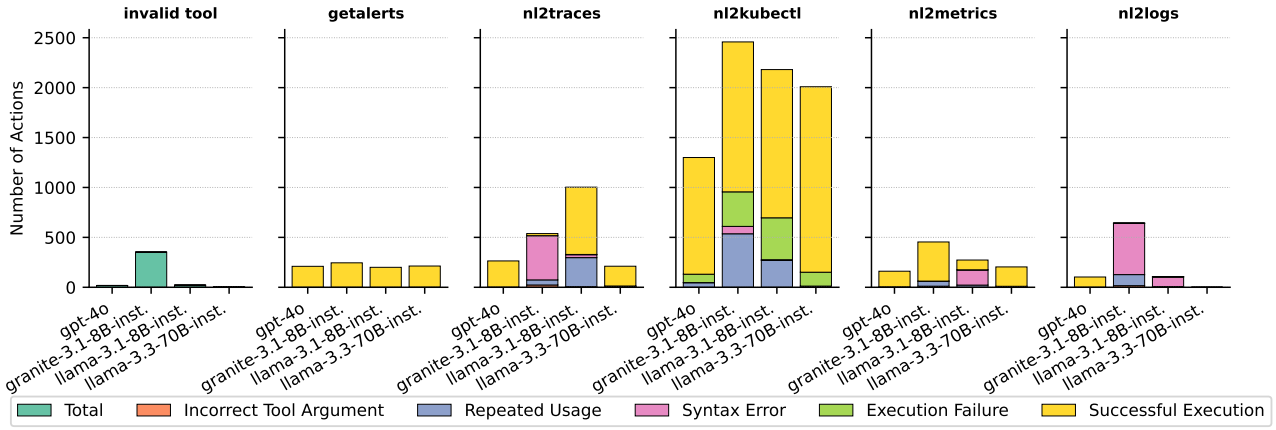
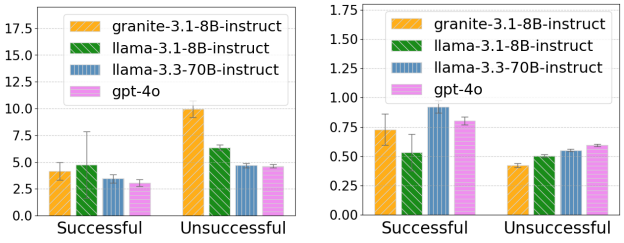


Figure 5: SRE-Agent Tool Usage Distribution

flaws. Based on this insight, we introduce two evaluation metrics: (i) **Detoured Services**: $|V_{\text{visited}} \setminus V_{\text{gt}}|$, the number of visited services that are *not* on the ground-truth chain (smaller $\rightarrow$ more focused search); and (ii) **Relative Covered Services**: $\frac{|V_{\text{visited}} \cap V_{\text{gt}}|}{|V_{\text{gt}}|}$, the fraction of ground-truth services visited (closer to 1 $\rightarrow$ better coverage).

As shown in Figure 6, successful diagnosis trajectories show fewer detours and higher coverage than unsuccessful ones, validating the metrics. Among successful trajectories, GPT-4o shows detours (Kolmogorov–Smirnov p-value $\geq$0.123) and coverage (p-value$\geq$0.089) that are comparable to other models (i.e., Granite-3.1-8B-Instruct, Llama-3.1-8B-Instruct, and Llama-3.3-70B-Instruct), while achieving higher coverage than Llama-3.1-8B-Instruct (p-value=0.024). This suggests that successful agents tend to follow similar reasoning patterns. For unsuccessful trajectories, GPT-4o significantly surpasses all baselines, showing both fewer detours (p-value$\leq$0.001) and greater coverage (p-value$\leq$ 0.011). These results underscore ITBench's utility in revealing insightful patterns in agent reasoning and overall performance.



(a) Detoured Services (b) Relative Covered Services

Figure 6: Quantitative Analysis of High-Level Reasoning

6. Discussion and Conclusion

We presented ITBench, the first framework and experimental platform to benchmark AI Agents for IT automation tasks. ITBench strives to capture the complexity of modern IT systems and the diversity of IT tasks. The reproducibility of ITBench ensures the community-driven effort despite inherent nondeterminism of large-scale IT systems.

One of the key design principles of ITBench is ensuring its flexibility to support diverse areas of different IT systems and its extensibility to new scenarios. While the current scope of ITBench is comprehensive and representative, we plan to further enrich the benchmark suites by adding other important processes essential to modern IT automation. Furthermore, we plan to expand our benchmarking beyond event-triggered scenarios. We are actively working to expand scenario coverage for the supported processes and promote growth through open-community contributions. We invite the community to reproduce their real-world-inspired incidents in a synthetic sandboxed environment leveraging the ITBench. We expect that everyone contributing can bring their expertise to the table.

We expect ITBench to drive the innovations of AI agent-based techniques with a direct impact on the safety, efficiency, and intelligence of today's IT infrastructures. With ITBench, we are starting to explore many deep, exciting open problems: How to develop domain-specific AI agents that specialize in certain types of IT tasks? How to orchestrate multiple agents with various expertise to collaborate on bigger projects? How can we ensure safety of agent-driven solutions? How can we effectively use human-in-the-loop while developing diverse adaptive agents? We invite everyone to participate in answering these questions and realizing the vision of using AI agents to automate critical IT tasks.

Impact Statement

Ethics & Broader Impacts

This research presents a novel benchmarking framework to measure the performance of AI agents across a wide variety of complex and real-life IT tasks, which has the potential to be a key enabler for AI-driven IT automation that is correct, safe, and fast. While the primary focus is on advancing the field of machine learning, as this effort is an open source framework built with open source technologies, it allows organizations with proprietary technologies to use it for developing and benchmarking their solutions more effectively. It also encourages mindsharing in the community and lowers the barrier to innovate in IT domain.

Agents that interact with the system pose several risks. We identify three main risks that could arise when building and using a ITBench and associated agents, then discuss how we incorporates measures that mitigate such problems.

First is the security risks that come with executing LM-generated code/commands on the system. Examples include executing commands like `kubectl delete node` and `rm -rf asset/`. To defend against this, we containerize the agents and also provide a self-contained Kubernetes environment to create various scenarios.

Second, if the wider community develops an interest in ITBench and associated agents and builds upon it, it is also possible that illegitimate evaluation datasets or infrastructure can be used to inject testing devices with malicious code or instructions to generate malicious code. For instance, an unofficial repository claiming to host an inference/evaluation harness for ITBench and associated agents could include a task instance with an issue description that tells the LM agent to build key logging functionality and store it in a hidden folder. To eliminate confusion and reduce the possibility of such an event, we provide clear guidelines listed on our GitHub repositories, data stores, and websites indicating the official repositories and channels that we actively maintain. We also encourage third parties to incorporate any improvements into our codebase and help with integrating such contributions.

Lastly are the consequences of ITBench agents being deployed in the real world. Prior works have conceptualized and put forth prototypes of agents that can carry out offensive security measures. It is also not difficult to imagine that a system like SRE-Agent can be incorporated into pipelines, resulting in the production of malicious code and libraries. The strong performance of agents on ITBench implies that future AI systems will likely be increasingly adept in the aforementioned use cases. Releasing ITBench agents as open source agents can support research toward designing sound, effective constraints for what software engineering agents are permitted to do. It can also serve as a system that legal experts and policy-making entities can experiment with to shape the future of what AI-driven end-to-end software engineering could look like.

Reproducibility

To help the greater community reproduce the results presented in this paper and build on the ITBench, we open source all of our resources that were created for this project. The source code for the interactive pipeline, context management logic, command implementations, interface design, and everything else is entirely available in a GitHub repository. We provide extensive text and video documentation describing how to run and modify different parts of the codebase. Practitioners should be able to easily recover our findings by running the agent with simple scripts. The results presented in the main and supplementary parts of this paper can be fully obtained by following instructions in the repositories. Finally, we also maintain an active online help forum to assist with any reproduction problems or questions about how to build on ITBench.

Acknowledgements

We would like to thank everyone at IBM and the University of Illinois at Urbana-Champaign who supported this project but are not on the author list. In particular, we are grateful to Ravishankar K. Iyer and Deming Chen for their guidance and encouragement, as well as to all who shared their excitement, provided feedback on early prototypes, and collaborated with the core team across various aspects of this work. We gratefully acknowledge the core contributors—Pavankumar Murali, Paulina Toro Isaza, and Xi Yang—for their decisive role in addressing reviewer feedback. Pavankumar designed the FinOps anomaly-detection scenarios, while Paulina and Xi carried out the agent-trajectory analyses. The work is supported in part by the IBM-Illinois Discovery Accelerator Institute (IIDAI) and National Science Foundation (NSF) CNS-2145295.

In addition, we acknowledge the support of our colleagues in IBM Instana: Ameet Rahane, Marc Palaci-Olgun, Guangya Liu, Brad Blancett, Chad Holliday, Arthur De Magalhaes, Ragu Kattinakere, Chris Bailey, Isabell Sippili, and Danilo Florissi; IBM Granite and Data Model Factory: Hui Wu and Bing Zhang; IBM Emerging Technology Engineering: Carlos Fonseca, Aditya Gidh, Mike Sava, Bill Rippon, and Danny Barnett; IBM UX Research: James Sutton, Connor Leech, and Justin McNair; IBM Research: Michal Shmueli-Scheuer, Lilach Edelstein, and Roy Bar-Haim.

References

CISO Compliance Assessment Agent. https://github.com/IBM/itbench-ciso-caa-

agent?tab=readme-ov-file, a. Accessed: 2025-01-30.

CISO Compliance Assessment Agent. https://github.com/IBM/itbench-sample-scenarios/tree/main/ciso, b. Accessed: 2025-01-30.

Ansible. https://www.redhat.com/en/technologies/management/ansible, a. Accessed: 2025-01-30.

OPA Gatekeeper. https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes/, b. Accessed: 2025-01-30.

Kyverno, Policy as Code, Simplified! https://kyverno.io, c. Accessed: 2025-01-30.

Open Policy Agent. https://github.com/open-policy-agent/opa, d. Accessed: 2025-01-30.

NIST Cybersecurity Framework 2.0 Quick-Start Guide for Creating and Using Organizational Profiles. https://www.nist.gov/cyberframework. Accessed: 2025-01-30.

Crew ai. https://www.crewai.com/. Accessed: 2025-01-30.

Datadog. https://www.datadoghq.com/. Accessed: 2024-11-30.

Dynatrace. https://www.dynatrace.com/. Accessed: 2024-11-30.

Finops sample data. https://github.com/FinOps-Open-Cost-and-Usage-Spec/FOCUS-Sample-Data. Accessed: 2025-05-30.

Grafana. https://grafana.com/. Accessed: 2025-01-30.

Instana. https://www.ibm.com/products/instana. Accessed: 2024-11-30.

Jaeger. https://www.jaegertracing.io/. Accessed: 2025-01-30.

Kubernetes. https://kubernetes.io/. Accessed: 2025-01-30.

Grafana Loki. https://github.com/grafana/loki. Accessed: 2025-01-30.

Prometheus. https://prometheus.io/. Accessed: 2025-01-30.

T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1737–1749, 2023a. doi: 10.1109/ICSE48619.2023.00149.

T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1737–1749. IEEE, 2023b.

B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 440–453, 2016.

S. Ashok, V. Harsh, B. Godfrey, R. Mittal, S. Parthasarathy, and L. Shwartz. Traceweaver: Distributed request tracing for microservices without application modification. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 828–842, 2024.

A. Avizienis, J.-C. Laprie, and B. Randell. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(1):1–23, Jan. 2004.

F. Bagehorn, J. Rios, S. Jha, R. Filepp, L. Shwartz, N. Abe, and X. Yang. A fault injection platform for learning aiops models. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.

L. A. Barroso, U. Hölzle, and P. Ranganathan. *The Datacenter as a Computer: Designing Warehouse-Scale Machines*. Morgan and Claypool Publishers, 3 edition, 2018.

B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne. *Site Reliability Workbook: Practical Ways to Implement SRE*. O'Reilly Media Inc., Aug. 2018.

B. Bogin, K. Yang, S. Gupta, K. Richardson, E. Bransom, P. Clark, A. Sabharwal, and T. Khot. Super: Evaluating agents on setting up and executing tasks from research repositories. *ArXiv*, abs/2409.07440, 2024.

L. Boisvert, M. Thakkar, M. Gasse, M. Caccia, T. L. S. D. Chezelles, Q. Cappart, N. Chapados, A. Lacoste, and A. Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks, 2024.

C. Boulton. Rising cloud costs leave CIOs seeking ways to cope. https://www.cio.com/article/3496509/rising-cloud-costs-

leave-cios-seeking-ways-to-cope.html. Accessed: 2025-01-30.

K. Budhathoki, L. Minorics, P. Blöbaum, and D. Janzing. Causal structure-based root cause analysis of outliers. In *International Conference on Machine Learning*, pages 2357–2369. PMLR, 2022.

B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5):50–57, May 2016.

S. Chakraborty, S. Garg, S. Agarwal, A. Chauhan, and S. K. Saini. Causil: Causal graph for instance level microservice data. In *Proceedings of the ACM Web Conference 2023*, pages 2905–2915, 2023.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, and J. K. et. al. Evaluating large language models trained on code, 2021.

Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, et al. Empowering practical root cause analysis by large language models for cloud incidents. *arXiv preprint arXiv:2305.15778*, 2023.

Y. Chen, M. Shetty, G. Somashekar, M. Ma, Y. Simmhan, J. Mace, C. Bansal, R. Wang, and S. Rajmohan. Aiopslab: A holistic framework for evaluating ai agents for enabling autonomous cloud. November 2024a.

Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the 19th European Conference on Computer Systems (EuroSys'24)*, Apr. 2024b.

Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, et al. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 674–688, 2024c.

CIS. Center for Internet Security Benchmarks List. https://www.cisecurity.org/cis-benchmarks, 2024. Accessed: 2025-01-30.

O. Community. Opentelemetry astronomy shop demo. https://opentelemetry.io/docs/demo/, 2024. Accessed: 2025-01-30.

J. Dean. Designs, Lessons and Advice from Building Large Distributed Systems. In *Proceedings of the the 3rd Large Scale Distributed Systems and Middleware (LADIS'09)*, Oct. 2009.

C. Di Martino, F. Baccanico, J. Fullop, W. Kramer, Z. Kalbarczyk, and R. Iyer. Lessons Learned From the Analysis of System Failures at Petascale: The Case of Blue Waters. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*, 2014.

Y. Diao, D. Horn, A. Kipf, O. Shchur, I. Benito, W. Dong, D. Pagano, P. Pfeil, V. Nathan, M. Narayanaswamy, and T. Kraska. Forecasting algorithms for intelligent resource scaling: An experimental analysis. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pages 126–143, 2024.

D. Donoho. Comments on Michael Jordan's essay "the AI revolution hasn't happened yet". *Harvard Data Science Review*, June 2019.

A. Drouin, M. Gasse, M. Caccia, I. H. Laradji, M. D. Verme, T. Marty, L. Boisvert, M. Thakkar, Q. Cappart, D. Vazquez, N. Chapados, and A. Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024.

H. Fang, T. Tao, and C. Zhai. Diagnostic evaluation of information retrieval models. *ACM Transactions on Information Systems (TOIS)*, 29(2):1–42, 2011.

Y. Fangkai, L. Wang, Z. Xu, J. Zhang, L. Li, B. Qiao, C. Couturier, C. Bansal, S. Ram, Z. Ma, I. Goiri, E. Cortez, T. Yang, V. Ruehle, S. Rajmohan, Q. Lin, and D. Zhang. Snape: Reliable and low-cost computing with mixture of spot and on-demand vms. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 631–643, 2023.

B. Feng, Z. Ding, and C. Jiang. Fast: A forecasting model with adaptive sliding window and time locality integration for dynamic cloud workloads. In *IEEE Transactions on Services Computing*, pages 1184–1197, 2023.

R. Fonseca, G. Porter, R. H. Katz, and S. Shenker. {X-Trace}: A pervasive network tracing framework. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, 2007.

C. for Internet Security (CIS). *CIS Benchmarks*. Center for Internet Security, 2024. Technical guidelines for secure system configurations.

D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in Globally Distributed Storage Systems. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, Oct. 2010.

F. Foundation. Using efficiency metrics to evaluate cloud optimization and value between parts of the organization or against industry peers to inform decision-making and align finops with business objectives. https://www.finops.org/framework/capabilities/benchmarking/, 2025a. Accessed: 2025-01-30.

F. Foundation. Finops kpis. https://www.finops.org/wg/finops-kpis/, 2025b. Accessed: 2025-01-30.

J. Gao, N. Yaseen, R. MacDavid, F. V. Frujeri, V. Liu, R. Bianchini, R. Aditya, X. Wang, H. Lee, D. Maltz, et al. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 253–269, 2020.

S. Ghosh, M. Shetty, C. Bansal, and S. Nath. How to fight production incidents?: an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC'22)*, Nov. 2022.

H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. 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.

H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC'16)*, Oct. 2016.

H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*, Feb. 2018.

C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. *SIGCOMM Comput. Commun. Rev.*, 45(4):139–152, Aug. 2015. ISSN 0146-4833. doi: 10.1145/2829988.2787496.

Harvard Business Review Research Report. Taming it complexity through effective strategies and partnerships. https://hbr.org/sponsored/2022/11/taming-it-complexity-through-effective-strategies-and-partnerships, 2022.

P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat. Cores that don't count. In *Proceedings of the 18th Workshop on Hot Topics in Operating Systems (HotOS'21)*, June 2021.

J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

L. Huang, M. Magnusson, A. B. Muralikrishna, S. Estyak, R. Isaacs, A. Aghayev, T. Zhu, and A. Charapko. Metastable Failures in the Wild. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*, July 2022.

IDC. Storm clouds ahead: Missed expectations in cloud computing. https://blogs.idc.com/2024/10/28/storm-clouds-ahead-missed-expectations-in-cloud-computing/, 2024. Blog post, published on 28 October 2024.

A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems*, 35:31158–31170, 2022.

S. Jha, S. Cui, S. S. Banerjee, T. Xu, J. Enos, M. Showerman, Z. T. Kalbarczyk, and R. K. Iyer. Live forensics for hpc systems: A case study on distributed storage systems. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

A. S. John. Idc predicts 80% of cios to leverage ai and automation for business agility and insights by 2028, 2024.

E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. M. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson. Cloud Programming Simplified: A Berkeley View on Serverless Computing. Technical Report UCB/EECS-2019-3, University of California at Berkeley, Feb. 2019.

S. Kapoor, B. Stroebl, Z. S. Siegel, N. Nadgir, and A. Narayanan. Ai agents that matter. https://arxiv.org/abs/2407.01502, 2024.

M. G. Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

S. Kendrick. What Takes Us Down? *USENIX ;login:*, 37 (5):37–45, Oct. 2012.

S. M. Kerner. Crowdstrike outage explained: What caused it and what's next. https://www.techtarget.com/whatis/feature/Explaining-the-largest-IT-outage-in-history-and-whats-next#:~:text=What%20might%20be%20considered%20the,Fortune%20500%20companies%20%245.4%20billion., 2024.

J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. C. Lim, P.-Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, and D. Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks, 2024.

J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish. Detecting failures in distributed systems with the falcon spy network. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 279–294, 2011.

H. Liu, S. Lu, M. Musuvathi, and S. Nath. What bugs cause production cloud incidents? In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS'19)*, May 2019.

M. Liu, L. Pan, and S. Liu. Cost optimization for cloud storage from user perspectives: Recent advances, taxonomy, and survey. *ACM Computing Surveys*, 55(13s):1–37, 2023a.

Q. Liu and J. Paparrizos. The elephant in the room: Towards a reliable time-series anomaly detection benchmark. In *NeurIPS 2024*, 2024a.

Q. Liu and J. Paparrizos. The elephant in the room: Towards a reliable time-series anomaly detection benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024b.

Z. Liu, C. Benge, and S. Jiang. Ticket-bert: Labeling incident management tickets with language models. *arXiv preprint arXiv:2307.00108*, 2023b.

L. Ma, T. He, A. Swami, D. Towsley, K. K. Leung, and J. Lowe. Node failure localization via network tomography. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 195–208, 2014.

B. Maurer. Fail at Scale: Reliability in the Face of Rapid Change. *Communications of the ACM*, 58(11):44–49, Nov. 2015.

T. Melissaris, K. Nabar, R. Radut, S. Rehmtulla, A. Shi, S. Chandrashekar, and I. Papapanagiotou. Elastic Cloud Services: Scaling Snowflake's Control Plane. In *Proceedings of the 13th ACM Symposium on Cloud Computing (SOCC'22)*, Nov. 2022.

Microsoft and contributors. Dowhy: A python library for causal inference. https://www.pywhy.org/dowhy/v0.12/, 2023. Version 0.12.

M. D. L. C. Miguel Carreon. Idc: 80

N. R. Murphy, B. Beyer, C. Jones, and J. Petoff. *Site Reliability Engineering: Monitoring Distributed Systems*. O'Reilly Media, 2024. Accessed: 2024-11-07.

NIST 800-53. NIST Special Publication 800-53 Revision 5. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf, 2020.

A. Nodari, J. Nurminen, and C. Fruhwirth. Inventory theory applied to cost optimization in cloud computing. In *Proceedings of the 31st annual ACM symposium on applied computing*, pages 470–473, 2016.

N. OSCAL. Open security controls assessment language. https://pages.nist.gov/OSCAL/, 2024. Accessed: 2025-01-30.

OSCAL-compass. Oscal-compass cncf sandbox project. https://github.com/oscal-compass, 2024. Accessed: 2025-01-30.

P. Osypanka and P. Nawrocki. Resource usage cost optimization in cloud computing using machine learning. In *IEEE Transactions on Cloud Computing*, pages 2079–2089, 2020.

L. Pan, M. Saxon, W. Xu, D. Nathani, X. Wang, and W. Y. Wang. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*, 2023.

N. Papanikolaou, S. Pearson, and M. C. Mont. Towards Natural-Language Understanding and Automated Enforcement of Privacy Rules and Regulations in the Cloud: Survey and Bibliography. In *Secure and Trust Computing, Data Management, and Applications*, 2011.

E. Parliament and the Council of the European Union. Digital operational resilience act for the financial sector and amending regulations. https://eur-lex.europa.eu/EN/legal-content/summary/digital-

operational-resilience-for-the-financial-sector.html, 2024.

D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. Technical Report UCB//CSD-02-1175, University of California Berkeley, Mar. 2002.

L. Pham, H. Ha, and H. Zhang. Root cause analysis for microservice system based on causal inference: How far are we? In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 706–715, 2024.

S. Pujar, L. Buratti, X. Guo, N. Dupuis, B. Lewis, S. Suneja, A. Sood, G. Nalawade, M. Jones, A. Morari, and R. Puri. Automated code generation for information technology tasks in yaml through large language models. https://arxiv.org/abs/2305.02783, 2023.

B. Qiao, F. Yang, C. Luo, Y. Wang, J. Li, Q. Lin, H. Zhang, M. Datta, A. Zhou, T. Moscibroda, S. Rajmohan, and D. Zhang. Intelligent container reallocation at microsoft 365. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1438–1443, 2021.

G. Quattrocchi, E. Incerto, R. Pinciroli, C. Trubiani, and L. Baresi. Autoscaling solutions for cloud applications under dynamic workloads. In *IEEE Transactions on Services Computing*, pages 804–820, 2024.

D. Roy, X. Zhang, R. Bhave, C. Bansal, P. Las-Casas, R. Fonseca, and S. Rajmohan. Exploring llm-based agents for root cause analysis. https://arxiv.org/abs/2403.04123, 2024.

G. Sahu, A. Puri, J. Rodriguez, A. Abaskohi, M. Chegini, A. Drouin, P. Taslakian, V. Zantedeschi, A. Lacoste, D. Vazquez, et al. Insightbench: Evaluating business analytics agents through multi-step insight generation. *arXiv preprint arXiv:2407.06423*, 2024.

Salesforce. Pyrca: A python machine learning library for root cause analysis. https://github.com/salesforce/PyRCA, 2023. Version 1.0.1.

N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. 2010.

J. Storment and M. Fuller. *Cloud FinOps*. O'Reilly Media, 2nd edition, 2023. Chapter 1.

X. Sun, R. Cheng, J. Chen, E. Ang, O. Legunsen, and T. Xu. Testing Configuration Changes in Context to Prevent Production Failures. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, Nov. 2020.

C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang. {NetBouncer}: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 599–614, 2019.

C. Tang, T. Kooburat, P. Venkatachalam, A. Chander, Z. Wen, A. Narayanan, P. Dowell, and R. Karl. Holistic Configuration Management at Facebook. In *Proceedings of the 25th ACM Symposium on Operating System Principles (SOSP'15)*, Oct. 2015.

C. Tang, K. Yu, K. Veeraraghavan, J. Kaldor, S. Michelson, T. Kooburat, A. Anbudurai, M. Clark, K. Gogia, L. Cheng, B. Christensen, A. Gartrell, M. Khutornenko, S. Kulkarni, M. Pawlowski, T. Pelkonen, A. Rodrigues, R. Tibrewal, V. Venkatesan, and P. Zhang. 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.

L. Tang, C. Bhandari, Y. Zhang, A. Karanika, S. Ji, I. Gupta, and T. Xu. Fail through the Cracks: Cross-System Interaction Failures in Modern Cloud Systems. In *Proceedings of the 18th European Conference on Computer Systems (EuroSys'23)*, May 2023.

M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, 2006.

S. Trask. State of FinOps: 2025 report. https://data.finops.org/, 2025.

H. Tupsamudre, A. Kumar, V. Agarwal, N. Gupta, and S. Mondal. AI-Assisted Controls Change Management for Cybersecurity in the Cloud. In *Thirty-Fourth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-22)*, 2022.

L. R. Varshney, N. S. Keskar, and R. Socher. Pretrained AI models: Performativity, mobility, and change. arXiv:1909.03290 [cs.CY]., Sept. 2019.

K. Veeraraghavan, J. Meza, S. Michelson, S. Panneerselvam, A. Gyori, D. Chou, S. Margulis, D. Obenshain, S. Padmanabha, A. Shah, Y. J. Song, and T. Xu. 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.

Q. Wang, J. Rios, S. Jha, K. Shanmugam, F. Bagehorn, X. Yang, R. Filepp, N. Abe, and L. Shwartz. Fault injection based interventional causal learning for distributed applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 15738–15744, 2023.

J. Xie, K. Zhang, J. Chen, T. Zhu, R. Lou, Y. Tian, Y. Xiao, and Y. Su. Travelplanner: A benchmark for real-world planning with language agents. https://arxiv.org/abs/2402.01622, 2024a.

Z. Xie, Y. Zheng, L. Ottens, K. Zhang, C. Kozyrakis, and J. Mace. Cloud atlas: Efficient fault localization for cloud systems using language models and causal insight. https://arxiv.org/abs/2407.08694, 2024b.

B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. https://arxiv.org/abs/2305.18323, 2023.

T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy. Do Not Blame Users for Misconfigurations. In *Proceedings of the 24th Symposium on Operating System Principles (SOSP'13)*, Farmington, PA, Nov. 2013.

J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024a.

X. Yang, R. Arora, S. Jha, C. Narayanaswami, C. Lam, J. Leichter, Y. Deng, and D. Sow. Optimizing it finops and sustainability through unsupervised workload characterization. In *Proceedings of the AAAI Conference on Artificial Intelligence.*, pages 22990–22996, 2024b.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Z. Yao, C. Pei, W. Chen, H. Wang, L. Su, H. Jiang, Z. Xie, X. Nie, and D. Pei. Chain-of-event: Interpretable root cause analysis for microservices through automatically learning weighted event causal graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pages 50–61, 2024.

A. Yehoshua, I. Kolchinsky, and A. Schuster. Cco - cloud cost optimizer. In *Proceedings of the 16th ACM International Conference on Systems and Storage*, pages 137–137, 2023.

C. Zhai and S. Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*, volume 12. Association for Computing Machinery and Morgan & Claypool, 2016. ISBN 9781970001174.

X. Zhang, T. Mittal, C. Bansal, R. Wang, M. Ma, Z. Ren, H. Huang, and S. Rajmohan. Flash: A workflow automation agent for diagnosing recurring incidents. October 2024.

L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. https://arxiv.org/abs/2306.05685, 2023.

X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 47(2):243–260, 2018.

Z. Zhu, C. Lee, X. Tang, and P. He. Hemirca: Fine-grained root cause analysis for microservices with heterogeneous data sources. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–25, 2024.

Appendix

In the appendix, we provide additional analyses and more extensive discussions about ITBench, individual personas (SRE, ComplianceOps, FinOps) and agent performance. Data, code, and leaderboard at "link anonymized."

Table of Contents

A. Related Work

LM and agents for resolving IT automation tasks. There is a surge in use of AI/ML for handling IT automation tasks. We describe related work for each persona.

A.1. Site Reliability Engineering

IT incident[2] resolution encompasses tasks such as detection (e.g., identifying anomalies or outages) (Guo et al., 2015; Leners et al., 2011; Sigelman et al., 2010; Fonseca et al., 2007), diagnosis (e.g., pinpointing root causes through metrics and logs) (Tan et al., 2019; Jha et al., 2020; Ma et al., 2014; Salesforce, 2023), and mitigation (e.g., operational fixes or code changes). These efforts often rely on supporting tasks like ticket analysis and routing (Gao et al., 2020; Liu et al., 2023b; Arzani et al., 2016), anomaly detection (Liu and Paparrizos, 2024a), topology extraction (Ashok et al., 2024; Chakraborty et al., 2023; Pham et al., 2024; Yao et al., 2024), causal (Budhathoki et al., 2022; Microsoft and contributors, 2023; Ikram et al., 2022; Chakraborty et al., 2023) and interventional (Wang et al., 2023; Bagehorn et al., 2022) analysis using IT data. Clearly, there is significant research in this area, fully automating incident resolution or providing actionable insights to humans remains elusive due to the complexity of real-world systems, the variability of incidents, and the challenge of incorporating contextual knowledge into AI systems (Jha et al., 2020). Recent advances in language models (LMs) have led to their adoption of ticket data analysis and diagnosis tasks (Roy et al., 2024; Ahmed et al., 2023a; Xie et al., 2024b; Chen et al., 2023; Zhang et al., 2024). Most notable examples include Cloud Atlas use LLMs for causal graph construction (Xie et al., 2024b), RCACopilot for ticket analysis (Chen et al., 2023) with the aim to diagnose and mitigate incidents. However, they achieve poor performance compared to other techniques. For example, (Roy et al., 2024) shows that chain-of-thought only achieves accuracy of 35%. More recently, LMs are used in agentic workflows, engaging with real or virtual environments, using several tools at their disposal, for tasks like web navigation (Drouin et al., 2024; Boisvert et al., 2024; Koh et al., 2024), system control (Sahu et al., 2024; Zhang et al., 2024; Chen et al., 2024a), and code generation (Yang et al., 2024a). However, the initial results of these works show a high variability in the success rate —35% in InsightBench (Boisvert et al., 2024) and the ReAct-based agent for ticket data analysis (Roy et al., 2024) to 100% in Flash (Chen et al., 2024a) for incident resolution despite the fact that it is a much harder task than identifying planted insights in tabular and ticket data. Our own results in this work suggest that LLMs and agents struggle to consistently complete incident resolution tasks. *We assert that the variability in success rate exists because of difference in realism of these datasets.* This highlights the urgent need for standardized and open source benchmarks to evaluate and improve the efficacy of AI methods on incident resolution tasks effectively.

SRE-focused Benchmarks The benchmarking landscape for IT operations (ITOps) tasks is still in its early stages, with a few existing efforts addressing specific aspects of the domain. AIOpsLab (Chen et al., 2024a) focuses on resolving IT incidents *only* for SRE personas, covering ten distinct problems created in a real environment. It does not follow SRE best practices for system and application observability, e.g., using an alert management system, lacks comprehensive coverage analysis, and a leaderboard for systematic automated evaluation.

InsightBench (Sahu et al., 2024) targets the analysis of ServiceNow ticket data, a critical supporting task for incident routing and finding relevant past incidents, but its reliance on synthetic data and the lack of a real environment limit its applicability to agentic workflows. Similarly, TSB-AD (Liu and Paparrizos, 2024b) is designed for univariate and multivariate anomaly detection, a core task for incident detection. However, it is limited to synthetic data and focuses only on anomaly detection.

A.2. Compliance

Compliance automation software is emerging to help businesses streamline and automate compliance processes, reducing the need for manual monitoring and tracking of regulations. This ensures continuous adherence to laws. In particular, compliance as code is a very recent development in the IT industry motivated by companies and audit agencies shifting from annual audits to expectations of continuous and automated measurement of compliance to maintain control of their regulated environments' posture and risks for cyberattacks.

Recent works (Papanikolaou et al., 2011; Tupsamudre et al., 2022) have applied AI/ML techniques to speed up these tasks, focusing on mapping regulatory requirements to standard control frameworks such as NIST 800-53 (NIST 800-53). Our agentic automation in the current ITBench solution pioneers this type of effort to author compliance artifacts through AI / ML by bridging compliance as code into policy as code. Policy engines have a longer history in the IT industry compared to compliance as code; however, emerging general usage policy engines such as (Int, d) try to address the need for a common framework for continuous compliance. We are not aware of any effort -albeit critical and needed- related to benchmarking of compliance automation software, whether with or without agentic support.

[2]We use the term scenario broadly to refer to failures, performance problems, compliance issues and, cost anomalies.

A.3. FinOps

The area of IT cost management encompasses multiple disciplines, namely FinOps, IT Financial Management (ITFM), Technology Business Management (TBM) and Porfolio Businesss Management (PBM). At present, the FinOps domain typically deals with cloud costs (Storment and Fuller, 2023; Yang et al., 2024b), which includes compute nodes, memory, other storage, networking, etc., that are incurred with one of the hyperscalers. ITFM includes on-prem infrastructure, licensing, labor, procured services, tech support, etc. The TBM Council provides a standard taxonomy to describe cost sources, technologies, IT resources (IT towers), applications, and services. In addition, there are industry-specific extensions to the taxonomy, such as for healthcare, banking, etc. In essence, this taxonomy provides a generally accepted way of categorizing and reporting IT costs and other metrics. PBM refers to the practice of managing a collection of projects and programs within an organization, ensuring alignment with the overall business strategy and maximizing their collective value by allocating resources efficiently. The FinOps Foundation has indicated that over time it will include elements from ITFM, TBM, and PBM.

Currently, for FinOps, there is no benchmark that fits the definition of benchmark that we are using in this paper. However, over the years, the FinOps Foundation (Foundation, 2025a) has compiled several KPIs that can form the basis for use cases and scenarios for a FinOps benchmark. Current FinOps Foundation KPIs include:

- Usage or Spend Apportionment Validation
- Total Unpredicted Variance of Spend
- Percent of Compute Spend Covered by Commitment Discounts
- Effective Savings Rate Percentage
- Percentage of Commitment Discount Waste
- Percent of Unused Resources
- Auto-scaling Efficiency Rate
- Forecast Accuracy Rate (Usage, Spend)
- Percentage of Unallocated Shared CSP Cloud Cost
- Percentage Variance of Budgeted vs. Forecasted CSP Cloud Spend
- Percentage of CSP Cloud Costs that are Tagging Policy Compliant
- Percent Storage on Frequent Access Tier
- Percentage of Legacy Resource

With the advent of the cloud, the academic and industrial research communities have also been active in investigating ways to optimize costs while balancing multiple objectives. Recent works in the space of FinOps have focused on applying machine learning and mathematical optimization techniques (Qiao et al., 2021; Yang et al., 2024b) to better serve customers' cloud infrastructure needs while offering them insights and recommendations on how they could optimize their overall cloud spend. (Fangkai et al., 2023) addresses the issue of helping customers make trade-offs between cost and resource availability in the presence of offerings such as spot VMs which are cheaper than on-demand VMs but have reduced availability. They propose a framework that uses constrained reinforcement learning to optimize cost and availability by identifying an optimal mix of on-demand VMs and spot VMs. Papers such as (Diao et al., 2024; Feng et al., 2023; Quattrocchi et al., 2024) propose forecasting algorithms to scale cloud resources for service level objectives, contributing to the broader field of FinOps-driven cost optimization. (Osypanka and Nawrocki, 2020) uses anomaly detection, machine learning, and particle swarm optimization to achieve a cost-optimal cloud resource configuration. (Liu et al., 2023a) analyze the process of using cloud storage to explore opportunities, motivations, and challenges of cost optimization from user perspectives. (Nodari et al., 2016) focuses on finding the optimal combination of on-demand and reserved instances, such that the demand is satisfied and the costs minimized. They model this optimization problem as a stochastic inventory control problem.

(Yehoshua et al., 2023) introduces a scalable cost optimizer that determines the most cost-effective deployment strategy for workloads on public or hybrid clouds, considering resource requirements and constraints to minimize costs. In FinOps, there is an urgent need to move beyond comparative scorecards and broad taxonomies to specific use cases that test the ability of automated agents to optimize IT investments and reduce resource waste. To our knowledge, no benchmarks exist for use cases like forecasting, anomaly detection, or cost optimization, nor are there standardized methods to evaluate these techniques with or without agentic support. We are confident that ITBench will unite research and development communities to tackle real-world problems through the power of AI and optimization.

B. ITBench

ITBench framework, as shown in Figure 7, supports two main phases corresponding to two personas as follows: (i) **benchmark registration** phase, where the target is the Benchmark Submitter persona, and (ii) **agent registration** phase, focusing on the Agent Submitter persona and the actual runtime benchmarking execution and evaluation.

B.1. Benchmark Registration

This phase comprises two main steps: (i) scenario development and registration, and (ii) tasks and evaluation metrics registration.

Scenario Development and Registration

Our scenarios are designed to instantiate real-world IT problems in realistic and manageable environments. Each scenario comprises of two core components: (i) an environment specification, and (ii) a scenario specification metadata. The Benchmark Submitter persona then registers these scenarios with ITBench, which stores them in its database. Each scenario is described using the metadata shown in Table 7.

Table 7: Scenario Metadata and Examples.

Field	Example
Type	CISO, SRE, FinOps
Name	For CISO: k8s CIS-b Minimize containers w/ shared net namespace
Description	For CISO: Minimize the admission of containers wishing to share the host network namespace
Complexity	Easy, Medium, Hard
Class	For CISO, this is defined based on the technology (e.g., k8s w/ Kyverno; k8s w/ OPA; Rhel9 w/ OPA).

Tasks and Evaluation Metrics Registration For each scenario type, the Benchmark Submitter registers a well-defined set of tasks that form the basis for the Agent performance evaluation. Table 2 summarizes the ITBench currently supported IT automation tasks. Moving forward, we plan to extend ITBench to incorporate additional tasks (e.g., threat analysis and resource optimization) and to broaden its applicability to other domains (e.g., DevOps).

B.2. Agent Registration

During this phase, the Agent Submitter first registers as a user on the platform, then follows with the Agent Registration.

B.2.1. Agent Registration

During Agent Registration, the Agent Submitter specifies the agent metadata as shown in Table 8.

Table 8: Agent Metadata and Examples.

Field	Example
Agent Name	–
Agent Type (predefined)	CISO, SRE, FinOps ...
Agent Level	Beginner, Intermediate, Expert (maps to scenario complexity: Easy, Medium, Hard)
Scenario Class	For CISO: rhel9 w/ OPA; Kubernetes w/ Kyverno; Kubernetes w/ OPA, Kyverno update

Once the agent has been registered, the Agent Submitter selects the agent, and the corresponding benchmarks are retrieved from the database using the *agent_type, agent_level*, and *scenario_class* specified during registration for the Agent. The Agent Submitter subsequently receives the tasks that the agent must complete to meet the designated objective, each of which has pre-defined evaluation metrics.

B.3. Leaderboard

Effective benchmarking of IT automation tasks, especially when selecting LLMs tailored to an organization's specific needs, requires consistent tracking and comparison of agent performance. The Leaderboard facilitates this need by offering a predefined, extensible set of performance metrics that provide clear insights into agent performance relative to the evaluation criteria.

The Leaderboard supports both API and UI interfaces, enabling a streamlined benchmarking workflow. Users must register the agent endpoint via the Leaderboard's UI or API. The agent can then query the Leaderboard to retrieve and deploy benchmark scenarios before reporting their operational status. The scenarios can be deployed either automatically by the ITBench, as described above, in its hosted environment, or manually outside the Leaderboard, in the user's hosted environment, in which case both agent and environment can still leverage the same Leaderboard API endpoint to publish status updates.

The end-to-end workflow for the agent benchmarking process, after its registration by the Agent Submitter, is illustrated in Figure 7, and summarized in the following.

1. New benchmark jobs are stored in the Benchmark Queue for processing.
2. The Benchmark Runner fetches a benchmark scenario

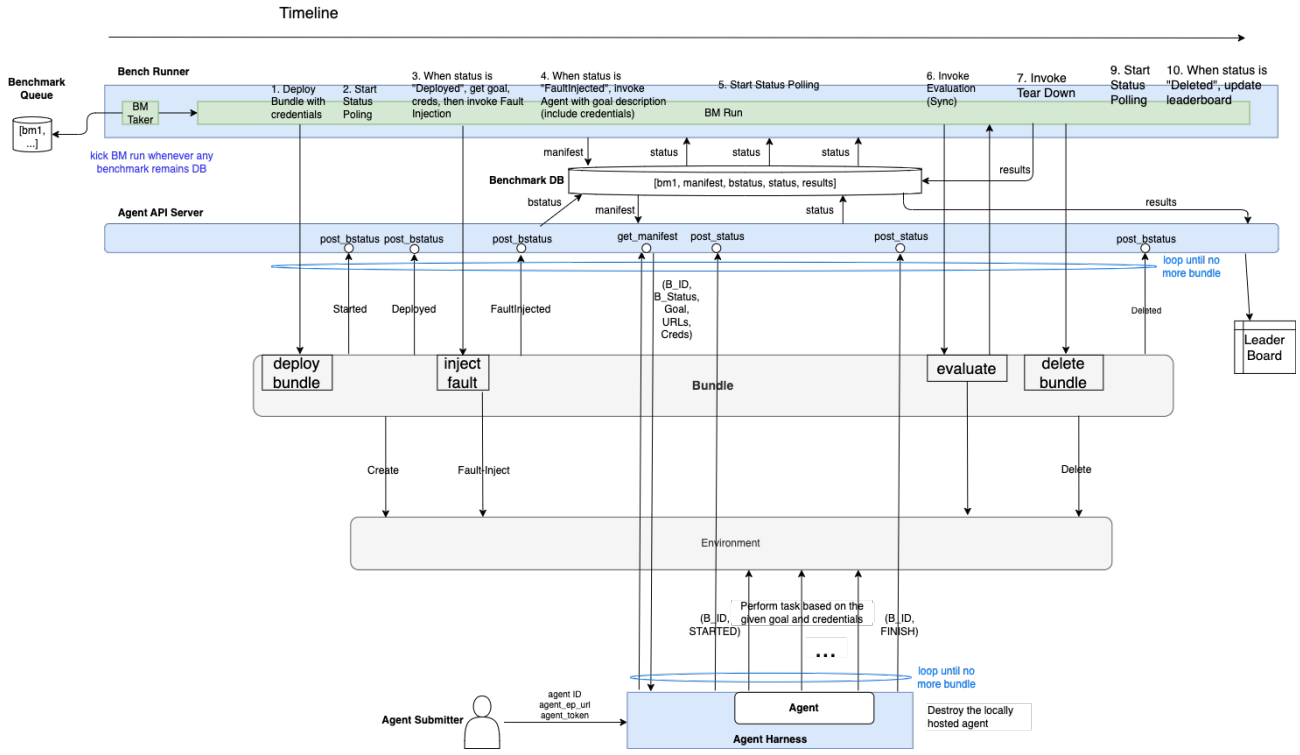


Figure 7: ITBench leaderboard workflow.

for a particular agent from the Benchmark Queue.

3. The Benchmark Runner provisions the environment as per the benchmark scenario specification. The scenario's environment is the set of systems required for the execution of a specific IT task.The Agent interacts with (and can potentially modify) the environment to solve the given IT automation tasks. A benchmark evaluation measures the Agent's performance based on whether it successfully completes the tasks in the given environment. The environment could be, for instance, a Kubernetes cluster running a target application or a RHEL 9 host with a specific configuration to be validated. The environment is under the direct control of the Agent and therefore may be subject to destructive actions (in case of faulty performance), thus functioning as a sort of "playground."
4. For each scenario included in the benchmark run, the Benchmark Runner and the Agent execute the following steps:
   (a) The Agent continuously polls the **get_manifest** API to monitor when a new manifest enters the **Ready** state.
   (b) Benchmark Runner deploys the scenario's environment by executing the **deploy_scenario** function. Each environment reports its status to the Agent API Server using the **post_bstatus** API.
   (c) The Benchmark Runner monitors the environment's status via the Agent API Server's **get_bstatus** API. Once the status becomes **Deployed**, it injects a fault into the environment by executing the **inject_fault** function.
   (d) The Benchmark Runner continues to monitor the environment's status using the **get_bstatus** API. Once the status reaches **FaultInjected**, it updates the manifest's status in the Benchmark DB to **Ready**, including key details such as Benchmark ID, Scenario ID, cluster credentials, and URLs in the manifest. This allows the Agent to access and retrieve this manifest for working with the environment.
   (e) Once the manifest status is **Ready**, the Agent retrieves it. The manifest contains URLs and credentials required to launch the Agent. Before starting the Agent, the Agent calls the **post_status** API of the Agent API Server to report its status as **STARTED**.
   (f) After the Agent completes its execution, the **post_status** API is called again to report the Agent's completion its status as **FINISH**.
   (g) Benchmark Runner starts the evaluation and executes the **delete_scenario** function.
5. Once the evaluation results for all the scenarios in the benchmark are ready, Benchmark Runner aggregates them and publishes the results to the Leaderboard.

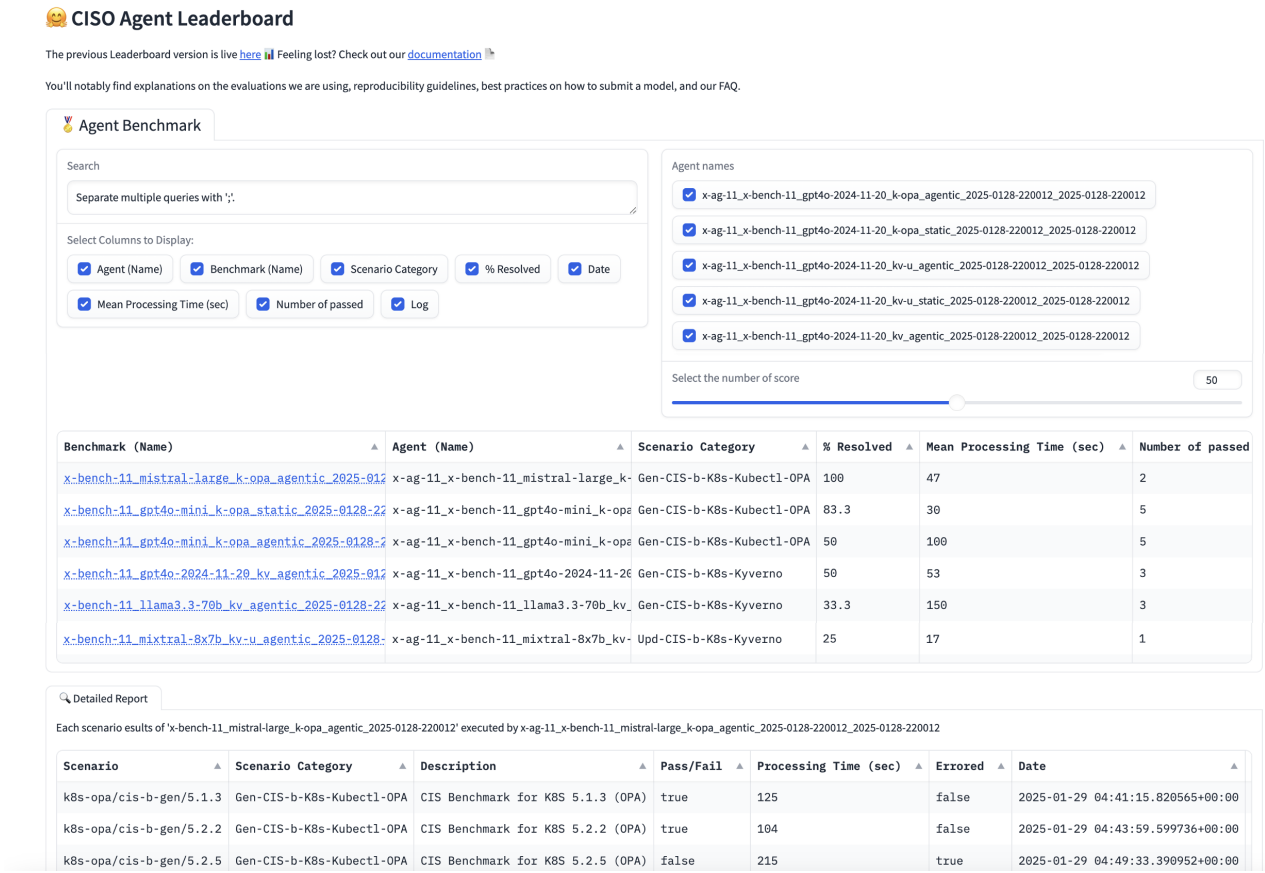


Figure 8: Example ITBench leaderboard.

We instantiated the Leaderboard evaluation metrics for a few IT automation tasks as detailed in Section 3.1, Table 2. In Figure 8 shows the Leaderboard landing page displaying the benchmarking metrics and results for the CISO compliance assessment agent.

C. Site Reliability Engineering

C.1. Background

With the unprecedented growth in scale and complexity of modern IT systems and infrastructures, failures are the norm instead of exceptions (Patterson et al., 2002; Gunawi et al., 2016; Kendrick, 2012; Di Martino et al., 2014; Veeraraghavan et al., 2018; Liu et al., 2019; Ghosh et al., 2022). First, *hardware failures* are frequent in large-scale IT infrastructures. For example, a new cluster at Google undergoes about a thousand individual machine failures and thousands of disk failures every year (Dean, 2009). Many of these failures further trigger correlated failures (Ford et al., 2010). New hardware fault models such as silent data corruptions in compute units (Hochschild et al., 2021) and fail-slow storage (Gunawi et al., 2018) further increase the challenges of detection and mitigation. In fact, in geo-distributed hyperscalar infrastructures, datacenter-level disasters are no longer rare events (Veeraraghavan et al., 2018).

Moreover, high velocity of software changes, *software failures* caused by code bugs (Gunawi et al., 2014) and misconfigurations (Xu et al., 2013) have also become a major cause of IT system failures and service outages, significantly outnumbering hardware failures in recent years (Maurer, 2015; Barroso et al., 2018). For example, IT systems undertake hundreds to thousands of configuration changes daily, which introduces misconfigurations and triggers latent bugs (Sun et al., 2020; Tang et al., 2015). Recent trends in software architectures such as microservices and serverless computing (Jonas et al., 2019) are further enlarging IT reliability challenges by magnifying system complexity and dynamics with sophisticated interactions (Tang et al., 2023) and emergent behavior (Huang et al., 2022).

The goal of Site Reliability Engineering (SRE) is to achieve high availability and serviceability of IT systems, in the presence of the aforementioned failures (Murphy et al., 2024). The essential job of SRE is failure management[3]—detecting, diagnosing, and mitigating failures in production systems to prevent production *incidents* (the failures that cause user-perceived impacts) or to minimize the impacts and damages of incidents when incident *alerts* are triggered. Specifically:

- **Detection.** SRE must promptly detect production failures via logs, traces, and other telemetry data; detecting failures is the first step to prevent incidents or at least minimize their blast radius and impacts.
- **Diagnosis.** SRE must analyze the root causes of detected failures and localize the faults (e.g., the faulty component and the condition that triggers the fault).
- **Mitigation.** SRE must mitigate the failures to prevent propagation that leads to larger failures or incidents. Mitigation typically follows a resolution plan outlining a sequence of actions to restore the system to its expected state (Chen et al., 2024b).

ITBench currently focuses on diagnosis and mitigation tasks with plans to include more tasks such as incident detection, prevention of similar failures/incidents (e.g., by regression testing).

Detection is simplified with golden-signal-based alerts, which observability tools provide natively. Though, the challenge intensifies during an event storm, requiring SREs to distinguish actionable alerts by suppressing false positives and prioritizing those that demand immediate attention — a daily struggle in incident resolution. Both of these tasks are included in ITBench by injecting multiple faults within certain scenarios, causing a flood of alerts. The agent must then determine which alerts to prioritize and in what order.

Urgent need of SRE automation. Currently, SRE is largely a human-based practice—SRE engineers are at the forefront of detecting, diagnosing, and mitigating failures and incidents daily (Beyer et al., 2018; Murphy et al., 2024). However, IT systems are growing in scale and demand beyond what human-based practice can reliably, continuously, and efficiently manage, and the cost of human resources and the limit of human reasoning has already become the bottleneck of failure and incident resolution. Today, SRE for IT systems has already become the major TCO (Total Cost of Ownership) of any cloud and software companies (Boulton; IDC, 2024). *Hence, SRE automation is no longer an optional enhancement, but an operational imperative.*

In fact, today's IT systems are already increasingly managed by operation programs that automate labor-intensive, human-based operations, known as *IT automation*. For example, modern cloud management platforms like Kubernetes (Burns et al., 2016), Twine (Tang et al., 2020), and ECS (Melissaris et al., 2022) implement *operator* programs to automate a wide variety of operations such as software upgrades, configuration, autoscaling, etc. However, so far, SRE has not yet become a common part of IT automation due to fundametnal challenges of failure managements.

C.2. Real-world Incident Example

Table 9 shows a real-world incident report based on SREs' raw work notes. In this incident, SREs were notified of several alerts of type — high error rate (>1% in last 10 minutes) on a service — by Slack. The *fault* occurred due to a "node failure" due to the accidental deletion of resources during a decommissioning process aimed at cutting IT costs. The fault caused shard unavailability, leading to an Elasticsearch

[3] We follow the classic Fault-Error-Failure model (Avizienis et al., 2004), where a *fault* is a root cause such as a software bug, a hardware malfunction, or a misconfiguration. A fault can produce abnormal behaviors referred to as *errors*. However, some of these errors are transient and have no system-level effect. Only errors that propagate and become observable manifest as *failures*, such as crash, hang, incorrect result, or incomplete functionality, etc.

failure and an SLO violation due to the error rate SLI. The fault propagated to cause unavailability of shards which in turn led to elasticsearch failure. The unavailability of elasticsearch caused SLO violation of error rate SLI.

The *Ops resolution plan* included trial and error to finally arrive at a state which allowed SRE personnel to execute existing mitigation playbook[4].

As shown, such incidents provide valuable information in terms of: (i) time to detection, diagnosis (post detection) and mitigation (post diagnosis), (ii) symptoms and customer impact, (iii) faulty condition, fault propagation path and depth, (iv) operation resolution plan, and (v) long term fix and improvements. Such real world insights into fault occurrence, propagation, and resolution are invaluable for fault prevention, and automating incident handling.

C.3. ITBench Architecture

ITBench uses open source technologies to create completely repeatable and reproducible incidents (scenarios) on a Kubernetes platform as shown in Figure 9.

Orchestration. The core workflow involves a sequence of interactions between the SRE-Agent[5] and various components of ITBench. Initially, SRE-Agent (❶) enrolls in the benchmark leaderboard by sending the `enroll` command, which prompts the ITBench to create a session (❷) and provide necessary credentials and details (e.g., Kubernetes access, time limits). Once ready, the agent sends the `ready` signal (❸), triggering the scenario executor to install a selected scenario from the scenarios database. This specification is used to set up the environment and inject the fault, including installation of the observability tools (❹). During the active phase (❺), the agent interacts with the environment using tools like *NL2Alerts*, *NL2Logs*, *NL2Metrics*, and *NL2Traces* to complete the task. Upon task completion or time expiration (❺), SRE-Agent sends the `finish` command (❻), signaling ITBench to evaluate the provided outputs and clean up the environment. The scenario executor validates the work of SRE-Agent (❼) restores the system to its baseline state (❽). The interaction ❸ —❽ continues until scenario manager sends session finish signal (❾).

[4]Playbook is a structured set of predefined procedures or automated scripts that outline the steps required to perform specific operational tasks or respond to incidents. Playbooks standardize responses, reduce errors, and enable automation of repetitive tasks, enhancing efficiency and reliability in IT operations.

[5]Henceforth, we will refer to the agents handling SRE tasks as SRE-Agent

C.3.1. Principles

Following the bench principles indicated in the introduction, our ITBench uses open-source technologies to construct completely repeatable and reproducible scenarios to simulate real-world incidents.

- **Mimic SRE Best Practices.** ITBench follows the guidelines outlined in SRE handbook (Murphy et al., 2024) such as alerting on golden signals per application and enabling monitoring and observability. Hence, in our current version, the detection is provided out-of-the-box using the approach outlined in (Murphy et al., 2024).
- **Mimic Real-world Incidents.** We systematically examined 105 real-world incidents from our SaaS products to derive relevant incident patterns. Although we integrated several of these patterns into our ITBench scenarios, not all were included due to the complexities of accurately reproducing these incidents and mirroring production-level characteristics. Nevertheless, our ITBench will continuously evolve through the ongoing incorporation of additional incident patterns. At the time of writing this paper, ITBench supports 24% Easy, 24% Medium, and 52% Hard incidents, as shown in Figure 11.
- **Provide Observability.** In real-world scenarios, SREs use observability tools alongside command-line access to monitor systems. These tools provide multiple data modalities such as traces, logs, metrics, and events—and support alerting for efficient anomaly detection, trend analysis, and automated troubleshooting. ITBench defaults to Grafana (gra) but can support other tools including IBM Instana (ins), Dynatrace (dyn), and Datadog (dat).
- **Model Data Variability.** Depending on system criticality and budget, some data modalities may be missing; for instance, only about 20% of applications have tracing enabled, complicating incident diagnosis. ITBench allows flexible control to enable, disable, or partially enable data modalities as needed.
- **Manage Scalability** Scenario hyperparameters consists of (i) environment specification and (ii) scenario specification. Environment specification allows (i) application selection and their related infrastructure selection (e.g., replica count), and data censoring parameters. Scenario specification allows selection of hyper parameters (e.g., service name on which to inject fault on). ITBench creates a database of scenarios offline using the aforementioned hyper parameters.
- **Ensure Determinism.** ITBench ensures that alerts are generated according to the scenario specifications before making the scenarios available in ITBench. Moreover, ITBench ensures that all the assertions (e.g., application is running correctly, alerts are fired correctly) are passed before sending the 'READY' state signal to the agent.

Table 9: An incident that occurred on a SaaS data platform. This incident shows the complex relationship between SRE and FinOps persona, as FinOps ensures that IT environment is cost optimized to meet the financial efficiency goals, while SREs focus is on minimizing service impact and resolving the issue.

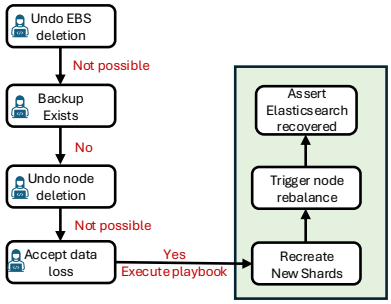
Incident	**Details**
Triggering alert	Seven alerts of type - “High error rate on service."
Summary	Error was encountered due to unexpected node failures and EBS volume issues during the downscaling of the Elasticsearch (ES) cluster because of a human error. Downscaling of ES was initiated to save AWS costs associated with running the service.
Incident duration	180 minutes
Time to detection	60 minutes
Time to diagnosis	60 minutes
Time to mitigate	120 minutes
Symptoms	[✓] Traffic: ↓, [✓] Error: ↑, [✗] Saturation, [✗] Latency
Customer impact	Yes.
Fault propagation depth	six
Fault propagation	↓ Human error: accidental removal of healthy nodes during decommissioning process (maintenance window) ↓ Primary failed while replica initializing (human extrapolation based on the context and manual validation) ↓ Shard assignments failed (ES event: shard unassigned) ↓ Elasticsearch became unhealthy (ES event: RED status) ↓ Services unable to get data from ES (trace) ↓ Increase in error rate on 7 services (events)
Faults	human error, failure during recovery
Resolution plan	
Resol. plan size	5 (4 human + 1 automation via playbooks)
Long term improvements	✓ Maintain 24-hour gap between instance deletion and EBS deletion ✓ Runbooks updated accordingly

Table 10: SRE tasks

Task	Task Description
Fault localization	Identify the faulty entity (root cause) and fault condition.
Fault propagation analysis (aka root cause analysis)	Identifying the causal chain from the root cause entity to the alert, including the identification of fault condition at each step of the chain.
Recommend mitigation actions	Identifying corrective actions to resolve the incident (excluding the execution).
Mitigate incident	Executing corrective actions to clear the alert.

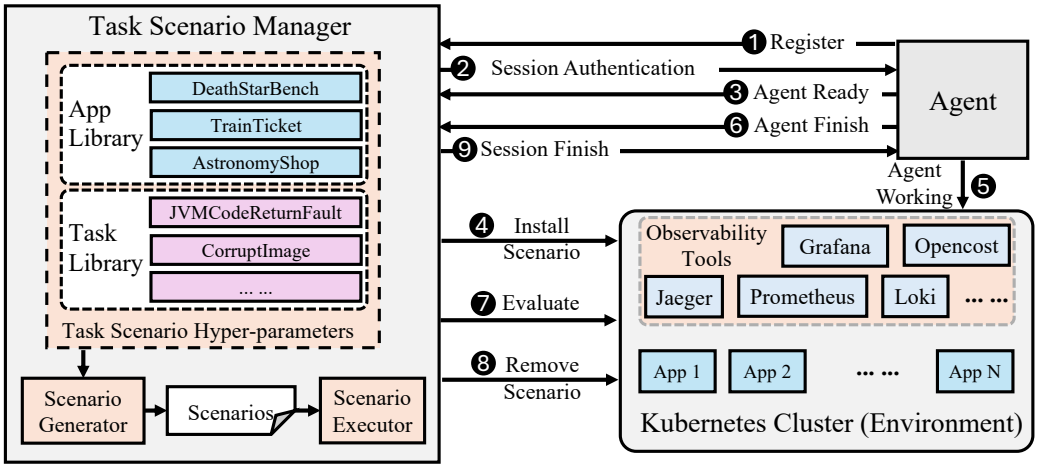


Figure 9: Architecture of ITBench responsible for orchestrating SRE scenarios.

C.3.2. Recreating Incidents in ITBench using Real-world Scenarios

By leveraging detailed incident reports from real-world outages, such as the one summarized in Table 9, we systematically reconstruct similar failure scenarios in ITBench. As outlined in Table 11, this involves configuring a multi-node Elasticsearch cluster with EBS volumes and introducing targeted disruptions—ranging from altering network configurations (e.g., changing ports or IPs) to simulating node and volume deletions, or disabling write operations on specific shards. Each recreated scenario is designed to mirror the complexity of the observed production failures with small variations, including similar failure propagation paths, impact on metrics (such as error rates and latency), and the associated operational mitigation steps. This ensures that ITBench incidents (scenarios) in ITBench accurately replicate real-world technical details while also capturing the associated decision-making challenges, allowing for a realistic and representative evaluation of agents.

C.4. Characterizing ITBench incidents

Table 12 lists the 21 seed scenarios currently available in ITBench. Evaluating each scenario with and without tracing yields 42 SRE scenarios in total for this study. We incorporated 14 additional scenarios after the ICML submission and before the camera-ready version, as detailed in Table 13.

Beyond these 70 ($= 21 \times 2 + 14 \times 2$) scenarios,ITBench can easily produce a far larger range of fault patterns by parameterizing key dimensions such as the target application, the precise location of fault injection, and the number and types of concurrent faults. For instance, if the target application is HotelReservation, the fault of the PodFailure scenario alone can be applied to any of the 18 pods, effectively extending to another 18 scenarios. In this way, ITBench can be used to systematically generate hundreds or even thousands of variations. In our evaluation, we focus on representative scenarios, while still enabling users to customize and scale their tests.

Figure 10 illustrates key incident characteristics observed in our dataset, including the *fault propagation chain length* (Figure 10a), the *resolution plan size* (Figure 10b), and the *number of distinct technologies* involved (Figure 10c). Intuitively, as the length of the fault propagation chain grows, the incident becomes more challenging to diagnose. Similarly, a longer resolution plan suggests that restoring service health requires multiple steps and interventions. The in-

Table 11: Recreated failure scenarios using the incident description described in Table 9.

Testbed Setup

- Develop an application that uses Elasticsearch for data storage and retrieval.
- A minimum of 3 nodes in the Elasticsearch cluster must be configured.
- Attach EBS volumes to each node to simulate the volume usage conditions as in the incident.
- Create an index with a sufficient number of documents to stress the system.

	Incident Scenario 1	**Incident Scenario 2**	**Incident Scenario 3**
Description	Make ES unavailable by changing port, IP address, etc.	(i) Identify a victim node: choose one of the nodes within the cluster and delete it, and (ii) delete the attached EBS volume.	Identify a victim shard and make it read-only (i.e., disable writes).
Fault propagation	IP/Port changed → ES unavailable → Increased error rate in app	Similar to incident described in Table 9	Similar to incident described in Table 9, except caused by hardware failure
Ops mitigation plan	Change the IP address/port to the correct value	Similar to incident described in Table 9	(i) Enable writes on the victim shard, or (ii) follow the procedure similar to incident described in Table 9

Table 12: Unique Scenarios available in ITBench.

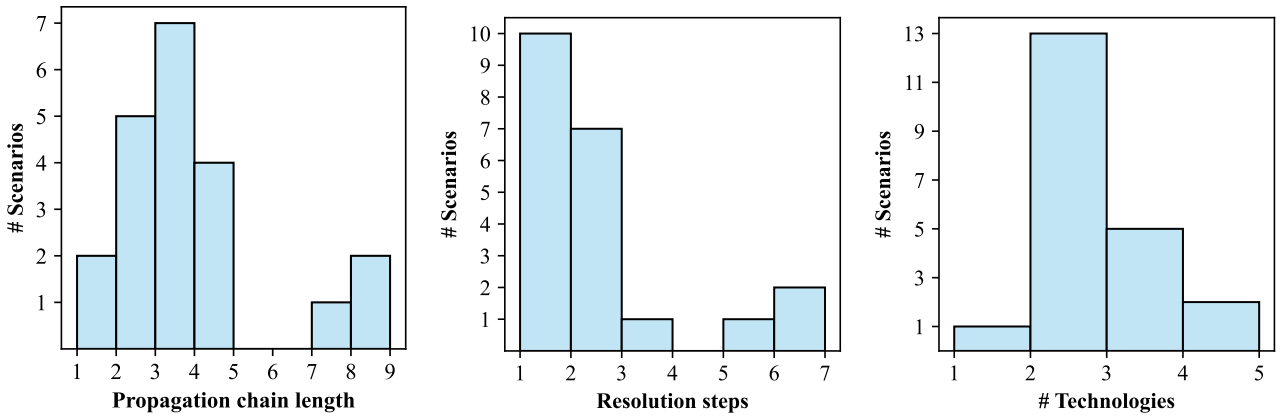
Scenario Pattern	**Technologies Impacted**	**# Fault Propagation**	**# Resolution Steps**
CacheFailure	Node.js	3	2
HighCPU	Java, Node.js	3	2
ServiceFailure	Java, Node.js	4	3
ManualGarbageCollection	Java, Node.js	3	2
MemoryLeak	Python, Node.js, Go	8	6
CorruptDeployment	Go, Java, Node.js	8	6
CorruptDeployment	Java, Go, Node.js	7	5
CorruptDeployment	Go, Node.js	2	1
NetworkDelay	Go, Python, Node.js	4	1
PodFault	Go, Node.js	2	2
NetworkPartition	Tonic, Rust, Go, Node.js	4	1
CorruptImage	Go, Node.js	3	1
CorruptImage	Node.js	2	1
CPUStress	Python, Node.js	2	2
HTTPRequestBodyTamperFault	Ruby, Go	3	1
HTTPRequestAbortFault	PHP, Go, Tonic, Rust, Node.js	4	1
HTTPRequestBodyTamperFault	Ruby, Go, Node.js	3	2
JVMCodeReturnFault	Java, Node.js	3	1
PodFailure	Java, Node.js	1	1
IncorrectAuthentication	.NET, Go, Node.js	2	1
MemoryResourceLimit	Go, Node.js	1	2

volvement of various technologies introduces additional complexity due to the diversity of tools, data sources, and failure modes.

Since *fault propagation length*, *resolution plan size*, and *technology heterogeneity* all influence the difficulty of incident resolution, we define overall task complexity as their

Table 13: Extended (NEW) Unique Scenarios available in ITBench.

Scenario Pattern	Technologies Impacted	# Fault Propagation	# Resolution Steps
JVMHeapStress	Java	3	1
PodUnavailable	Go	3	1
PodUnavailable	Ruby	3	1
PortMisconfigure	Kubernetes, Go	2	2
StorageClassReplace	Kubernetes, Java	3	2
ReplicasetScale	Kubernetes, Java	2	1
UnregisterCredentials	MongoDB	4	2
NonExistentImage	Kubernetes, Go	2	1
UnsupportedImage	Kubernetes, Go	2	1
RedisPassword	Redis, .NET	4	1
RedisOOM	Redis, .NET	3	2
NodeAssign	Kubernetes	1	1
BinaryIncorrect	Kubernetes, Go	2	1
NetworkPolicy	Kubernetes, TypeScript	3	1



(a) Fault propagation chain length. (b) Mitigation plan size. (c) # Technologies involved.

Figure 10: Characterizing ITBench scenarios.

geometric mean. Equation (6) captures this relationship:

$$\text{Complexity} = \sqrt[3]{(\text{propagation path length} \times \#\text{ resolution steps} \times \#\text{ technologies})} \tag{6}$$

This formulation offers a balanced complexity measure, where the geometric mean ensures that all three factors contribute proportionally, rather than allowing one dominant factor to skew the assessment. While factors like required skill sets or the number and type of diagnostic interactions (e.g., tool invocations or queries) could further refine our complexity measure, these factors are often highly dependent on the observability platform, domain expertise, and team-specific processes. As discussed, LMs can potentially mitigate skill gaps through targeted fine-tuning and knowledge integration, thereby reducing the variability introduced by differences in human expertise and diagnostic strategies. Thus, we focus on the three core factors that are more consistent and inherent to the complexity of the incident itself.

Figure 11 presents the distribution of task complexity values across our incident dataset using the above geometric mean formulation. The results show a diverse range of scenarios, with varying degrees of difficulty reflected in the natural interplay among propagation depth, resolution steps, and multi-technology integration. This complexity quantification provides a foundation for future analyses, including

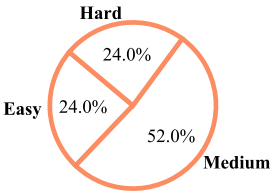


Figure 11: SRE scenario complexity.

evaluating how automated reasoning tools, enriched observability stacks, or improved operator training might shift the distribution toward easier, more manageable tasks.

C.4.1. Experimental Setup

These tasks are implemented as Ansible playbooks to benefit from automation pipelines such as Ansible AWX. Below, we present one of our fault injection implementations, which utilizes Kubernetes network policies to simulate port blocking for a target service. We use roles to define different actions related to both fault injection and fault removal respectively. Our fault injections can be reconfigured using the variables to target different services to create additional scenarios. Each scenario has been validated to produce a relevant alert in Grafana, which provides important context to an agent working on a scenario.

```
---
- name: Define Network Policy to block port
    8080
  set_fact:
    network_policy_spec: |
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: "deny-{{ target_service }}-{{
            target_port }}"
        namespace: "{{
            target_namespace_project_name
            }}"
      spec:
        podSelector:
          matchLabels:
            app.kubernetes.io/name: "{{
                target_service }}"
        policyTypes:
        - Ingress
        ingress:
        - ports:
          - protocol: TCP
            port: {{ target_port }}
          from: []
  when:
    - is_custom
    - is_fault_injection or
        is_fault_removal
    - is_network_policy_service_block

- name: Apply Network Policy
  kubernetes.core.k8s:
    kubeconfig: "{{ kubeconfig }}"
    state: present
    definition: "{{ network_policy_spec }}"
  register: network_policy_apply_result
  when:
    - is_custom
    - is_fault_injection
    - is_network_policy_service_block

- name: Remove Network Policy
  kubernetes.core.k8s:
    kubeconfig: "{{ kubeconfig }}"
    state: absent
    api_version: v1
    kind: NetworkPolicy
    name: "deny-{{ target_service }}-{{
        target_port }}"
    namespace: "{{
        target_namespace_project_name }}"
  register: network_policy_removal_result
  when:
    - is_custom
    - is_fault_removal
    - is_network_policy_service_block
```

For our experiments, we utilized an AWS m4 xlarge cluster configured with 1 control-plane node and 3 worker nodes. The worker nodes had 12 cores and 48 GiB of RAM, with 16 cores and 64 GiB of RAM being used in total. To gain insights into the resource demands imposed by our scenarios, we analyzed the cluster's performance during a one-hour test period. The key metrics include Persistent Volume Claim (PVC) usage, CPU consumption, and memory utilization, as summarized in Table 14.

Table 14: Cluster resource usage during fault injection.

Resources	Usage	Requests	Limits
CPU	2.06571 cores	8.19 cores	6.16 cores
Memory	13.84 GiB	12.89 GiB	16.93 GiB
PVC	62.21 GiB	-	160 GiB

ITBench also supports experiments on Kind clusters, offering a lightweight and portable option for local testing. We validated this capability on a machine with the following configuration: 1 control-plane node, Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz, 12 CPU cores, and 16 GB RAM, running Red Hat Enterprise Linux. This setup allows researchers to efficiently simulate fault scenarios, such as observability stack deployment, OpenTelemetry application deployment, and fault injection tasks, with minimal infrastructure overhead. For example, Incident 22 demonstrated an average CPU usage of 361.71% and memory consumption of 93.53%, confirming the feasibility of Kind clusters for reproducible testing.

C.5. SRE-Agent

As described in Section 3.2, agents interact with the target environment, collect observability data, and execute action to accomplish its goals. For SRE, the goal is to diagnose and mitigate incidents. Below, we describe the observability data collected by the SRE-Agent and our LM-based, multi-agent system implementation.

C.5.1. Observability Data

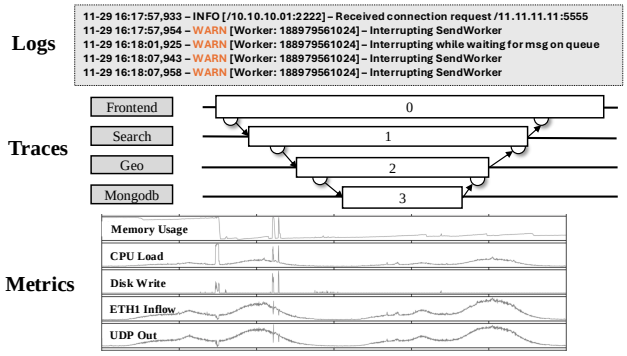


Figure 12: Multi-modality data for SRE task.

As shown in Figure 12, SRE tasks involve analyzing multi-modal observability data: logs, traces, and metrics.

Logs. Logs are semi-structured text records that capture hardware and software events. They are often categorized by severity levels, such as INFO, WARN, and ERROR, to reflect the system's runtime status and the seriousness of its behavior.

Traces. Request traces describe the execution flow of user requests as they traverse through various service instances in a distributed system. They provide a hierarchical representation of service invocations, where each operation is referred to as a span. A span records information about a single service invocation, such as its start time, duration, and associated metadata, including tags and logs. Spans are linked together to form a trace, capturing the complete execution path of the request. Additionally, program exception traces capture program crashes, providing valuable insights for developers during debugging.

Metrics. Metrics provide time-series data monitoring system performance and user-perceived indicators, such as latency, error rates, and resource utilization.

C.5.2. SRE-Agent Architecture and Implementation

The SRE-Agent architecture consists of two LM-based agents, a Diagnosis Agent and a Resolution Agent as shown in Figure 13. We first define the following basic components used in our implementation:

- *Agent.* An agent is an autonomous or semi-autonomous software program that uses a LM to plan, make decisions, interact with the target environment, and execute actions to accomplish goals.
- *Task.* A task is a specific goal that the agent must accomplish before its execution terminates. In our implementation, a task is a complex multi-step process (e.g. diagnosing the cause of an incident). Tasks also have tools associated with them that the agent can use to achieve the goal.
- *Tool.* A tool is a function or API call that the agent can use to perform a specific sub-task, such as, interact with the target environment to collect observability data.

We now describe our implementation of each of the above components.

Tools. Table 15 lists all the tools available to SRE-Agent. All our tools are also LM-based, where the LM is prompted with an utterance from the agent instructing it to perform the required sub-task. The tools are of two types based on whether they generate natural language (e.g., Mitigation) or function calls (e.g., NL2Kubectl). Further, to potentially improve the accuracy and usability of our tools, we equip them with the following features.

- *Reflection.* To enable automatic correction of wrong LM responses, they are provided with external feedback (Pan et al., 2023; Huang et al., 2023) from *linters*. Specifically, for tools that generate function calls, linters are developed to validate the syntax and semantics of the output. If the linter finds a problem with the generated function call, the LM is re-prompted with the linter's feedback so that it can attempt to fix the problem. Similarly, if the generated function call passes linting, but causes an error upon execution, the error message and the failing function call are used to re-prompt the LM for a fixed function call.
- *Summarizer.* For some tools, such as NL2Logs and NL2Traces, the output is not directly returned to the agent because it is very long and contains extraneous information. These tools utilize an additional step that prompts a LM with the output and asks it to provide a detailed summary with only relevant information.

Tasks: We define the following two tasks to be completed by SRE-Agent. Each task includes a description of its completion process and the expected output upon completion. Each task also has tools associated with it that the agent can use to execute sub-tasks, gather information or interact with the environment.

Table 15: List of the tools used by SRE-Agent

Name	Description	Supports Reflection
NL2Kubectl	Interacts directly with Kubernetes	yes
NL2Traces	Interacts with Grafana API for traces	yes
NL2Metrics	Interacts with Grafana API for fetching metrics stored in Prometheus	yes
NL2Logs	Interacts with Grafana API for fetching logs stored in Loki	yes
NL2Alerts	Interacts with Grafana API for fetching alerts	yes
Mitigation	Generates mitigation plans	no
Wait	Pauses execution for the specified seconds	no
Summarization	Summarizes the input content	no
DiagnosisJsonReport	Generates JSON Report of the diagnosis	no
MitigationJsonReport	Generates JSON Report of the mitigation plan	no

- *Diagnosis Task.* For diagnosis, the goal is to identify the entire fault propagation chain, i.e., *fault propagation chain* (FPC) analysis, and identify the exact cause of the problem within the chain, i.e., *fault localization* (FL).
- *Mitigation Task.* For mitigation, the goal is to provide natural-language mitigation plans, and execute them to successfully clear the triggering alert. The mitigation plans increase agent explanability and help SREs in understanding why the agent executed certain commands.

Agents. Overall, SRE-Agent consists of two agents, namely, diagnosis and mitigation agents. Each agent is assigned tasks that it must complete. In general, multi-agent systems can be *hierarchical* or *sequential*. Sequential execution allows tasks to be completed in a fixed, linear order. In hierarchical execution, a "manager" agent determines the task execution order and co-ordinates with the other agents. We adopt sequential execution because it is well suited for the SRE use case, where an incident must be diagnosed before it can be resolved. Although, the order of task execution is fixed, the sub-tasks or steps within each task may be completed in any order as determined by the agent itself. We describe the overall workflow of both our agents below.

- *Diagnosis Agent.* First, the Diagnosis Agent uses the NL2Alerts tool to retrieve the active alerts in the environment. The agent then flexibly and iteratively uses observability tools to gather traces, logs, and metrics from the affected entity mentioned in the alert, and entities associated with the affected entity. It may also use NL2Kubectl commands to investigate the environment. Once the agent determines that it has sufficient information to provide a diagnosis, it proceeds to generate a structured diagnosis report in JSON format with its findings to facilitate evaluation. After the report is generated, the Mitigation Agent takes over.
- *Mitigation Agent.* The Mitigation Agent ingests the diagnosis report to create mitigation plans and then utilizes the available tools to implement the plan. This involves using NL2Kubectl commands. To ensure that the executed commands mitigated the incident, it can also use the NL2Alerts tools to check whether the alerts in environment have been cleared. Further, since alerts could sometimes temporarily appear to get cleared due to fluctuations in a live environment, the agent can use the Wait tool to check whether the alerts *stay* cleared even after some time. Finally, upon completion of the execution, the agent generates a JSON explaining the mitigation steps that it took.

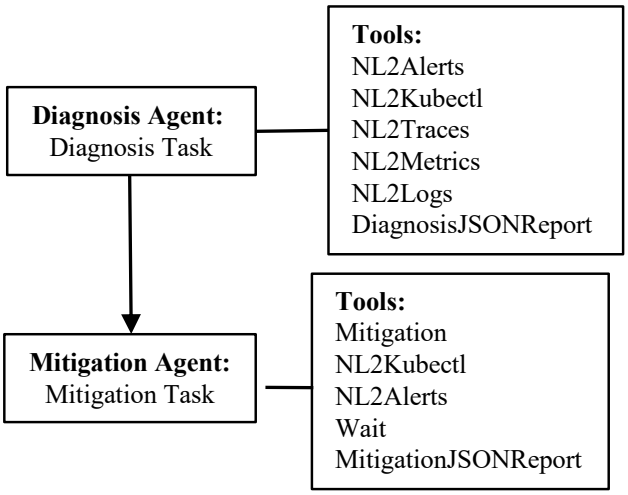


Figure 13: SRE-Agent architecture

C.6. ITBench Evaluation

C.6.1. Experimental details

We evaluate the SRE-Agent agent on a set of 42 SRE scenarios in the ITBench. For the agent's LM-based planning component, we consider four distinct models: gpt-4o, granite-3.1-8b-instruct, llama-3.3-70b-instruct, and llama-3.1-8b-instruct. None of these models are fine-tuned.

Table 16 shows the main hyper-parameter values used in our experiments. These values were chosen to ensure as deterministic results as possible. $decoding\_method$ is applicable for all models except gpt-4o.

Table 16: Model hyper-parameters.

Hyper-parameter	Value
$temperature$	0
top_p	1e-7
$seed$	42
$decoding\_method$	greedy

C.6.2. Evaluation Metrics

We evaluate each LM-based agents on two primary tasks: (i) Diagnosis and (ii) Mitigation.

Diagnosis. The agent is evaluated for diagnosis based on its ability to provide accurate *fault localization* and *fault propagation chains*. Fault localization allows SREs to identify the exact resource *causing* the problem, whereas fault propagation chain allows SREs to understand how the fault is cascading across the application stack and impacting the application. Fault propagation chain can be further used for other important tasks such as blast radius analysis.

- *Fault localization* performance is measured using pass@1 and Normalized Topology-Aware Match (NTAM).
- *Fault propagation chain* is assessed with NTAM. Additionally, we track Mean Time to Diagnosis (MTTD) to gauge overall diagnostic efficiency.

Mitigation. For mitigation, we evaluate how effectively the agent resolves incidents (i.e., clears alerts).

- Success rate is quantified using pass@1.
- Efficiency is captured through Mean Time to Resolution (MTTR).

At the time of writing of this paper, ITBench lacks the ability to automatically measure the natural language-based unstructured outputs fault condition (i.e., what is wrong with the identified resource) but have plans to extend to this task using LM-as-a-judge (Zheng et al., 2023).

C.6.3. Metric definitions

pass@1. We evaluate both fault localization and mitigation using the pass@1 metric (Chen et al., 2021), which is defined as follows:

$$\text{pass@}k := \mathbb{E}_{\text{Scenarios}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

It is an unbiased estimator of correctness in k=1 trials across all scenarios. For *fault localization*, correctness means whether the predicted root cause exactly matches the ground truth root cause. For *mitigation*, correctness means whether the alerts are cleared.

Normalized Topology-aware Matching. Existing approaches for evaluating *fault propagation chains* and *fault localization* focus on exact matches with the ground truth (Ahmed et al., 2023b; Zhu et al., 2024; Chen et al., 2024c), which overlooks topology and finer-grained analysis of propagation chains. For example, existing approaches cannot effectively differentiate agents and models when predicted propagation chains or root causes do not exactly match the ground truth, as they fail to measure how close the predictions are to the actual faults. Hence, we propose a new metric *Normalized Topology-Aware Match* (NTAM), which measures agent performance compared to ground truth via topology-aware distance calculation.

NTAM requires a topology graph, where the nodes are the entities of the system, and edges indicate various types of connections between them (e.g., Deployment owns ReplicaSet). Given such a topology, it can be used to evaluate both the set of entities in the fault propagation chains, and the set of root cause entities for fault localization. NTAM is inspired by topology-based distance metrics and information retrieval concepts, such as BM25 (Fang et al., 2011), that down-weight less discriminative features. It is a flexible, general function with configurable components for fine-grained evaluation of predicted output quality.

Specifically, it consists of the following main components:

- *Topology-based distance scoring* functions, which consider both the edge-type and sub-tree size, rewarding predicted entities closer to the ground truth. Further, nodes with fewer connections (smaller sub-trees) receive higher scores, as they are more discriminative for fault localization.
- A *node importance factor* based on the position of the ground truth entity in the propagation chain. This captures the intuition that predicting the ground truth root-cause entity correctly should be rewarded more than getting another entity on the chain correct.
- *Penalization terms for length mismatch* between the predicted and ground-truth entities. This is to ensure

Table 17: Experimental details

Models	Scenarios	Experiment Setup		
		#Repeats	#Total	%Agent Submission
granite-3.1-8B-instruct	42	10	420	98.76%
llama-3.1-8B-instruct	42	10	420	100.0%
llama-3.3-70B-instruct	42	10	420	100%
gpt-4o	42	10	420	99.75%

Note: "%Agent submission" is the percentage of all trials completed in which the agent returned results.

that predictions having too many or too few entities get lower scores.

All the components have corresponding hyper-parameters that can be tuned to adjust their contributions to the overall score. The final score is normalized to be between 0 and 1, where 1 indicates a perfect match. For fault localization, instead of evaluating the set of all entities, only the ground-truth and predicted root-cause entities are considered.

Mean Time to Diagnosis. For the scenarios where an agent finishes diagnosis successfully (i.e., root cause entities are found), we calculate *MTTD*, which measures how soon (in seconds) an agent performs diagnosis. Otherwise, *MTTD* is set to infinite.

Mean Time to Repair. Similarly, for mitigation, we identify the scenarios where an agent executes an automated action to resolve the faults successfully (i.e., alerts are cleared). For these scenarios, we calculate *MTTR* (in seconds), which measures how soon an agent performs mitigation. Otherwise, *MTTR* is set to infinite.

C.6.4. Evaluation Results

We present evaluation results for four LM-based agents across 42 SRE scenarios in the ITBench framework.

Overall agent results. gpt-4o shows the strongest performance, achieving a 13.81% pass@1 in diagnosis and 11.43% pass@1 in mitigation (Table 4), significantly higher than any other agent. Moreover, it also attains the best scores on the NTAM metrics (FL and FPC). Notably, in hard scenarios (Table 20), gpt-4o is the only agent capable of performing multiple accurate diagnosis (granite only succeeded once), and *none of the agents can repair the hard scenarios*. Meanwhile, llama-3.1-8B, despite having fewer parameters, offers the fastest detection (lowest MTTD of 57.50s) and repair times (lowest MTTR of 245.13seconds) among successful attempts. Although granite-3.1-8B shares the same parameter size as llama-3.1-8B, it demonstrates slightly better diagnostic capabilities yet weaker mitigation ability. llama-3.3-70B performs second best overall, trailing behind gpt-4o on all the metrics we compute.

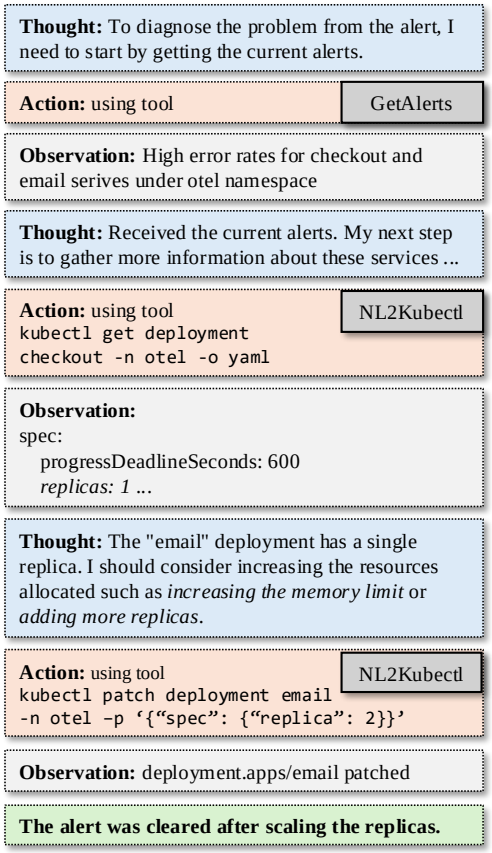


Figure 14: Sample Trajectory of llama-3.3-70b-instruct in Scenario 15

Result analysis by scenario complexity. As we categorize the benchmark scenarios into Easy, Medium, and Hard levels based on the complexity described in Equation (6), a clear performance gap emerges as *scneario_complexity* increases. In particular, Table 18 shows lower diagnosis accuracy (pass@1) in more complex scenarios, and Table 19 reveals a corresponding drop in mitigation success (pass@1). Among the five hard scenarios, none can be resolved by any agent in any run. By contrast, for easy scenarios, over half

Table 18: Diagnosis pass@1 (in %).

Model	Easy	Medium	Hard
gpt-4o	36.00 ± 4.73	7.73 ± 1.74	5.00 ± 2.24
granite-3.1-8B-instruct	8.00 ± 2.68	2.73 ± 1.05	1.00 ± 1.03
llama-3.1-8B-instruct	1.18 ± 1.20	1.36 ± 0.79	0.00 ± 0.00
llama-3.3-70B-instruct	10.00 ± 2.93	1.36 ± 0.78	0.00 ± 0.00

Table 19: Repair pass@1 (in %)

Model	Easy	Medium	Hard
gpt-4o	21.00 ± 4.06	12.27 ± 2.19	0.00 ± 0.00
granite-3.1-8B-instruct	1.00 ± 1.01	0.00 ± 0.00	0.00 ± 0.00
llama-3.1-8B-instruct	5.88 ± 2.48	1.36 ± 0.80	0.00 ± 0.00
llama-3.3-70B-instruct	7.00 ± 2.50	3.18 ± 1.16	0.00 ± 0.00

(five out of eight) scenarios were successfully repaired by at least one agent, and six were diagnosed correctly. We use difference-of-proportions z-test to compare success rates across different task levels (evaluating two levels each time). *The agent performance consistently declines from Easy to Hard scenarios, validating our complexity model based on propagation chain length, resolution steps, and technology diversity.*

Interdependence between diagnosis and mitigation. Interestingly, diagnosis and mitigation are often intuitively assumed to be interdependent, with accurate diagnosis serving as a prerequisite for effective mitigation. However, *our findings reveal that: in some scenarios, agents can successfully mitigate an incident despite misidentifying the root cause.* For example, in scenario 15 (Figure 14), an agent using the llama-3.3-70b model incorrectly identified the root cause as "memory limit" in the service, while the real root cause was HTTP request corruption fault; yet it still managed to resolve the issue by scaling up the email service pods to make it functional, essentially bypassing the handling of actual HTTP fault. Such cases illustrate how generic mitigation actions, such as restarting services or scaling replicas, can sometimes fix the system symptoms even without a fully accurate diagnosis. We observed similar behavior in real-world SRE incident analysis, where, despite the root cause remaining unidentified, SREs were able to mitigate the incident.

Conversely, some scenarios highlight the opposite issue: scenario 13, though labeled as "easy", cannot be fixed by any of the tested agents, even though they achieved high scores in diagnosing the root cause. Notably, gpt-4o attained roughly 80% pass@1 and over 0.7 on both FPC and RC (NTAM) metrics. This implies that, *although the agent cannot fully resolve certain issues, it can still offer near-accurate diagnostic insights, potentially assisting human operators in debugging.*

Some scenarios yield even worse outcomes. For example, in scenario 19 (Figure 15), the agent fails to identify the root cause and cannot repair the system, offering only a mitigation plan at the end that is entirely ineffective.

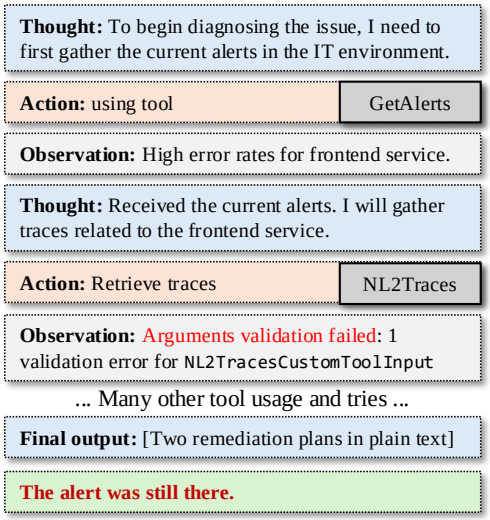


Figure 15: Sample Trajectory of gpt-4o in Scenario 19

Inconsistency between runs. Another crucial observation is the agents' inconsistency across repeated runs. For instance, llama-3.1-8b-instruct and mistral-large-2 in scenario 11 occasionally succeed in only a single run out of 10. Though gpt-4o can reliably repair the scenario 8, its running time can fluctuate between 100 and 800 seconds. *These inconsistencies stem from real-time telemetry fluctuations, where minor changes (e.g., CPU utilization reported at 58% in one run vs. 71% in another) affect LM outputs, leading to varied diagnostic and mitigation results.*

Impact of tracing on accuracy. Many benchmarks provide raw telemetry data, a key differentiator of ITBench is its

alert-driven workflow, which mirrors how SREs are notified of faults through golden-signal-alerts triggered from collected telemetry data. To further assess the importance of different telemetry sources, *ITBench also supports automated telemetry data masking.* As shown in Table 20 and Table 21, gpt-4o sees its diagnosis pass@1 drop from 18.10% (with traces) to 9.52% (without traces), and its mitigation pass@1 plummet to 2.86%. Similarly, llama-3.3-70B experiences its diagnosis rate decline from 5.24% to 0.95%. In fact, only three scenarios were successfully resolved by gpt-4o once trace data was masked. Take Scenario 13 (easy level) as an example. The agent is able to achieve an 80% diagnosis rate in its all runs; however, when masking the traces, the rate drops to 0. Note that all of these telemetry masking and agent evaluation steps are integrated into IT-Bench 's automated pipeline. Agents can be evaluated with different observability configurations in ITBench.

Flexibility and extensibility. Beyond scenario design, *IT-Bench is designed with flexibility and extensibility as guiding principles, allowing for both the addition of new scenarios within existing tasks and the support of new tasks like resource management.* We have already integrated four distinct applications, including both microservice applications (OpenTelemetry-Demo and Hotel-Reservation from DeathStarBench), and non-microservice (TiDB application and Elasticsearch application). ITBench makes it straightforward to incorporate custom applications by simply adding an Ansible playbook. It took around four human hours for the external collaborators to add an application to ITBench. Moreover, by ITBench introduces realistic faults at multiple system layers (e.g., application, virtualization), ensuring a comprehensive evaluation of agent performance across a wide range of failures.

C.6.5. Diagnosis and Mitigation Report

Example of agent JSON report is also provided in Figure 20. The JSON report contains information collected by the agent during its investigation. This includes a list of the faulty entities it encountered, as well as its best guess at the cause of the incident. Further, it contains a list of actions the agent took to try to mitigate the problem.

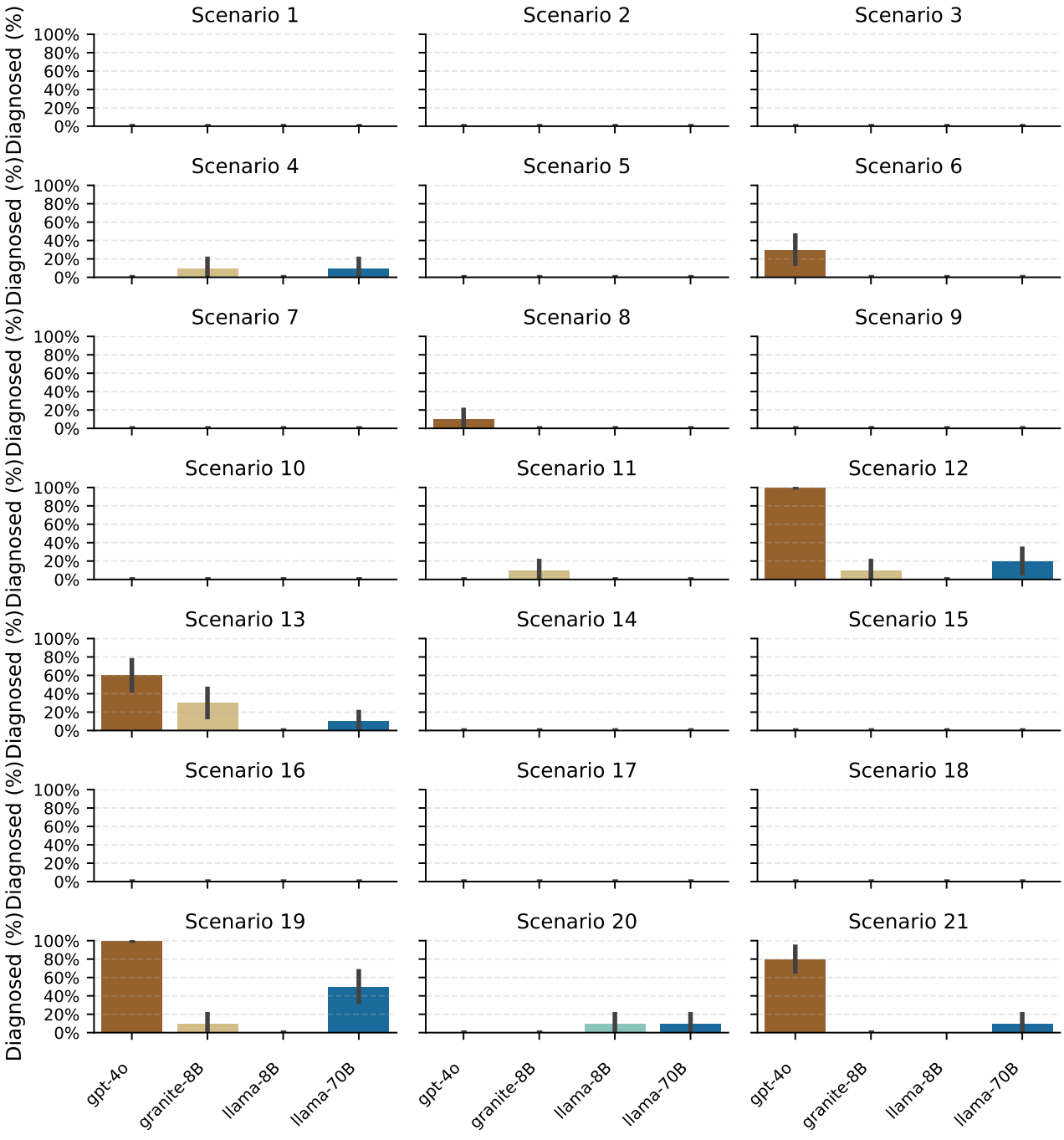


Figure 16: Percent diagnosed for each scenario with tracing enabled.

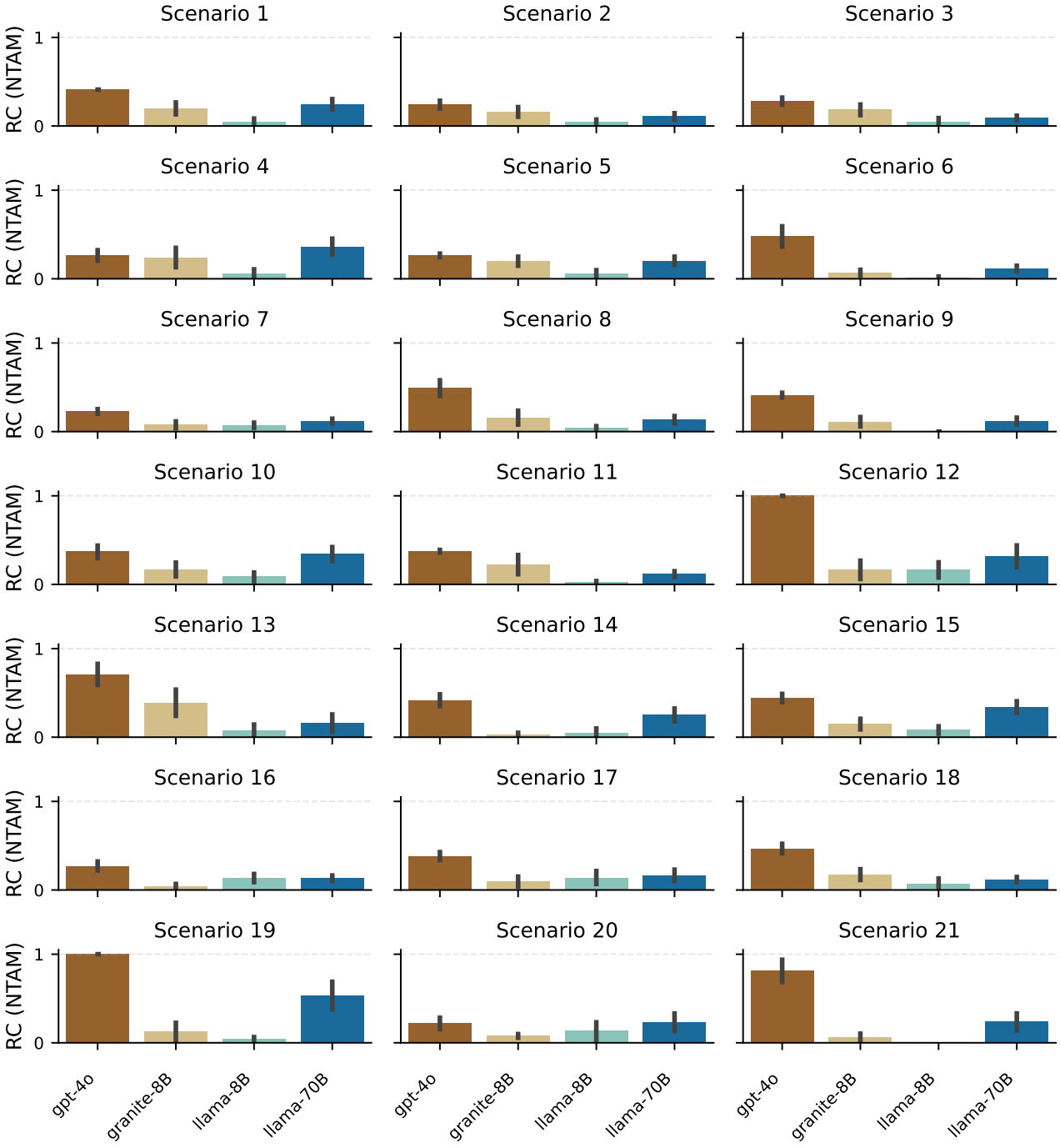


Figure 17: Normalized topology-aware metric (NTAM) for root cause for scenarios with tracing enabled.

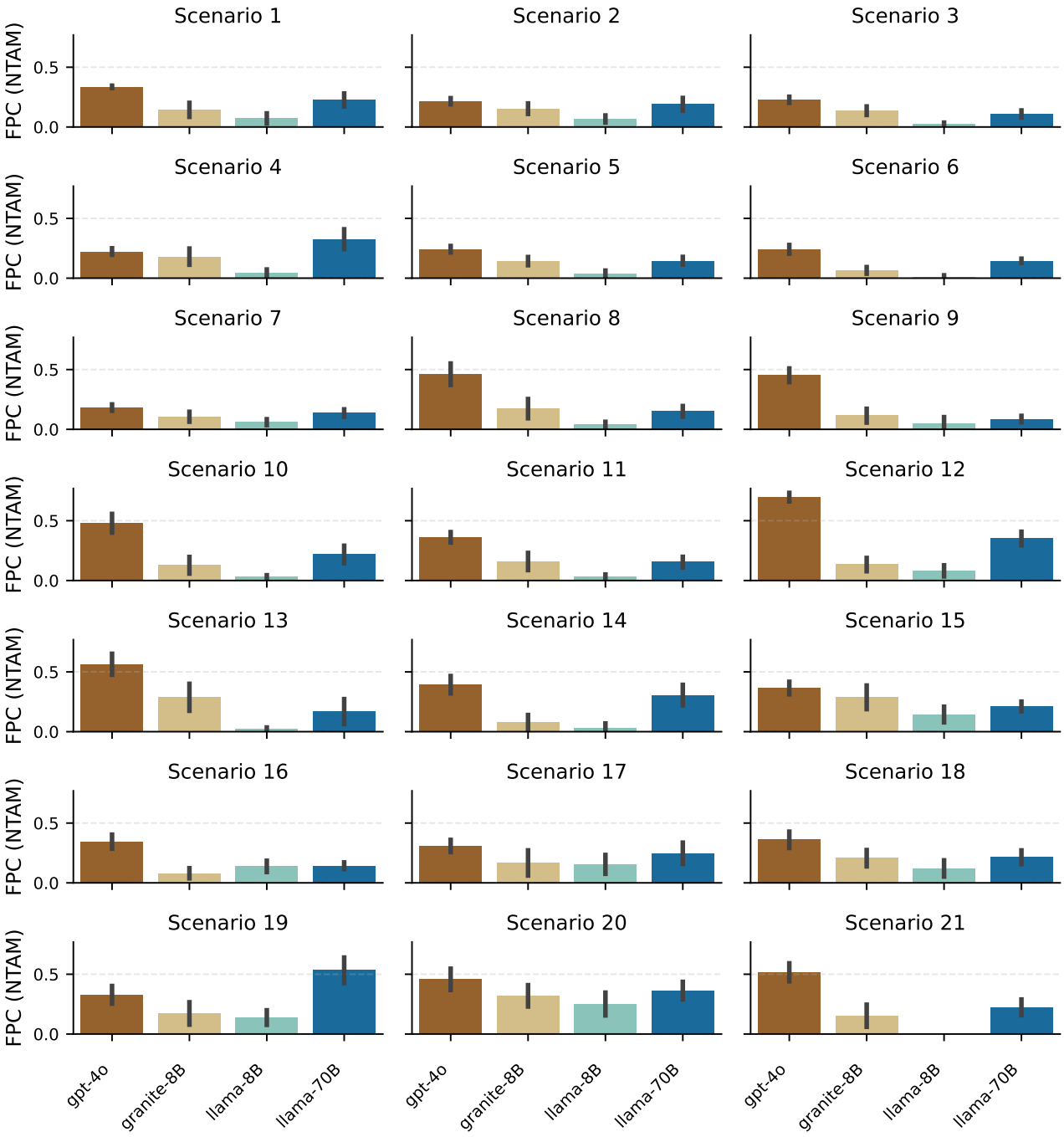


Figure 18: Normalized topology-aware metric (NTAM) for fault propagation chain (FPC) for each scenario with tracing enabled.

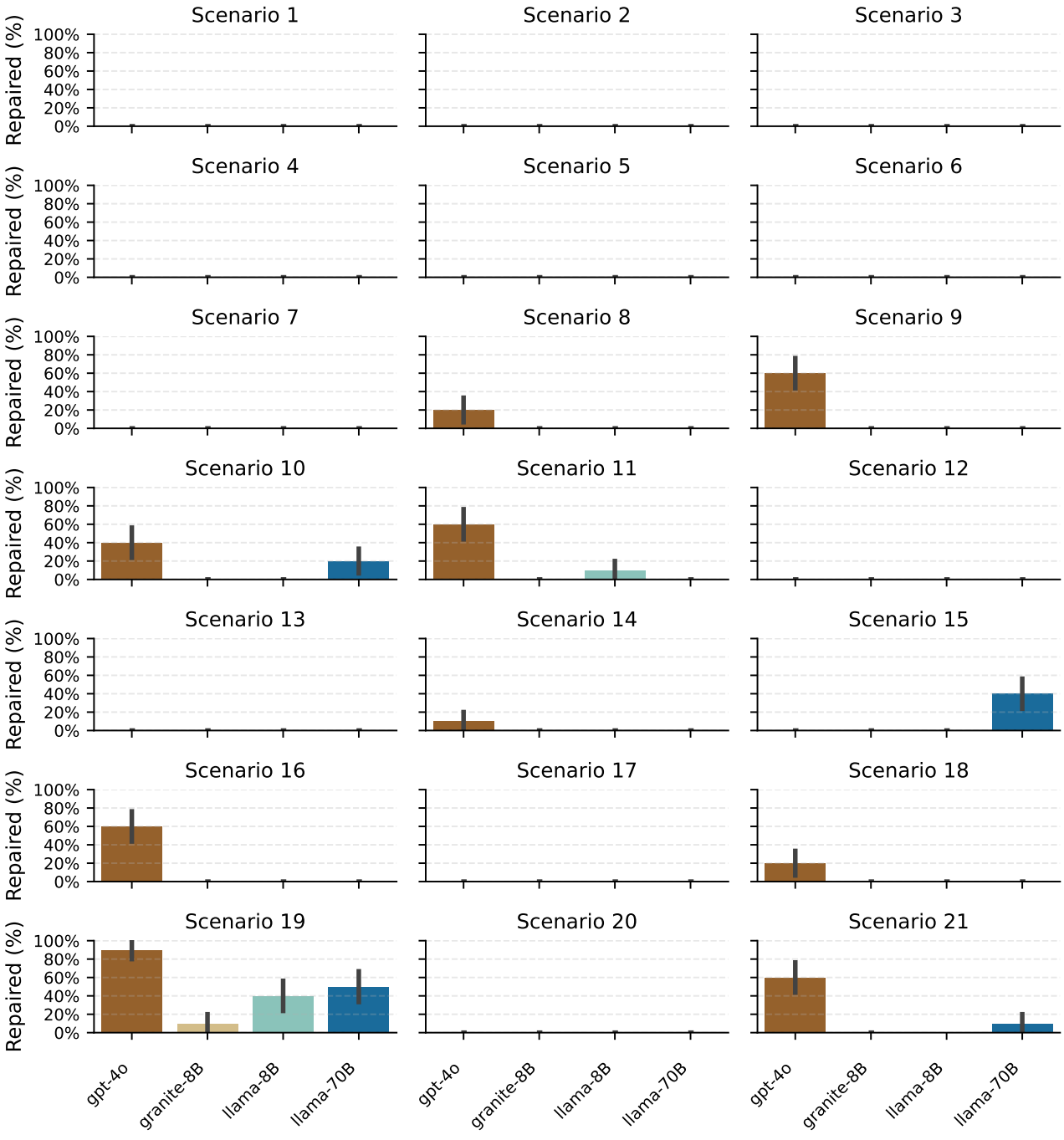


Figure 19: Percent repaired for each scenario with tracing enabled.

```
{
    "alert_start_time": "2025-01-25T14:54:24.976978",
    "entities": [
        {
            "id": "checkoutservice-779456f5fb-x5824",
            "root_cause": false
        },
        {
            "id": "emailservice-768cd9c799-m6wz9",
            "root_cause": true
        }
    ],


    "propagations": [
        {
            "source": "checkoutservice-779456f5fb-x5824",
            "target": "emailservice-768cd9c799-m6wz9",
            "condition": "improper configuration of emailservice to handle volume of
                 requests from checkoutservice",
            "effect": "high error rate in checkoutservice due to emailservice not
                properly handling requests"
        }
    ],
    ...
```


Figure 20: Example agent output for Scenario 15.


```
    ...
    "mitigation": [
        [
            {
                "action": "Describe the deployment emailservice in the otel-demo
                    namespace to understand its current configuration"
            },
            {
                "action": "Patch the deployment emailservice in the otel-demo
                    namespace to increase the memory limit of the container
                    emailservice to 200Mi"
            },


            {
                "action": "Patch the deployment emailservice in the otel-demo
                    namespace to increase the number of replicas to 2"
            }
        ]
    ]
}
```


Figure 21: Example agent output for Scenario 15. - Continued

Table 20: Evaluation of SRE-agent only on scenarios with tracing enabled.

Models	Diagnosis				Mitigation	
	pass@1 (%)↑	FL (NTAM)↑	FPC (NTAM)↑	MTTD (s)↓	pass@1 (%)↑	MTTR (s)↓
granite-3.1-8B-instruct	3.33 ± 1.20	0.15 ± 0.02	0.16 ± 0.01	341.58 ± 81.71	0.48 ± 0.50	$845.50 \pm -$
llama-3.1-8B-instruct	0.50 ± 0.51	0.07 ± 0.01	0.08 ± 0.01	$58.24 \pm ^{-}$	2.50 ± 1.09	245.39 ± 49.45
llama-3.3-70B-instruct	5.24 ± 1.59	0.21 ± 0.02	0.22 ± 0.02	155.78 ± 19.91	5.71 ± 1.60	449.50 ± 46.59
gpt-4o	18.10 ± 2.58	0.45 ± 0.05	0.37 ± 0.03	67.53 ± 3.84	20.00 ± 2.75	266.97 ± 32.95

21 scenarios, 10 runs per scenario.

Table 21: Evaluation of SRE-agent only on scenarios in which tracing is disabled.

Models	Diagnosis				Mitigation	
	pass@1 (%)↑	FL (NTAM)↑	FPC (NTAM)↑	MTTD (s)↓	pass@1 (%)↑	MTTR (s)↓
granite-3.1-8B-instruct	3.81 ± 1.30	0.18 ± 0.02	0.21 ± 0.02	160.97 ± 51.06	0.00 ± 0.00	—
llama-3.1-8B-instruct	1.46 ± 0.84	0.06 ± 0.01	0.07 ± 0.01	57.26 ± 2.88	1.46 ± 0.83	244.96 ± 68.53
llama-3.3-70B-instruct	0.95 ± 0.68	0.11 ± 0.02	0.10 ± 0.02	$430.86 \pm -$	0.95 ± 0.67	1429.80 ± 552.71
gpt-4o	9.52 ± 2.15	0.32 ± 0.05	0.31 ± 0.04	85.19 ± 12.84	2.86 ± 1.12	385.87 ± 15.292

21 scenarios, 10 runs per scenario.

D. Chief Information Security Officer (CISO) and Benchmarking the Compliance Assessment Agent

D.1. Background

Advances in technology are increasing application and infrastructure complexity. As a result, traditional approaches that depend on a dedicated security and compliance team to identify vulnerabilities in production systems and mitigate them based on threat that they pose to the organization, are no longer working. Modern organizations rely on a Development, Security, and Operations (DevSecOps) practice, a process in which application security is verified before deployment, where security and regulatory controls are put in place at software development time. Then, post deployment, runtime checks take over. With these multiple layers of security and compliance checks owned by different teams, some of them with limited cybersecurity knowledge, not only it is no longer feasible to have manual compliance processes for the technical security controls, but the automation of those processes also needs an unprecedented acceleration to keep up the go-to-market pace and scale.

The overall process starts with CISOs, security administrators, or regulators establishing and authoring the relevant body of compliance recommendations, typically in natural language, for specific mission critical environments. Then they rely on dev teams or security focals to collect status as evidence across those environments, and to validate it against the recommendations in view of obtaining the authorization to operate or other certifications. The sought after benefit from automating the **evidence collection** and its **validation** is to enable scalability, both in handling complex environments and supporting frequent scans -daily or on demand per system fix or update- for posture measurement and reporting. Examples of validation automation tools are policy engines such as Kyverno (Int, c), OPA Gatekeeper (Int, b) for Kubernetes , Ansible (Int, a) for PaaS, or Cloud Security Posture Management (CSPM) solutions for the cloud. Figure 22 illustrates the dichotomy of compliance authoring on the left versus compliance validation via policy scripts and their diverse programmatic languages on the right.

Automating the translation of natural-language recommendations into policy scripts requires an unprecedented level of trust and synchronization across domains and experts typically in different business units. Additionally, it also demands an unprecedented level of technical knowledge for the compliance teams typically focused on legal and IP matters.

The rising popularity of AI agents and their projected ability to handle intricate tasks have increased the demand for AI agents managing IT systems (John, 2024; Miguel Carreon, 2024). Given the complexity of the compliance tasks, a major hurdle for this research is establishing systematic methods to assess the effectiveness of our AI agents prior to their production deployment. Consequently, there is an urgency to develop methods for evaluation of AI agents based on real IT tasks and their corresponding environments.

We detailed our CISO compliance assessment agent deployment and execution in its git repo documentation available in open-source (CIS, a). We present below our CISO agent benchmarking performance using a well defined benchmarking methodology with real-world scenarios and environments. A sample of those scenarios with their environment executable automation packages is also available in open-source (CIS, b).

D.2. Real-World Benchmarking

Our CISO compliance assessment agent (CAA) and corresponding bench bring together the latest technology on compliance as code to enable the programmatic expression of regulatory controls and their posture assessment, using Gen AI generation of code to fulfill these tasks. Our agent aim is to empower a compliance team in accelerating the adoption and operation of new regulatory programs by automating the generation of code for the evidence collection and for its posture validation against the requirements, based on compliance requirements described in natural language. Our benchmarking experiments cover the end-to-end agentic workflow from the discovery of the policy assessment engine in the benchmark scenario, the generation of assessment policy as code and its real-time git PR management, deployment, execution, and posture generation. Finally, the results are evaluated, rated, and reported in our ITBench solution leaderboard.

D.2.1. Terms and Notations

We define the following key terms used hereafter to describe the main aspects of the agent framework and benchmarking methodology:

1. *Agent:* An agent is an AI driven software that autonomously acts on behalf of a persona to solve a given task. We group the agents by Agent Types that reflect the IT operations personas, for example CISO, FInOps, or SRE type.

2. *Task:* A task is a specific job corresponding to the role of a persona that the agents aim to automate. Typical tasks for CISO are to collect evidence and assess compliance controls posture.

3. *Scenario:* A scenario is a real-life occurrence of a task in a given setting. For CISO, for instance, *each* Kubernetes CIS-benchmark requirement instantiated

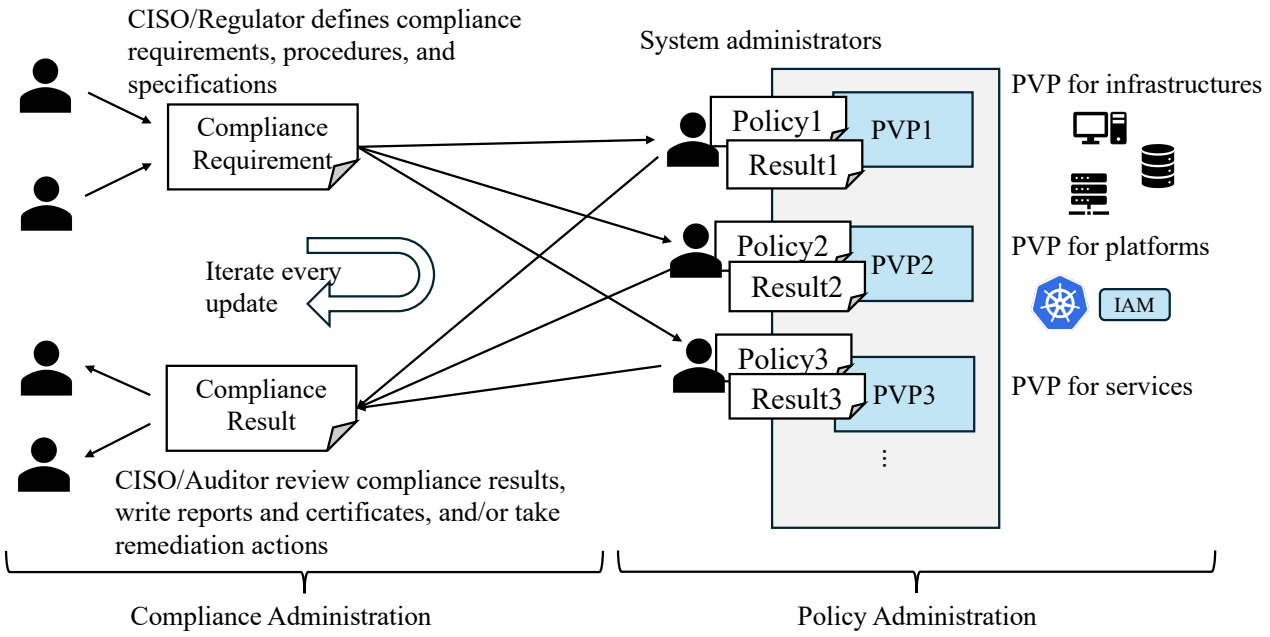


Figure 22: Compliance Authoring and Administration vs. Policy Validation Point Engines

on OPA is a unique scenario. The scenarios can be grouped in classes.

4. *Scenario Class:* A scenario class is a class of real-life scenarios that are grouped together expecting the same behavior and outcome from the corresponding persona. Examples of scenarios classes are the *set* Kubernetes CIS-benchmarks on OPA engine, the *set* of RHEL9 CIS-benchmarks on Ansible engine, or a *set* of Kubernetes CIS-benchmarks updates on Kyverno engine.

5. *Scenario Environment:* A scenario environment is the part of the scenario that specifies the deployment settings.

6. *Environment State:* An environment has a countable set of states that we consider to mark a particular condition at a specific time. Example of states are an environment initial deployment state, an environment failure state after a fault or non-compliant configuration injection, or an environment remediated or compliant state after mitigation.

7. *Goal:* A goal is the desired state for the environment known as the goal state.

Agents are tasked to transition environments from their initial state to their goal state in the most efficient manner. At their disposal are environment actions, including requests for observations or actuation attempts to affect the state of the system. The agents first step towards moving the environment into the goal state is by reasoning over the outcomes of several observational actions to determine the next optimal step. With a strong hypothesis for what that is, agents seek to find and execute strategies to move towards the goal state in the most efficient way.

Our CISO compliance assessment tasks are coupled with pre-defined scenarios for assessing the effectiveness of the agents delivering the automation in a standard manner. We detail in the next section these pre-defined CISO tasks before detailing our ITBench and CISO scenarios.

D.2.2. CISO Compliance Assessment Tasks

The compliance assessment tasks include various activities aimed at comparing the actual state of the systems with the desired state described in English in the policy. Based on this comparison, the system provides a "pass" or "fail" posture with respect to the policy.

- **Identify Evidence Collector (IEC):** Acquiring evidence requires selecting collection mechanisms appropriate to the target system's characteristics. For instance, evidence about the state of an application in a Kubernetes cluster necessitates access to the Kubernetes API, often through tools like "kubectl" command. For host configuration evidence, tools like Ansible Playbooks are suitable. This IEC task and associated agent or tools identifies the collector used in the environment in view of generating the script for its corre-

sponding language and interface.

- **Identify Policy Assessment Tool (IPA):** Evaluating evidence against policies requires selecting a suitable policy engine. For general scenarios, the industry is using the open source policy engine Open Policy Agent (OPA) (Int, d) with its specific programmatic language Rego. Alternatively, for Kubernetes-specific configurations, Kyverno policies prevalently used along OPA. This IPA task and associated agent or tools identifies the appropriate policy engine for the scenario's policy at hand, in view of generating policies code according to the policy engine's programmatic language and interface.
- **Collect Evidence (CE):** This CE task and associated agent or tools is responsible for the actual evidence collection, including the generation of code, management of code, deployment and execution of the evidence collection code to acquire the actual evidence from the environment. Proper placement and execution of the code are necessary to achieve this in a reliable and scalable manner.
- **Scan Assessment Posture (SAP):** This SAP task and associated agent or tools is responsible for generating the posture whether the evidence does or does not satisfy the scenario CIS-benchmark requirement. It includes the generation of validation code, management of code, deployment and execution of code on the policy engine to assess the evidence and produce the compliance posture.

Table 2 summarizes the CISO tasks initially supported in our ITBench, namely CE and SAP. The other will be covered in subsequent releases. These tasks are executed in ITBench against predefined, standard scenarios and compared to the ground truth expected assessment posture "pass"/"fail" stored in the ITBench under each scenario environment specification.

D.3. ITBench Architecture for handling CISO Tasks

ITBench uses open source technologies to create repeatable and reproducible scenarios and environments for the CISO tasks, on a Kubernetes cluster as shown in Figure 23.

D.3.1. Principles

Following the bench principles indicated in the introduction, our ITBench uses open-source technologies to construct completely repeatable and reproducible scenarios that simulate real-world incidents.

- **Mimic CISO best practices.** ITBench follows the guidelines outlined by the National Institute of Standards and Technology (NIST) Cybersecurity Framework (CSF) (NIS) for CISOs and security teams to improve their organization's cybersecurity as follows: (1) *Identify* critical data, systems, assets, and capabilities to protect; (2) *Protect* via security measures that limit the impact of incidents; (3) Develop a strategy to *detect* non-compliance with clear procedures and tools; (4) *Respond* via plans to quickly eliminate threats and mitigate damage; (5) Design a *recovery* policy to support timely recovery to normal operations. Our first ITBench release covers the CISO's CSF first three activities by evaluating the following compliance tasks: (1) Identify and collect evidence from the systems in the selected scenarios; (2) Implement the policies recommended in the scenarios; (3) Assess the policies posture to detect failure into non-compliance. The remaining two CISO's CSF activities of *respond* and *recover* will make the topic of a future ITBench release that will support agents collaboration, namely the leverage of the SRE agent for mitigation.
- **Mimic CISO real-world problems.** We studied and used in ITBench the real-world Cloud Internet Security (CIS) Benchmarks (CIS, 2024) which are a set of best practices for securing the IT cloud infrastructure. They are recognized worldwide as the cloud security standards. We used those CIS-benchmarks to create our CISO compliance policies scenarios of various complexity levels: 25% Easy, 50% Medium, and 25% Hard policies (see Figure 24).
- **Provide observability.** Cloud Native Compute Foundation (CNCF) recent Sandbox project (OSCAL-compass, 2024) released a compliance as code SDK to support the machine readable compliance as code standard (OSCAL, 2024) of the National Institute of Standards and Technology for programmatic usage in compliance automation. ITBench CISO automation leverages this methodology to represent the CIS-benchmarks requirements, detect the events of creation or update of requirements, and trigger the creation or update of evidence collection and validation code.
- **Ensure Determinism.** ITBench enforces the scenarios and their environments are generated as per the specification, while the environment cleanup after each scenario ensures a clean slate for the next run.

D.3.2. ITBench Architecture

The environments for the benchmarking scenarios comprise a Kubernetes Cluster and virtual machines (VMs). Each CISO scenario pre-defined in the ITBench as described in the section hearafter, is managed by a deployable stack, a software component responsible for handling the benchmarking process and environment in a real run-time environment.

The deployable stack manages various tasks, including preparation ("deploy_environment"), fault injection ("in-

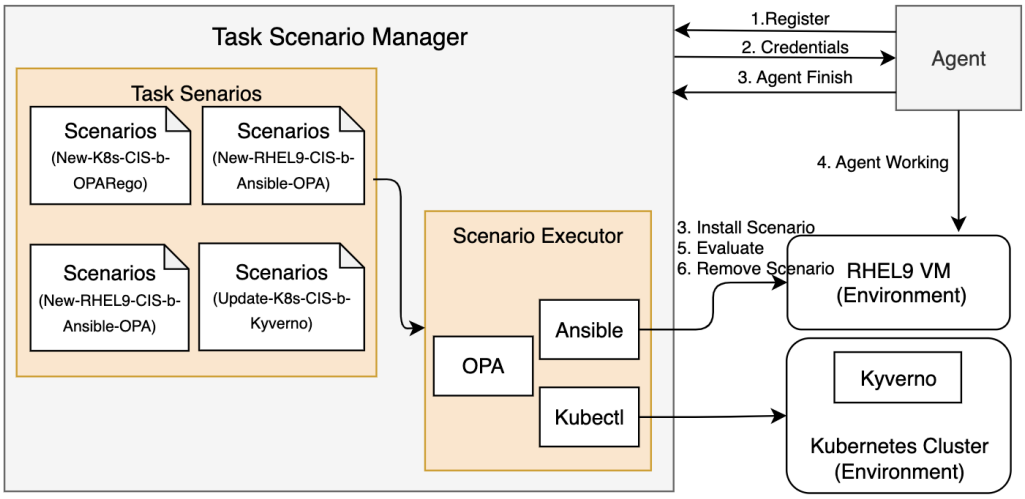


Figure 23: Architecture of ITBench responsible for orchestrating CISO scenarios.

ject_fault"), agent performance evaluation ("evaluate"), and environment cleanup ("delete_environment") for each benchmark scenario run. Each deployable stack is specifically designed for a particular CISO compliance assessment scenario, ensuring the necessary tools, configurations, and policies are in place. Environment administrators define these scenarios deployable stacks and configure the required software, which can involve setting up policy engines, tools, or creating conditions that simulate violations of specific compliance requirements.

A deployable stack may for instance intentionally exhibit a misconfiguration settings to mimic a violation of a particular compliance standard. In the execution of a scenario, the agent is presented initially with the natural language description of that scenario compliance requirement. Based on the description, the agent generates the necessary artifacts, including scripts for evidence collection and policies for evidence evaluation. The run-time scenario environment is deployed and made accessible to the agent. The agent accesses the environment to retrieve evidence, deploy policies, and work toward achieving the specified automation goal of assessing the compliance posture, in this case as "fail" or "not-satisfied".

During the scenario run process, the agent notifies the ITBench of the start and completion of its tasks. Upon receiving the completion notification, the ITBench accesses the environment to measure the benchmarking metrics. Once the metrics for all predefined scenarios are collected, they are aggregated and displayed on the ITBench Leaderboard. Fig. 25 illustrates the end-to-end benchmarking process for the CISO scenarios.

In the context of our CISO compliance assessment benchmarking, the scenario environment, including Kubernetes Clusters and VMs, is prepared by the Agent Submitter. The setup and metric measurements are automated using a tool called Mini-Bench. Benchmarking results, including Task Metrics, are registered with a central Bench Server via API. These results are displayed on the Leaderboard on the Bench Server. This setup allows benchmarking to proceed uniformly, whether using the Agent Submitter's local environment or a remote environment, as the same API interactions are employed in both cases.

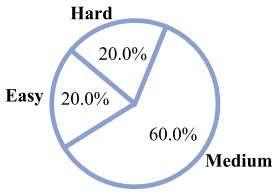


Figure 24: CISO scenario complexity.

D.4. ITBench Real-World CISO Scenarios

We used in ITBench the real-world Cloud Internet Security (CIS) Benchmarks (CIS, 2024) standard to create our CISO compliance assessment scenarios. The technologies that we have considered as playground for benchmarking our CISO agent are Kubernetes and RHEL9, however, any other technology available in CIS can be leveraged with their CIS-benchmark infusing the policies scenario in ITBench.

Table 22 and Table 23 illustrate examples of the typical CIS-benchmarks recommendations. Each scenario rendered on ITBench is designed to mirror the complexity of the recommendation. This ensures that ITBench replicates real-

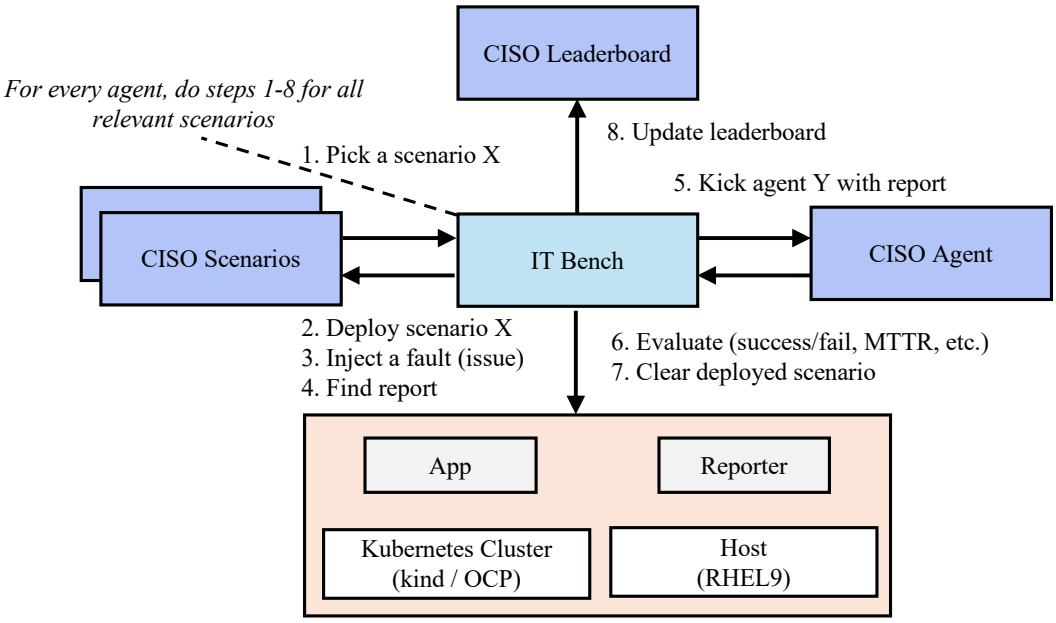


Figure 25: CISO Compliance Assessment Agent end-to-end Benchmarking Process.

world compliance requirements thus allowing for a realistic evaluation of agents.

D.5. CISO Scenario Classes and their Complexity

D.5.1. New-K8s-CIS-b-Kyverno

New-K8s-CIS-b-Kyverno represents the Easy scenario class in the ITBench. The 10 scenarios in this class are prepared based on the CIS Benchmark for Kubernetes, specifically focusing on the Pod Security Policy. This scenario assumes a Kubernetes cluster with a pre-configured Kyverno policy engine. Within the cluster, certain misconfigurations related to Pod Security Policy are present, but the Agent is unaware of their exact locations.

The requirements for the misconfigurations that need to be addressed are communicated to the Agent. Based on these requirements, the Agent generates a Kyverno policy and deploys it to the cluster. Subsequently, the Agent collects the report from the cluster. The accuracy of this report is verified by checking whether it successfully identifies the misconfigurations. If the policy is correctly generated and deployed, the report should indicate the appropriate posture. Conversely, if errors occur, an incorrect posture will be reported.

In this scenario, the four Compliance Assessment Task are evaluated as follows:

- IEC: Assessed by verifying whether the correct configuration is reflected in the Kyverno policy.
- IPA: Evaluated by confirming that the policy is successfully generated for Kyverno.
- CE: Measured by verifying whether a Kyverno report is generated in the predefined location.
- SAP: Determined by whether the posture reported in the Kyverno report matches the expected value.

D.5.2. New-K8s-CIS-b-OPARego

New-K8s-CIS-b-Kubectl-OPARego is categorized under Medium complexity scenarios. This benchmark comprises 10 scenarios derived from the CIS Benchmark for Kubernetes, specifically focusing on Pod Security Policies.

The foundational background for this scenario closely resembles that of New-K8s-CIS-b-Kyverno, sharing the assumption of a Kubernetes cluster as the operating environment and CIS Benchmark for Kubernetes. The key difference lies in the dual output for Open Policy Agent (OPA) policy engine: the generation of both an evidence fetcher script and a policy checker code.

- A fetcher script is designed to gather the required evidence from the target cluster by executing kubectl commands.
- A policy checker verifies the collected evidence for compliance based on predefined rules. This is implemented using the Open Policy Agent (OPA) and defined through OPA Rego policies.

Table 22: Kubernetes - Center for Internet Security Benchmarks (sample)

Section #	Recommendation #	Title	Assessment Status	Description
1.1.1	1.1.1.1	Ensure cramfs kernel module is not available	Automated	The ‘cramfs‘ filesystem type is a compressed read-only Linux filesystem embedded in small footprint systems. A ‘cramfs‘ image can be used without having to first decompress the image.
1.1.1	1.1.1.2	Ensure freevxfs kernel module is not available	Automated	The ‘freevxfs‘ filesystem type is a free version of the Veritas type filesystem. This is the primary filesystem type for HP-UX operating systems.
1.1.1	1.1.1.3	Ensure hfs kernel module is not available	Automated	The ‘hfs‘ filesystem type is a hierarchical filesystem that allows you to mount Mac OS filesystems.
1.1.1	1.1.1.4	Ensure hfsplus kernel module is not available	Automated	The ‘hfsplus‘ filesystem type is a hierarchical filesystem designed to replace ‘hfs‘ that allows you to mount Mac OS filesystems.
1.1.1	1.1.1.5	Ensure jffs2 kernel module is not available	Automated	The ‘jffs2‘ (journaling flash filesystem 2) filesystem type is a log-structured filesystem used in flash memory devices.
1.1.1	1.1.1.8	Ensure usb-storage kernel module is not available	Automated	USB storage provides a means to transfer and store files ensuring persistence and availability of the files independent of network connection status. Its popularity and utility has led to USB-based malware being a simple and common means for network infiltration and a first step to establishing a persistent threat within a networked environment.
1.1.1	1.1.1.9	Ensure unused filesystems kernel modules are not available	Manual	Filesystem kernel modules are pieces of code that can be dynamically loaded into the Linux kernel to extend its filesystem capability, or so-called base kernel, of an operating system. Filesystem kernel modules are typically used to add support for new hardware (as device drivers), or for adding system calls.

The goal of the agent in this scenario is to generate two outputs: 1) a script executing kubectl commands (fetcher), 2) an OPA Rego policy for compliance verification (checker).

The verification process evaluates the outputs generated by the agent as follows: The fetcher script, which consists of kubectl commands, is executed against a real Kubernetes cluster to collect evidence. The collected evidence is then assessed using the generated OPA Rego policy and the OPA policy engine to verify whether the results align with expected compliance outcomes.

The scenario is assessed on the following four Compliance Assessment Tasks:

- IEC: Determine whether a fetcher script, incorporating kubectl commands, is successfully generated.
- IPA: Verify if the checker, implemented as an OPA Rego policy, is correctly generated.
- CE: Check whether evidence can be successfully collected by executing the fetcher script against the cluster.
- SAP: Verify that the OPA Rego policy evaluates the collected evidence as expected, producing the correct compliance assessment.

This approach provides a structured evaluation of the agent's capability to generate effective scripts and policies for Kubernetes cluster compliance assessment.

D.5.3. NEW-RHEL9-CIS-B-ANSIBLE-OPA

New-RHEL9-CIS-b-Ansible-OPA belongs to the Medium complexity scenario class and comprises 20 scenarios based on the CIS benchmark for RHEL9 OS. This scenario shares common characteristics with New-K8s-CIS-b-Kubectl-OPARego, including the generation of two codes (a fetcher script and a checker policy), and the use of OPA Rego Policies for compliance verification. The primary distinction lies in the target system: unlike New-K8s-CIS-b-Kubectl-OPARego, which focuses on Kubernetes clusters, this scenario targets RHEL9 hosts. Consequently, instead of using kubectl as the fetcher script, New-RHEL9-CIS-b-Ansible-OPA employs Ansible playbooks. The objective of this scenario is to generate Ansible playbooks as fetcher scripts and OPA Rego Policies as checkers.

The verification process evaluates the outputs generated by the agent as follows: the fetcher script (Ansible playbook) is executed against a real RHEL9 host to collect evidence. The collected evidence is subsequently analyzed using the generated OPA Rego policy on the OPA policy engine, assessing whether the results align with the expected compliance out-

Table 23: Red Hat Enterprise Linux - Center for Internet Security Benchmarks (sample).

Section #	Recommendation #	Title	Assessment Status	Description
1.1.2	1.1.2.3	Ensure noexec option set on /tmp partition	Automated	The ‘noexec‘ mount option specifies that the filesystem cannot contain executable binaries.
1.1.2	1.1.2.4	Ensure nosuid option set on /tmp partition	Automated	The ‘nosuid‘ mount option specifies that the filesystem cannot contain ‘setuid‘ files.
1.1.3	–	Configure /var	–	The ‘/var‘ directory is used by daemons and other system services to temporarily store dynamic data. Some directories created by these processes may be world-writable.
1.1.3	1.1.3.2	Ensure nodev option set on /var partition	Automated	The ‘nodev‘ mount option specifies that the filesystem cannot contain special devices.
1.1.3	1.1.3.3	Ensure nosuid option set on /var partition	Automated	The ‘nosuid‘ mount option specifies that the filesystem cannot contain ‘setuid‘ files.
1.1.4	–	Configure /var/tmp	–	The ‘/var/tmp‘ directory is a world-writable directory used for temporary storage by all users and some applications. Temporary files residing in ‘/var/tmp‘ are to be preserved between reboots.
1.1.4	1.1.4.2	Ensure noexec option set on /var/tmp partition	Automated	The ‘noexec‘ mount option specifies that the filesystem cannot contain executable binaries.
1.1.4	1.1.4.3	Ensure nosuid option set on /var/tmp partition	Automated	The ‘nosuid‘ mount option specifies that the filesystem cannot contain ‘setuid‘ files.
1.1.4	1.1.4.4	Ensure nodev option set on /var/tmp partition	Automated	The ‘nodev‘ mount option specifies that the filesystem cannot contain special devices.
1.1.5	–	Configure /var/log	–	The ‘/var/log‘ directory is used by system services to store log data.
1.1.5	1.1.5.2	Ensure nodev option set on /var/log partition	Automated	The ‘nodev‘ mount option specifies that the filesystem cannot contain special devices.
1.1.5	1.1.5.3	Ensure noexec option set on /var/log partition	Automated	The ‘noexec‘ mount option specifies that the filesystem cannot contain executable binaries.
1.1.5	1.1.5.4	Ensure nosuid option set on /var/log partition	Automated	The ‘nosuid‘ mount option specifies that the filesystem cannot contain ‘setuid‘ files.
1.1.6	–	Configure /var/log/audit	–	The auditing daemon, ‘auditd‘, stores log data in the ‘/var/log/audit‘ directory.
1.1.6	1.1.6.2	Ensure noexec option set on /var/log/audit partition	Automated	The ‘noexec‘ mount option specifies that the filesystem cannot contain executable binaries.
1.1.6	1.1.6.3	Ensure nodev option set on /var/log/audit partition	Automated	The ‘nodev‘ mount option specifies that the filesystem cannot contain special devices.
1.1.6	1.1.6.4	Ensure nosuid option set on /var/log/audit partition	Automated	The ‘nosuid‘ mount option specifies that the filesystem cannot contain ‘setuid‘ files.
1.1.7	–	Configure /home	–	Please note that home directories could be mounted anywhere and are not necessarily restricted to ‘/home‘, nor restricted to a single location, nor is the name restricted in any way. Checks can be made by looking in ‘/etc/passwd‘, looking over the mounted file systems with ‘mount‘ or querying the relevant database with ‘getent‘.

comes.

In this scenario, the agent's performance is evaluated on the following four Compliance Assessment Task:

- IEC: Does the agent generate a fetcher script in the form of an Ansible playbook?
- IPA: Does the agent generate an OPA Rego policy for the checker?
- CE: Can the Ansible playbook successfully execute against an RHEL9 host and collect relevant evidence?
- SAP: Can the generated OPA Rego policy evaluate the collected evidence and produce the expected compliance results?

This scenario is designed to assess the agent's performance in compliance evaluation tasks for host management environments other than Kubernetes, specifically focusing on RHEL9 systems.

D.5.4. UPDATE-K8S-CIS-B-KYVERNO

Update-K8s-CIS-b-Kyverno falls under the scenario class with a complexity level classified as Hard and currently includes 10 scenarios. Unlike New-K8s-CIS-b-Kyverno, which generates new Kyverno policies based on new requirements specified for that environment in the goal, this scenario involves a different objective. Specifically, it takes an existing Kyverno policy as input, along with instructions detailing modifications to the original requirements, and generates an *updated* policy to meets the revised requirements. The updated policy is then deployed (or updated) as the final output.

The validation process for this scenario is consistent with the methodology used in New-K8s-CIS-b-Kyverno.

D.6. CISO ITBench Evaluation

We conduct our experiments primarily on AWS EC2 instances (m4.xlarge), although ITBench can also be readily deployed on a consumer-grade laptop using a pseudo-cluster, thus making it easier to develop AI agents.

We measure the efficacy of our CISO compliance assessment agent on a set of 50 scenarios across the four scenario classes introduced in Table 2. Each scenario class imposes a distinct set of CIS-benchmarks requirements (e.g., "minimize the admission of containers wishing to share the host network namespace"), each class has a specific level of complexity (e.g., Easy, Medium, Hard), and generates scenario-specific code artifacts.

In our evaluation we considered a variety of LLMs, such as GPT-4o, Llama-3.3-70B-instruct, Llama-3.1-8B-instruct, and Granite-3.1-8B-instruct for tasks that rely on natural language understanding and reasoning. For code-focused use cases, we additionally utilize GPT-4o-mini, Llama-3.1-405b-instruct, and Mixtral-8x7b-instruct. All models use a context window of 128K tokens, enabling them to process more extensive input sequences.

D.6.1. EVALUATION METRICS

The efficacy of our CISO agents is measured based on the ability to detect artifact misconfigurations (aka non-compliance, e.g., no minimum count of containers sharing namespace, or the count is above the threshold), or confirm proper configurations (aka compliance), within the varied environments of the scenario classes randomly injected with misconfigurations.

We evaluate how effectively the agent detects the (non)compliance using the following metrics:

- Success rate is quantified using pass@1.
- Efficiency is captured through Time to Process (TTP).

D.6.2. METRIC DEFINITIONS

pass@1. We evaluate the agent proper assessment of the posture "pass"/"fail" using the pass@1 metric (Chen et al., 2021), which is defined as follows:

$$\text{pass@}k := \mathbb{E}_{\text{Scenarios}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

It is an unbiased estimator of correctness in k=1 trials across all scenarios.

Time to Process. We identify the scenarios where an agent identifies the posture successfully (i.e., misconfiguration results in a "fail" posture, while a compliant configuraiton results in a "pass" posture). For these task scenarios, we calculate *TTP*, which measures how soon an agent performed the assessment. Otherwise, *TTP* is set to infinite.

D.6.3. EVALUATION RESULTS

Overall agent results.

Overall our results in Table 5 and Figure 28 show the GPT-based models dominate on both pass@1 and Time to Process metrics. The pass@1 is nearly 2x better than second-best models (alternating between llama-3.1-405b-instruct and mistral-large-2), while the TTP shows a handling of the scenarios in the minimal time across our scenario classes.

Impact of Scenario Complexity

The complexity of the CISO scenarios is directly mapped to scenario classes. For example, Kyverno scenarios are of Easy complexity, k8s-opa and rhel-opa are of Medium complexity, while Kyverno-update scenarios are of Hard

complexity. As expected, all the models struggle, as expected, as the difficulty of the scenarios increases from the Easy *kyverno* class to the Hard *kyverno-upadate* class.

D.6.4. CISO Compliance Assessment Agent Trajectories

We examine two sample trajectories from our experiments to highlight the agent's thought patterns and actions. These include two scenarios, one successful and one unsuccessful, derived from the NEW-CIS-B-RHEL9-ANSIBLE-OPA experiments with LLaMa 405B, focusing on the CIS Benchmark for RHEL9 (control section 5.1.8 and 5.2.19).

In the successful scenario, the playbook is successfully generated, tested independently, leading to successful data collection, followed by the successful creation of an OPA, and ultimately detecting issues through evaluation in the bench.

In the unsuccessful scenario, the playbook is successfully generated and tested independently, leading to successful data collection, but the OPA cannot be properly created. This results in repeated syntax error corrections and retries, eventually reaching the maximum retry limit.

Following this, we present the trajectories for both cases. In the successful scenario the agent is tasked with performing a compliance assessment based on the following request:

> Ensure cron is restricted to authorized users. The system must: Allow only authorized users to schedule jobs using cron. Specific requirements:
>
> - The file /etc/cron.allow must exist and specify the list of authorized users.
>
> - The file /etc/cron.deny must not exist.
>
> - Unauthorized users (e.g., user1, user2) must not be listed in /etc/cron.allow.
>
> To check the condition, do the following steps. - collect configuration from the RHEL host and save it locally. you can use ansible-playbook to do that. - chcek if the condition is met by using rego policy with the input given by the step above.
>
> for those steps, you need to create ansible playbook 'playbook.yml' and OPA rego policy 'policy.rego'. If you can fix the generated code, do it and run the fixed code again. Once you get a final answer, you can quit the work.

The agent runs this task, provided with the target environment credentials and a working directory for saving the generated files to be submitted. The Figure 26 shows the trajectory that the agent takes to achieve this request.

In the next scenario (CIS Benchmark control section 5.2.19),

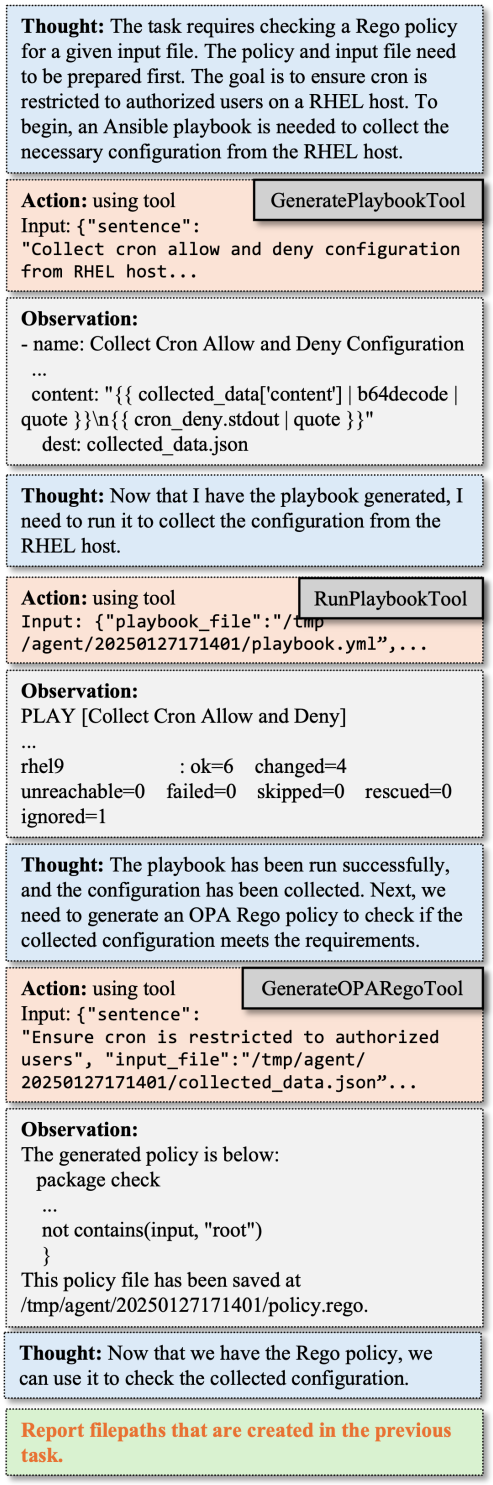


Figure 26: Sample Trajectory of LLaMa 405B in Success Scenario

the agent is given a different compliance requirement. This time, the focus is on ensuring that SSH login sessions adhere to security best practices. However, unlike the previous scenario, the agent encounters a critical issue: it fails to correctly write a syntactically valid Rego policy, causing repeated OPA errors during execution.

The compliance requirement for Scenario 5.2.19 is as follows:

> Ensure SSH LoginGraceTime is set to one minute or less
>
> To check the condition, do the following steps. - collect configuration from the RHEL host and save it locally. you can use ansible-playbook to do that. - chcek if the condition is met by using rego policy with the input given by the step above.
>
> for those steps, you need to create ansible playbook ‘playbook.yml‘ and OPA rego policy ‘policy.rego‘. If you can fix the generated code, do it and run the fixed code again. Once you get a final answer, you can quit the work.

The agent run this task, provided with the target environment credentials and a working directory for saving the generated files to be submitted. The Figure 27 show the trajectory that the agent takes to achieve this request.

After repeating this process three times, the maximum retry limit was reached, and only the currently saved playbook.yml and policy.rego were used at the evaluation. At the evaluation, this policy.rego is syntactically wrong so the evaluation did not pass.

D.6.5. CISO Compliance Assessment Agent Open-Source

We invite our reader to get a more detailed introduction to our CISO compliance assessment agent deployment and execution in its open-source git repo documentation (CIS, a). The CISO agent comes with a sample of scenarios and their environment executable automation packages, available as well in open-source (CIS, b).

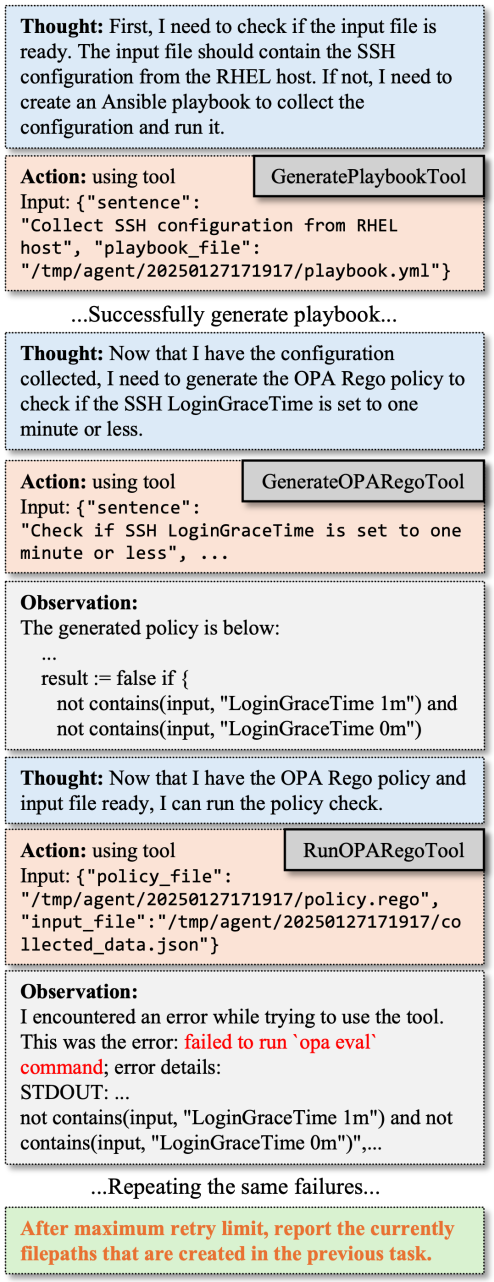


Figure 27: Sample Trajectory of LLaMa 405B in Failure Scenario

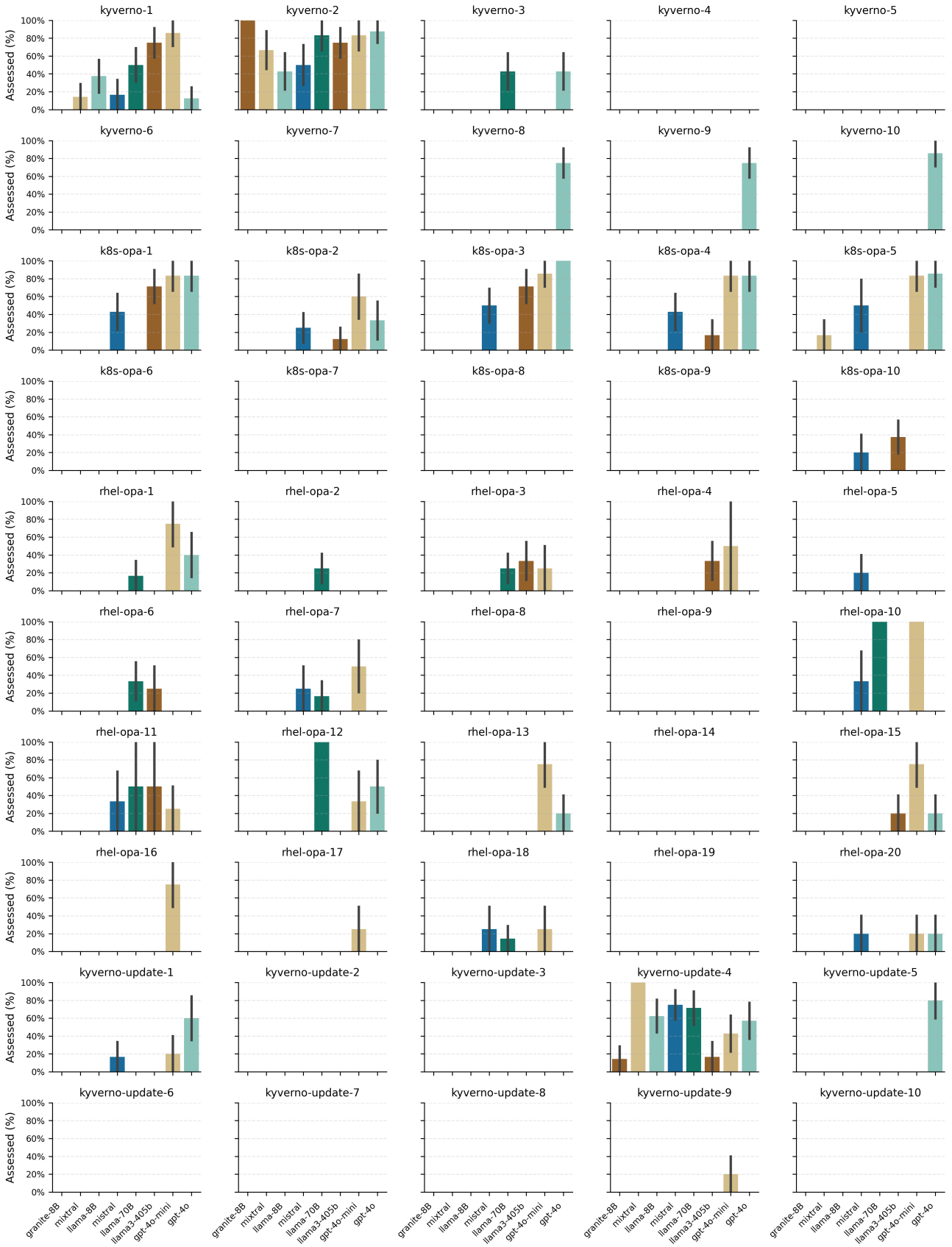


Figure 28: Percent pass@1 for each scenario.

Table 24: Experimental details

CISO/Agent: Models	Scenarios	Experiment Setup			
		#Repeats	#Total	%Exp. Completion	%Agent Submission
granite-3.1-8B-instruct	50	8	400	91.41%	88.48%
mixtral-8x7B-instruct	50	8	400	91.02%	94.67%
llama-3.1-8B-instruct	50	8	400	93.21%	85.71%
llama-3.3-70B-instruct	50	8	400	92.82%	89.11%
mistral-large-2	50	8	400	94.48%	85.23%
llama-3.1-405B-instruct	50	8	400	92.43%	84.50%
gpt-4o-mini	50	8	400	95.25%	85.71%
gpt-4o	50	8	400	90.83%	90.54%

Note: "%Exp. Completion" is the percentage of experiments that were started and finished by the bench runner correctly.
Note: "Trials Agent Submitted" is the percentage of all trials completed in which the agent returned results.

Table 25: CISO/Agent: Assessed Results

Models	Scenario pass@1 (%) ↑				Avg. pass@1 (%) ↑	MPR (s) ↓
	kyverno	k8s-opa	rhel-opa	kyverno-upadate		
granite-3.1-8B-instruct	7.84 ± 3.84	0.00 ± 0.00	0.00 ± 0.00	1.59 ± 1.58	1.71 ± 0.76	197.03 ± 2.52
mixtral-8x7B-instruct	7.35 ± 3.19	1.43 ± 1.42	0.00 ± 0.00	1.29 ± 4.34	3.94 ± 1.03	120.63 ± 3.77
llama-3.1-8B-instruct	8.57 ± 3.37	0.00 ± 0.00	0.00 ± 0.00	7.46 ± 3.23	3.59 ± 1.07	121.49 ± 3.00
llama-3.3-70B-instruct	18.46 ± 4.94	0.00 ± 0.00	1.43 ± 2.88	8.06 ± 3.50	9.32 ± 1.67	189.61 ± 2.71
mistral-large-2	6.56 ± 3.20	22.73 ± 5.32	7.23 ± 2.88	10.45 ± 3.77	11.55 ± 1.95	167.98 ± 3.42
llama-3.1-405B-instruct	16.22 ± 4.32	20.83 ± 4.86	8.75 ± 3.26	3.17 ± 2.22	12.46 ± 1.98	178.89 ± 3.37
gpt-4o-mini	16.18 ± 4.54	43.10 ± 6.99	30.38 ± 5.43	9.43 ± 4.08	25.19 ± 2.80	102.40 ± 3.70
gpt-4o	40.28 ± 5.99	39.34 ± 6.55	7.61 ± 2.81	17.74 ± 4.92	24.74 ± 2.64	101.29 ± 3.81

"pass@1" values are in percent. pass@1 is calculated as defined in Codex (Chen et al., 2021)
"MPR" mean processing time
kyverno = New K8s CIS-benchmarks on Kyverno, **k8s-opa** = New K8s CIS-benchmarks on OPA, **rhel-opa** = New RHEL9 CIS-benchmarks on Ansible-OPA, **kyverno-update** = Update K8s CIS-benchmarks on Kyverno.

E. Financial Operations

E.1. Background

FinOps (Finance + Operations) is an operational framework and cultural practice which maximizes the business value of cloud and creates shared financial accountability. One of the primary objective is to enable timely data-driven decision making by fostering collaboration between engineering, finance, and business teams. FinOps comprises of three iterative phases - Inform (Visibility & Allocation), Optimize (Rates & Usage), and Operate (Continuous Improvement & Usage). Importantly, success in FinOps hinges on making iterative changes using real-time insights with data gathered from Application Performance Management (APM), Application Resource Management (ARM), and Finance (Cloud Cost Management) (Storment and Fuller, 2023).

Contrary to common belief, FinOps is about maximizing business value and not just about reducing operating costs. The 2025 State of FinOps Report (Trask, 2025) gathered data from 861 respondents representing $69M in public cloud spend, with 31% currently spending more than $50 million dollars a year in public cloud, 20% spending more than $100m/yr and over 20% spending over $1bn/yr. Similar to the observation made in the 2024 State of FinOps Report, workload optimization and waste reduction remained key priorities across the board. Management of cloud discount programs (such as Savings Plans and Reserved Instances), and accurate forecasting of spend remained high on the list. Other areas such as increased Automation, AI costs, and sustainability registered growing interest. A new observation made this year was that AI spending is now managed by the majority of respondents (63% up from 31% last year) and is expected to impact nearly all FinOps practitioners.

E.1.1. Key terms

FinOps is performed by working iteratively on the Framework Capabilities through three phases (Foundation, 2025b) namely, Inform, Optimize and Operate, which are described below:

(1) The Inform phase involves identifying data sources for cloud cost, usage and efficiency data. This data is then used for budgeting, allocation, forecasting, analysis and reporting.

(2) The Optimize phase identifies opportunities to improve cloud efficiency using the data and capabilities developed in the Inform Phase.

(3) The Operate phase implements operationalizes FinOps using the data and capabilities developed in the Inform and Optimize phase.

Some of the common KPIs in the FinOps that are amenable to optimization (full set is listed in https://www.finops.org/wg/finops-kpis/) include

(1) Percentage Resource Utilization - This is the amount of resources utilized as a percentage of the the total capacity allocated.

(2) Total Unpredicted Variance of Spend - Measures the unpredicted variance of cost associated with CSP Cloud usage recorded over a given period of time.

(3) Auto-scaling Efficiency Rate - Maximum capacity cost of running workload to meet workload demand / Cost of running workload with auto-scaling to meet same workload demand.

(4) Forecast Accuracy Rate (Usage) - Compares forecasted vs. actual cloud usage (vCPUs, Memory, etc) over a specific period (e.g., day, month, quarter).

(5) Forecast Accuracy Rate (Spend)- This metric compares forecasted vs. actual cloud spend over a specific period (e.g., day, month, quarter).

(6) Percent of Compute Spend Covered by Commitment Discounts - Measures the percentage of compute cost (excluding Spot) covered by commitment discount for a specific time period.

(7) Percentage of Commitment Discount Waste - The percentage of commitments not applied to on-demand spend.

(8) Percent of Unused Resources - Measure of unused cloud resources, e.g., unattached/orphaned storage volumes, load balancers, EIPS, Network gateways, snapshots.

(9) Percentage of Unallocated Shared CSP Cloud Cost - This measurement refers to expenses that cannot be directly attributed to a specific project, team, or department within an organization.

(10) Percentage Variance of Budgeted vs. Forecasted CSP Cloud Spend - Measures the difference between budgeted costs and the forecasted costs for using CSP cloud services

(11) Effective Savings Rate Percentage - Actual Spend with Discounts / Equivalent Spend at On Demand Rate

(12) Percentage of CSP Cloud Costs that are Tagging Policy Compliant - Total Costs Associated with Tagging Policy Compliant CSP Cloud Resources During a Period of Time / Total CSP Cloud Costs During a Period of Time.

(13) Percent Storage on Frequent Access Tier - Number of GB in Standard (or "frequently accessed" tiers vs. total GBs stored)

(14) Percentage of Carbon Associated with Untagged CSP Cloud Resources

There are several analogous KPIs related to carbon footprints instead of dollar costs. In the current version of the bench our scenarios have used an alert based on variance in spend. The remaining KPIs offer a rich basis for formulating many additional scenarios.

E.2. Motivating Example and FinOps Scenarios

We have analyzed common FinOps scenarios in three main categories; incidents with cost alerts, data insights generation using natural language queries, and cost anomalies in cloud bills. Incident scenarios are triggered by a cost alert and require diagnosis of the root cause of the alert and remediation steps to clear the alert. Table 28 shows an exemplar budget overrun alert-driven incident with the steps for diagnosis and resolution. In this incident, FinOps practitioners were notified of an —unusual cost increase (>20% more than last week's average) —alert by OpenCost. The increase is mainly caused by the increase in replica counts as a result of an observed load spike. The autoscaler increased the number of replicas to serve the demand as expected. However, budget thresholds are not updated which causes false alerts. Agent finds out that the application is healthy and recommends updating budget alerts based on the new load level. Similarly, Table 29 demonstrates a sample scenario where a budget overrun alert has been generated due to increased replicas. However, in this scenario application services are scaled up significantly despite low utilization in containers. Agent finds out autoscaler scaled up the application at low utilization thresholds and analyze the autoscaler configuration. It detects low thresholds for scale up policy and recommends updating autoscaler accordingly.

FinOps practitioners need to analyze cloud bills as part of their daily tasks. However, existing reporting tools rely on predefined set of reports and analyzing data outside of these reports is not trivial. We defined new data insight generation scenarios based on the common KPIs listed in E.1. Each scenario starts with a natural language query and retrieves, processes, and summarizes the related data. Table 26 presents description of scenarios and the corresponding KPIs.

Anomaly detection and management (abbreviated as "AD") is a key FinOps capability that allows practitioners to continuously monitor cloud spend and flag unexpected spend events immediately. The lifecycle of cloud cost anomaly detection includes cost monitoring, forecasting, identifying drift, outliers and anomalies, identifying root causes, and defining mitigation or resolution strategies. Under AD, we consider two FinOps scenarios - anomaly detection and anomaly ranking. Due to the complex topology of FinOps business dimensions and tags that could be mapped to a hierarchical structure, identifying the subset of dimensions where anomalies exist, their temporal impact, persistence and relevance to the user, an agentic solution is beneficial in utilizing the FinOps knowledge base and data to perform the task of anomaly detection and prioritizing them.

The anomaly detection scenario takes in a natural language query to first gather the relevant data from a database, subject to a time period of interest, and provided values for other relevant dimensions. From here, the data may need to be aggregated (e.g.: hourly or daily) according to the query. Next, an anomaly detection tool is employed to identify anomalies. The output of the tool is summarized by the agent and provided as the final LLM output.

Similarly, in the case of anomaly ranking, a user-specified scoring of anomalies is provided as part of the natural language query. Example scoring mechanisms include normalized product of anomalous cost and the duration of the anomaly. In this case, the agent is expected to gather the relevant data from the database, perform aggregation, pass it to the anomaly detection tool, compute an importance score for every identified anomaly and rank order anomalies as per the criteria specified in the query. Since the agent does not accurately identify all anomalies (that is, precision <

Table 26: FinOps Data Insight Scenarios

Scenario	Natural Language Prompt	Related KPI
Scenario 1: Cost by resource type	I need a summary report on the total cost for services in compute and storage categories between 09-01-2024 and 09-30-2024. Please calculate the total cost for each category.	Workload Optimization and waste reduction
Scenario 2: Highest Cost Contributors	I want to find out which applications contribute the cost most between 09-01-2024 and 09-30-2024. Give me the list of top 3 applications with highest cost. Ignore costs without an application information.	Workload Optimization and waste reduction
Scenario 3: Cost Allocation Performance	What is the percentage of allocated/total cost ratio per cloud provider for September 2024. Consider a cost allocated if there is an application attribution. Ignore records with negative costs for the calculation.	Full Allocation of spending
Scenario 4: Peak/Average Cost Ratio	List top 5 applications with highest peak to average ratio in terms of daily total cost during September 2024. For cost calculation do not include negative costs. Calculate maximum daily cost to average daily cost ratio per application and give the list with application, peak cost, average cost, ratio, and peak cost date details.	Workload Optimization and waste reduction
Scenario 5: Cost Variance	Find the top 5 applications with highest daily cost variance between 09-01-2024 and 09-08-2024. Use population variance for ranking. Do not include records with negative costs.	Workload Optimization and waste reduction
Scenario 6: Commitment Discount Coverage	Cloud providers regularly adjust bills by providing credits to the accounts after the initial charges. Calculate the discount coverage for each sub account including tax. Find five sub accounts which have total bill grater than 0 and have the maximum discount percentage. Return sub account information and discount coverage percentage.	Rate Optimization

1.0 and recall < 1.0), comparing ranking performed by the agent to the ground truth ranking, using techniques such as Normalized Discounted Cumulative Gain (NDCG) is not viable. Hence, we rely on comparing the importance score computed by the agent to the ground importance score, identifying the relative rank of an anomaly and computing the fraction of the anomalies with the correct relative rank. This is called the "rank score". Table 27 presents description of scenarios and the corresponding KPIs.

E.3. ITBench Architecture for Constructing FinOps Scenarios

For alert-driven scenarios, we have extensively leveraged the set up that we established for the SRE scenarios. We employ OpenCost to monitor costs and raise an alert when the predefined budget and efficiency thresholds are crossed. OpenCost is an opensource tool which distributes the cost of a virtual machine/node to the Kubernetes deployments running on it based on the allocated resources. It selects higher of utilization or request numbers of each container and distributes the cost using a load distribution policy. In our experiments, we have forced a custom pricing model which mainly includes hourly CPU cost rate for a single core and memory cost rate per 1 GB memory. Other pricing components such as networking cost and spot instance pricing are not the main scope of our evaluated scenarios. Thus, we have included negligible costs for these components. We mimic real world scenarios as explained in SRE scenarios by including cost variation alerts.

For data insights and anomaly detection use cases, we have leveraged sample FinOps data from FinOps Foundation (foc) in FOCUS format. Dataset includes more than 5 million cost records of 107 accounts for the services of AWS, Microsoft Azure, Oracle, and GCP. We provide natural language to SQL tool to retrieve the data from the database. The provided tool is responsible for linting the SQL code generated by the agent and if the code is syntactically correct, it executes the SQL query and return query results to the agent. Similarly, for anomaly detection scenarios, agent is responsible to generate the proper SQL code to retrieve the related data based on given natural language prompt. Then, the agent is responsible to execute anomaly detection algorithms on the retrieved data and summarize the results.

E.4. Evaluation

We have evaluated agents for each FinOps scenario category independently. For alert driven incident scenarios, we have used the same evaluation framework used in SRE-bench by including OpenCost and cost alerts in the scenarios. Data insights scenarios are evaluated based on the retrieved data accuracy and total token count consumed by the agent.

E.4.1. Evaluation Metrics

For alert driven incident scenarios, we evaluate each LLM-based agent for two scenarios: (i) diagnosis and (ii) mitigation. Data insights scenarios are evaluated based on accuracy of the retrieved results.

Anomaly detection scenarios are explicitly evaluated on

Table 27: FinOps AD Scenarios

Scenario	Natural Language Prompt	Related KPI
Scenario 1: Anomaly identification	Generate a detailed report consisting of the distinct 'ChargePeriodStart' values with corresponding anomalies in 'BilledCost' during 09-01-2024 and 09-30-2024 for the service 'Amazon Elastic Compute Cloud' and column 'PricingUnit' being equal to 'Hours'. Summarize the number of anomalies found.	Workload Optimization and waste reduction
Scenario 2: Anomaly ranking	Please create a report with a rank ordered list of the hours with anomalies in billed cost during 09-01-2024 and 09-30-2024 for the Amazon Elastic Compute Cloud service category after filtering on column PricingUnit being equal to 'Hours'. Compute an importance score for each anomaly based on the cost and the duration for which the anomaly persisted. Finally, rank these anomalies in the descending order of their importance score.	Workload Optimization and waste reduction

Table 28: Optimization Use Case: Increased Cost Alert - Increasing Demand

Optimization Use Case	Details
Triggering Alert	Cost increase alert for an application "Foo".
Summary	Cost alert was seen on application "Foo". The increase was 20% higher than the expected budget. Investigations reveal that the application is healthy and cost increase is caused by load increase. Client budget is flexible and thus cost alert is updated accordingly to increase the threshold. Long term fix requires additional check in CI/CD pipeline and possible automation of budget adjustments.
Time to detection	7 days (Current practices to observe utilization metrics for cost analysis
Time to diagnose	60 minutes
Time to mitigate	15 minutes
Event Type	An alert is generated to show there is more than 20% increase in expected cost.
Cost Overrun	20% increase in cost
Diagnosis Steps	• Checked infrastructure size changes. e.g. Increase in replica counts in Kubernetes cluster. • Found replica counts are greater than historical average. • Checked whether there is a legit increase in application load. • Found there is a stable increase in the application load and the cost increase is acceptable. • Checked budget constraints of the application. • Found application budget is flexible for scaling up.
Resolution Plan	• Calculated the new cost alert thresholds. • Update cost alert budget thresholds to accommodate new stable load level to prevent false alerts.
Long Term Improvements	Created playbooks to ensure such handling such adjustments are automated.

the agent's performance in identifying anomalies within a time period of interest at a granularity (hourly, daily etc.) of interest for a data subset defined by additional business dimensions such as department, cost center, application, region etc. Given the nature of the sub-tasks, the agent's performance is implicitly evaluated in terms of its ability to gather the right dataset based on the prompt and to produce the solution obtained from the anomaly detection algorithm as the final LLM output. F1-scores are computed to evaluate the performance of the agent on the anomaly detection scenario. For ranking anomalies, we compute the rank score of the true positive anomalies, as identified by the agent, with the correct importance score, as per the importance definition specified by the user in the prompt. Misinterpretation of the prompt or hallunications by the LLM will lower the agent's performance.

Diagnosis. The agent is evaluated on diagnosis by its ability to provide accurate *root cause* of budget alerts. Diagnosis includes the analysis steps that the agent follows to identify the root cause of the budget alerts. Agent performance in diagnosis is measured using pass@1 scores that indicate the accuracy of the root cause analysis provided to the SREs.

Mitigation. Mitigation involves recommending resolution steps for incidents to clear the alerts and optimizing the cost and efficiency of the application. We evaluated agents with the success rate using the pass@1 score and using proximity scores to analyze the performance of agents to achieve the optimal cost and efficiency metrics defined in Section E.4.2. Both diagnosis and mitigation evaluations

Table 29: Optimization Use Case: Increased Cost Alert - Faulty Auto-scaler

Optimization Use Case	Details
Triggering Alert	Cost increase alert for an application "Foo".
Summary	Cost alert was seen on application "Foo". The increase was 20% higher than the expected budget. Investigations reveal that auto scaler was misconfigured. SREs manually updated the auto scaling configuration by changing the scale up policy. Long term fix requires additional check in CI/CD pipeline.
Time to detection	7 days (Current practices to observe utilization metrics for cost analysis
Time to diagnose	60 minutes
Time to mitigate	15 minutes
Event Type	An alert is generated to show there is more than 20% increase in expected cost.
Cost Overrun	20% increase in cost
Diagnosis Steps	• Checked infrastructure size anomalies, e.g., increase in replica counts in Kubernetes cluster. • Found replica counts are greater than historical average. • Checked whether there is a legit increase in application load. • Found an increase in load and pending containers. • Checked utilization of containers • Found low utilization • Checked autoscaler for scaling policy • Found low threshold for scale up rules
Resolution Plan	• Configure auto scaler to update scaling policies. • Manually delete extra replicas.
Long Term Improvements	Created playbooks to ensure such misconfigurations do not happen for future deployments.

can be expanded in the future to automatically measure the performance of agents using LLM-as-a-judge similar to SRE-bench evaluations.

Data Insights. Agents are evaluated in the accuracy of retrieved results using pass@1 scores. Partially correct answers despite generating similar SQL codes to the ground truth are considered failed for the scenario. Efficiency of the agents are evaluated by calculating token utilization.

Anomaly Detection. Agents are evaluated for their ability to identify anomalies by using F1 score.

Anomaly Ranking. For ranking of anomalies, we first compute anomaly importance score I_a for an anomaly a as: $\frac{C_a \times \delta_a}{max_{a \in A}(C_a \times \delta_a)}$, where A represents the set of anomalies, C_a represents the cumulative billing cost associated with an anomaly a, and δ_a represents the duration for which anomaly a persisted. Next, we compute the rank score of the anomalies, that were correctly identified by the agent, also having the correctly computed importance score.

E.4.2. Metric definitions

Metrics for Alert Driven Incident Scenarios.
pass@1. We use the same metric defined in Section C.6.2 for both diagnosis and mitigation steps recommended by the agents.

Proximity metrics. To analyze the performance of agents in achieving optimal cost and efficiency, we defined proximity scores for hourly CPU cost, hourly memory cost, workload CPU efficiency, and workload memory efficiency. Proximity scores indicates the performance by calculating the proportional absolute difference between observed and optimal values for the measured cost or efficiency metric. We subtract the proportional value from 1 such that having proximity score of 1 indicates achieving the optimal performance. Proximity scores are calculates as follows:

$$\text{proximity\_}i = 1 - \frac{|\text{observed\_}i - \text{optimal}|}{\text{optimal}}$$

where i represents the experiment trial, $observed$ is the retrieved value for the measured metric, and $optimal$ is the value given in the ground truth for the same field.

Hourly CPU cost: The cost of allocating a single CPU core for an hour. We used custom pricing policy through opencost to ensure the ground truth values are not affected by changes in pricing between different cloud providers and on-premise deployments.

Hourly Memory cost: The cost of allocating 1 GB of memory. It also uses custom pricing policy.

Workload CPU efficiency: We take the average CPU efficiency of all containers of the application. We calculate CPU efficiency by dividing the CPU utilization of a container by the request amount. For instance, if the CPU request is 100m in the deployment configuration but the container uses 50m, the efficiency is calculated as 50.

Workload memory efficiency: It is calculated in the same way with CPU efficiency by dividing the utilization by requested memory amount.

We take the arithmetic mean of all runs for each metric and calculates the standard deviation. We present our results in Table 6.

Metrics for Data Insight Scenarios.
pass@1. We have compared the generated results with ground truth values for the given prompts. Only requested fields in the prompt are evaluated. The results with identical values with ground truth data are considered a "Pass" and all other results are evaluated as a "Fail".

Token Utilization. Agents require different number of iterations until reaching the final answer. For each run, we calculated total number of tokens during an experiment run for all iterations. Token utilization indicates how efficiently the agent optimizes between iterations and finalize its thoughts. This is used in evaluating the agent's performance for the anomaly detection scenarios as well.

Metrics for AD Scenarios.
F1 score. Treating anomalies identified by an anomaly detection algorithm as the ground truth data, we compute the F1 score to account for both the precision and recall of the anomalies identified by the agent.

Rank score. This denotes the fraction of the true positive anomalies, as identified by the agent, with the correct importance score, as per the importance definition specified by the user in the prompt.

E.5. Results

Table 30 presents the evaluation results for data insights scenarios. It shows that larger models perform better in both accuracy and average token utilization. The token utilization is mainly depending on the performance of updating the generated SQL queries based on natural language to SQL tool responses. Larger models understands tool feedback better and update code accordingly. However, all models largely failed to generate the correct queries for scenarios that require multiple data processing steps, as can be seen in the results of Scenarios 4, 5 and 6. We found data hallucination, making up function or column names based on prompt, and ignoring database version as common problems for all agents.

The results for the anomaly detection scenarios are contained in table 31. The first observation is that once again larger models, GPT-4o in this case, outperforms the smaller Granite-3.1-8b-instruct and Llama-3.1-8b-instruct. An exception to this is the Llama-3.3-70b-instruct where the agent was unable to translate the natural language query accurately to an SQL query. It also misinterpreted the query and thought that it was expected to detect anomalies without accessing the AD tool provided to it. Due to this, the columns required by the AD tool were not part of the SQL query due to which the AD tool failed to run. The end result of this misstep was that the anomalies identified by the agent were 100% incorrect. Since the anomaly ranking scenario needed anomalies to be identified first, the agent failed totally in those scenarios as well when using this LLM model. For the remaining 3 models, during data query as well as during summarizing the output of the AD tool, we noticed that the agent hallucinated, thereby lowering both the F1-score and the rank score metrics.

E.6. Example Trajectories

Data insights and anomaly detection scenarios require understanding of the prompt, retrieving data, and processing it. In our evaluations, we have found that agents commonly hallucinate about the column names in the table, which results in execution error in SQL queries. The performance of the agents in correcting these errors in the following iterations vary significantly. Similarly, agents are unable to consider the version of the database server to use the proper function names or syntax.

FinOps issues firing alerts are highly connected to used infrastructure and deployment configurations and policies for Kubernetes deployments. Incidents at infrastructure or configuration can trigger cost alerts. Agents commonly need to use the same tools as SRE-agent to understand the status of underlying infrastructure and deployment configuration of the application to identify the root cause of a FinOps concern. Similarly, to investigate budget variations accurately, they need analyzing load and utilization patterns to avoid diverging to irrelevant resolution recommendation. Figure 29 shows a trajectory for Scenario 37 where agent starts retrieving the alerts and observing CPU hourly cost has exceeded the budget threshold. It continues checking deployment details using NL2Kubectl tool and retrieve replica counts for each deployment. Due to high number of replicas in adservice, it considers scaling down the adservice deployment. Missing utilization checks and not analyzing the deployment details caused agent to an inaccurate diagnosis and led to scaling down recommendation despite the high load for the application. In a correct trajectory, an agent would analyze the utilization and decides the high load in

Table 30: Evaluation of Data Insights Scenarios: Pass@1 %

FinOps/Agent: Models	Scenario 1		Scenario 2		Scenario 3		Scenario 4		Scenario 5		Scenario 6	
	Pass@1	#Token	Pass@1	#Token	Pass@1	#Token	Pass@1	#Token	Pass@1	#Token	Pass@1	#Token
granite-3.1-8B-instruct	0	511096	100	10304	0	10210	0	5778	0	413810	0	9514
llama-3.1-8B-instruct	0	90424	0	76178	0	99769	0	131882	0	69447	0	60531
llama-3.3-70B-instruct	0	7347	100	6410	100	8107	0	8029	0	6350	0	7666
gpt-4o	50	6232	100	6415	50	8909	0	60973	0	6772	0	83378

Note: "# Token is the average token utilization over ten runs of each scenario.
(Pass@1 values are shown as percentage.)

Table 31: Evaluation of AD scenarios

FinOps/Agent: Models	Anomaly Detection		Anomaly Ranking	
	F1-score	#Token	Rank score	#Token
granite-3.1-8B-instruct	0.4 ± 0.07	22024	0.3 ± 0.0	205042
llama-3.1-8B-instruct	0.4 ± 0.03	14461	0.4 ± 0.0	166614
llama-3.3-70B-instruct	0	-	0	-
gpt-4o	0.6 ± 0.0	550	0.5 ± 0.0	164313

the system and would recommend changing the budget alert thresholds if total budget allocation allows or analyze the deployments which could be scaled down without hurting the application performance.

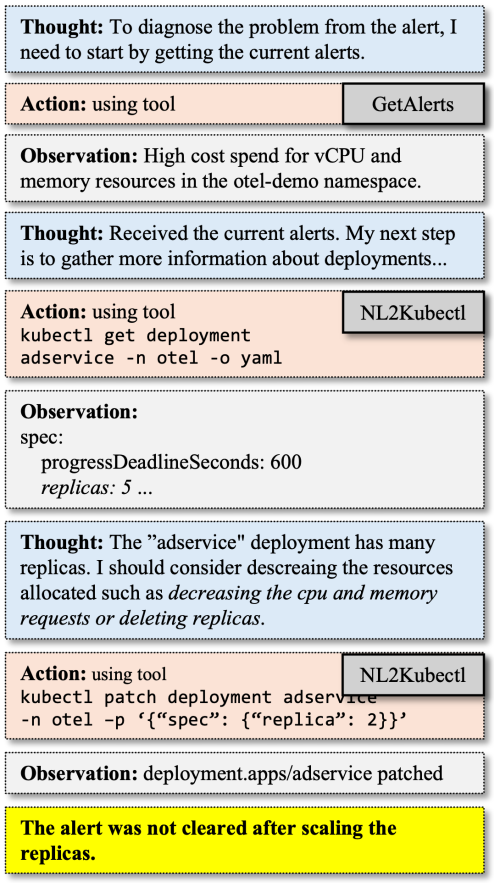


Figure 29: Sample Trajectory of unusual cost variation use case.

Table 32: Evaluation of Alert Driven FinOps scenarios.

Models	Diagnosis	Mitigation				
	pass@1 (%) ↑	pass@1 (%) ↑	Proximity to Optimal CPU Cost ↑	Proximity to Optimal Memory Cost ↑	Proximity to Optimal CPU Efficiency ↑	Proximity to Optimal Memory Efficiency ↑
granite-3.1-8B-instruct	0	0	0.47 ± 0.01	0.48 ± 0.06	0.53 ± 0.04	0.94 ± 0.01
llama-3.1-8B-instruct	0	0	**0.49** ± 0.01	0.46 ± 0.07	0.56 ± 0.08	0.96 ± 0.02
llama-3.3-70B-instruct	16.6	0	0.47 ± 0.01	0.49 ± 0.05	0.53 ± 0.03	0.96 ± 0.02
gpt-4o	**33**	0	0.48 ± 0.01	0.51 ± 0.02	**0.63** ± 0.07	0.92 ± 0.08

pass@1 values are shown as percentages. Proximity values shows how close the observed values to optimal values. One represents achieving optimal and any deviations from 1 represents sub-optimal performance.

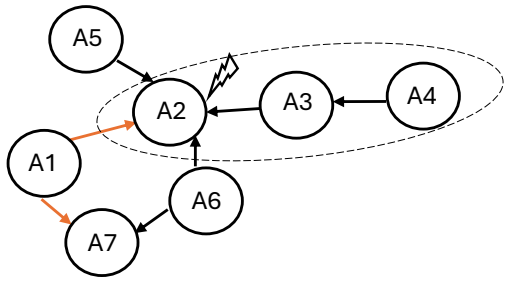


Figure 30: NTAM sample illustration. Node A2 is the ground truth root cause. Orange arrows represent edges of 'owns' type and black arrows represent edges of 'calls' type. Dotted area represents ground truth fault propagation chain.

F. Normalized Topology-aware Match

Existing work for root cause analysis has used general metrics including lexical and semantic metrics like BLEU, ROUGE-L, and BERTScore (Ahmed et al., 2023b), and metrics like F1, ranked list evaluation metrics like hit rate, etc.(Zhu et al., 2024; Chen et al., 2024c). However, these metrics are not accurate and granular enough. For example, if the root cause is "adservice pod", "adservice deployment" is a better prediction than "recommendation pod" but lexical metrics do not reflect that. Similarly, rank-evaluation metrics only capture whether the ground truth is in the predicted list or not. There is a need to develop domain-specific metrics that accurately measure the quality of the predicted root cause and fault propagation chains.

To this end, we propose **NTAM (Normalized Topology-Aware Match)**, inspired by:

- **Distance-based metrics**, rewarding predictions that are "closer" in a topology graph to the ground-truth entities.
- **Information-retrieval** concepts such as BM25 (Fang et al., 2011), where features that appear very commonly carry lower weight. By analogy, our scoring function down-weights highly connected nodes (i.e., large subtree), so that nodes that appear in many possible paths are less rewarded.

In the following sections, we describe NTAM that can be used both for scoring the predicted fault propagation chain and fault localization.

F.1. Notation

We first introduce notation used in NTAM.

F.1.1. Ground Truth

The ground truth G is composed of a set of P **unordered fault propagation chains** (FPCs):

$$G = \{\, FP_p \mid p \in \{1, \ldots, P\}\}.$$

Each chain FP_p contains I fault entities $\{g_1, \ldots, g_I\}$.

Every entity g_i has a *level*, denoted $\text{level}(g_i) = k$, which describes its position in the propagation chain ($k \in \{1, \ldots, K\}$). Typically, each chain has a single **root cause** entity at level K. But, in general, each unordered fault propagation chain in the ground truth can have U root cause entities.
$R_p = \{g_u\}$, , $u \in \{1, \ldots, U\}$ For example, in Figure 30, A2 is root cause and A2, A3, A4 is the propagation chain.

We also define an **entity importance** measure, $GI(g_i)$, to capture that not all entities are equally informative:

$$GI(g_i) = \left(\frac{K+1}{K-k+1}\right)^{\gamma},$$

where $k = \text{level}(g_i)$, and γ is a positive parameter. Entities closer to the root cause (higher k) get larger importance.

F.1.2. Predicted Output

Similarly, the model output O is a set of Q unordered fault propagation chains:

$$O = \{\, FP'_q \mid q \in \{1, \ldots, Q\}\}.$$

Each predicted chain FP'_q consists of J fault entities $\{o_1, \ldots, o_J\}$. The agent also identifies V root cause entities, per propagation chain.
$R'_q = \{o_v\}$, $v \in \{1, \ldots, V\}$

F.1.3. Distance

Let T be a topology graph whose nodes are the entities in our system. We define a distance $\mathbf{TD}(\mathbf{g_i}, \mathbf{o_j})$ that accounts for:

- The type of each edge along that path. In a typical Kubernetes-based system, as illustrated in Figure 30, we define two main edge types:
  - owns (e.g., Deployment owns ReplicaSet, ReplicaSet owns Pod),
  - calls (indicating an actual invocation or request from one microservice to another).

  Let cost(edge type) be a positive real-valued function. We define:

$$\text{cost(owns)} = 1, \quad \text{cost(calls)} = \zeta.$$

For example, in Figure 30, if A2 is the root cause pod and A1 is the corresponding deployment, intuitively, predicting A1 as the root cause is better than predicting A6 because A6 is part of another deployment, even though both are one hop away on the undirected graph. So, ζ is set to be greater than 1.

- The sub-tree size of each node along a shared path in the graph. We are inspired by information retrieval scoring functions, such as TF–IDF or BM25 (Zhai and Massung, 2016), where, terms that appear in many documents have lower *inverse document frequency* weight, as they are less discriminative. Analogously, if a predicted node has a large sub-tree size in the topology graph T (i.e. it is connected to many other nodes), it is less informative for pinpointing the fault's true location.

  For example, in Figure 30, predicting A5 is more discriminative than predicting A6 because A6 calls both A2 and A7, whereas, A5 calls only A2.

Combining the two factors above, we define *edge cost* between nodes x, y in T as follows:

$$\text{cost}(x, y) = \text{cost}(\text{edge\_type}(x, y)) \times \text{subtree\_size}(y)$$

Finally, $TD(g_i, o_j)$ is then defined as the shortest distance between g_i and o_j in weighted T, where weights are the edge costs.

To avoid penalizing exact matches, $\text{cost}(x, x) = 0$

In addition to $TD(g_i, o_j)$ between o_j and a *single* entity g_i in FP_p, we also define $\mathbf{FD}(\mathbf{FP_p}, \mathbf{o_j})$ as the average TD between o_j and every entity in FP_p.

These distance quantities guide our scoring function by rewarding predicted entities that lie close to the ground-truth nodes on the topology graph.

F.1.4. Task score

The overall task is to score the quality of O compared to G, defined by $\mathbf{S}(\mathbf{G}, \mathbf{O})$, where higher score means better quality.

Fault localization is a special case of this task, where only subsets R_p and R'_q are compared instead of entire FP_p and FP'_q .

F.2. Overall Score: NTAM

We now describe all components of NTAM below.

F.2.1. GI-Weighted Inverse Topology Distance

$$ITD_{wt}(g_i, o_j) \;=\; GI(g_i) \;\times\; \big(\frac{1}{TD(g_i, o_j) + 1}\big)^{\delta}$$

This is based on the following intuitions:

- A predicted entity closer to the ground-truth node on the topology should get a higher score.
- Further, the farther you move from a ground-truth entity, the relative penalization should *decrease*. In other words, to avoid a significantly large impact on the total score as the distance increases, the penalization should saturate. The above two are ensured by the 1/TD term. If $TD(g_i, o_j)$ is large (involving many high-subtree-size nodes or high-cost edges), then $ITD_{wt}(g_i, o_j)$ becomes small, reducing the credit for matching g_i with o_j. δ is a parameter to controlling how quickly penalties grow for larger distances
- Closer to *more important* ground-truth entities is better. This is ensured by the GI term.

F.2.2. GI-Weighted Inverse FD

$$IFD_{wt}(FP_p, o_j) \;=\; \frac{1}{I} \sum_{g_i \in FP_p} ITD_{wt}(g_i, o_j).$$

A predicted entity closer (on average) to *all* entities in the FPC is preferred. Analogous intuitions described for ITD_{wt} above hold for IFD_{wt} as well.

F.2.3. Scoring One Ground-Truth Chain vs. One Predicted Chain

Consider a single ground-truth chain FP_p of size I and a single predicted chain FP'_q of size J. We define:

$$S(FP_p, FP'_q) \;=\; \sum_{i=1}^{I} \frac{S1 \;\times\; S2}{\big(|J - I| + 1\big)^{c}}, \qquad (7)$$

where,

$S1 = \big(\max_J \big[\, ITD_{wt}(g_i, o_j)\big]\big)^{\alpha}$,
$S2 = \big(\max_J \big[\, IFD_{wt}(FP_p, o_j)\big]\big)^{\beta}$,
and $\alpha, \beta, c > 0$ are parameters.

We select the best matching predicted output entity based on distance to a single ground truth entity, and to the whole propagation chain. The denominator $\big(|J-I|+1\big)^{c}$ penalizes large mismatches in the chain length. This is based on the following two intuitions:

- If adding extra predicted entities that are farther away does not add information, they should hurt the score.
- Conversely, if adding extra predicted nodes *reduces* distance to ground-truth entities, do not over-penalize.

F.2.4. Combining All Chains: $S(G, O)$

Given all ground-truth chains $\{FP_1, \ldots, FP_P\}$ and predicted chains $\{FP'_1, \ldots, FP'_Q\}$, we define:

$$S(G,O) \;=\; \sum_{p=1}^{P} \frac{\max_Q \left[S(FP_p, FP'_q)\right]}{\big(|P-Q|+1\big)^d}.$$

Again, the best matching output propagation chain is selected for each ground truth propagation chain. The term $\big(|P-Q|+1\big)^d$ penalizes a mismatch in the total number of predicted chains vs. ground-truth. Same intuitions as described for scoring a single chain above hold.

Normalized TAM (NTAM). We normalize $S(G, O)$ to fall within $[0, 1]$ by dividing by the *ideal* score achieved when the prediction matches ground truth exactly:

$$NTAM \;=\; \frac{S(G,O)}{S(G,G)}.$$

Thus, $NTAM = 1$ indicates a perfect match.

F.3. Fault Localization: FL (NTAM)

For *fault localization*, only the *root cause* entities are considered. Let R_p denote the root-cause(s) in ground-truth chain FP_p, and R'_q the root-cause(s) in predicted chain FP'_q. Adapting Equation (7) to only these subsets:

$$S(R_p, R'_q) \;=\; \sum_{u=1}^{U} \frac{S3 \;\times\; S4}{\big(|U-V|+1\big)^c},$$

where,

$S3 = \big(\max_V[\,ITD_{wt}(g_u, o_v)]\big)^{\alpha}$,
$S4 = \big(\max_V[\,IFD_{wt}(FP_p, o_v)]\big)^{\beta}$.

Summing over all chains, then normalizing:

$$FL\;(NTAM) \;=\; \frac{\sum_{p=1}^{P} \frac{\max_Q S(R_p, R'_q)}{\big(|P-Q|+1\big)^d}}{\sum_{p=1}^{P} \max_P S(R_p, R_p)}.$$

This measures how well the model predicted the root-cause nodes alone.

F.4. Parameter Tuning

Key hyper-parameters used in NTAM are summarized in Table 33.

Table 33: Key NTAM hyper-parameters.

Symbol	Meaning	Chosen Value
α	Exponent for the max ITD term	1
β	Exponent for the max IFD term	1
γ	Power for entity importance GI	1
δ	Distance penalization exponent	0.289
c	Chain-length mismatch penalty exponent	0.5
d	Number-of-chains mismatch penalty exponent	0.5
ζ	Ratio of cost of edge type calls wrt cost of edge type owns	2.1

In practice, one can use grid-search or a numeric optimization method (Taylor et al., 2006) to select the parameter values that best align with expert preferences. For example, we could measure how well the ordering induced by NTAM matches expert judgments, using standard correlation coefficients such as Kendall Tau-b (Kendall, 1945).

Currently, all values except δ are not tuned and simply chosen to give uniform and equal weights to all terms. δ is determined analytically leveraging domain knowledge-based assumptions. As the distance increases by one hop, we want ITD (non-GI-weighted) to decrease by 50%. Further, on average, Kubernetes graphs have a subtree size of 10. Based on these two assumptions and $\alpha = 1$, we get $\delta = 0.289$.