Building AI Agents for Autonomous Clouds: Challenges and Design Principles

Manish Shetty[2], Yinfang Chen[3], Gagan Somashekar[1], Minghua Ma[1], Yogesh Simmhan[4], Xuchao Zhang[1], Jonathan Mace[5], Dax Vandevoorde[6], Pedro Las-Casas[1], Shachee Mishra Gupta[1], Suman Nath[5], Chetan Bansal[1], Saravan Rajmohan[1]
[1]Microsoft, [2]University of California, Berkeley, [3]University of Illinois Urbana-Champaign, [4]Indian Institue of Science, [5]Microsoft Research, [6]Agnes Scott College

ABSTRACT

The rapid growth in the use of Large Language Models (LLMs) and AI Agents as part of software development and deployment is revolutionizing the information technology landscape. While code generation receives significant attention, a higher-impact application lies in using agents for the operational resilience of cloud services, which currently require significant human effort and domain knowledge. There is a growing interest in AI for IT Operations (AIOps), which aims to automate complex operational tasks, like fault localization and root cause analysis, reducing human intervention and customer impact. However, achieving the vision of autonomous and self-healing clouds through AIOps is hampered by the lack of standardized frameworks for building, evaluating, and improving AIOps agents. This vision paper lays the groundwork for such a framework by framing the requirements and then discussing design decisions that satisfy them. We also propose AIOpsLab, a prototype implementation leveraging agent-cloud-interface that orchestrates an application, injects real-time faults using chaos engineering, and interfaces with an agent to localize and resolve the faults. We report promising results and lay the groundwork to build a modular and robust framework for building, evaluating, and improving agents for autonomous clouds.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**;
• **Computing methodologies → Artificial intelligence**.

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11
https://doi.org/10.1145/3698038.3698525

KEYWORDS

Reliability, Autonomous Clouds, Large Language Models

ACM Reference Format:
Manish Shetty[2], Yinfang Chen[3], Gagan Somashekar[1], Minghua Ma[1], Yogesh Simmhan[4], Xuchao Zhang[1], Jonathan Mace[5], Dax Vandevoorde[6], Pedro Las-Casas[1], Shachee Mishra Gupta[1], Suman Nath[5], Chetan Bansal[1], Saravan Rajmohan[1]. 2024. Building AI Agents for Autonomous Clouds: Challenges and Design Principles. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3698038.3698525

1 INTRODUCTION

IT applications and services for enterprises and web-scale companies are becoming more complex to develop, deploy, and maintain. The broad adoption of the micro-services pattern to compose complex applications and serverless deployment models hosted on the cloud has arguably simplified application development and scaling. But this has also introduced more moving parts that make their operations, reliability, fault diagnosis and recovery even harder [30]. Cloud services have set the expectation of five-9s of availability, and missing it can affect customer satisfaction. At the same time, failures in the clouds are becoming more frequent and with significant impact [74, 84, 92]. For instance, for one major recent outage, the estimated cost for one hour of service downtime for Amazon was approximately $100 million [92].

Site Reliability Engineers (SRE) and DevOps engineers are tasked with deployment and operational maintenance of cloud services [80]. They typically follow four steps when rectifying faults: fault detection, triaging, localization, and mitigation. This scope can be broader when we include proactive fault predictions and other preemptive maintenance. Many tools help with monitoring, anomaly detection, incident triage, root cause analysis, etc. [13, 94]. However, SREs face information overload and complex decision-making in operating large-scale cloud environments. E.g., a large-scale study of outages in Microsoft Teams found that root-causing and mitigation require significant manual effort, context awareness and technical expertise [31]. Hence, the need for

AIOps Agents, which can automatically detect, localize and mitigate faults with minimal human intervention, is becoming critical. These AIOps agents are key to achieve the vision of Autonomous Clouds.

While the concept of self-healing clouds has been proposed earlier [24, 56], the emerging adoption of *AIOps*, i.e., the use of AI to support IT Operations, is making this a reality [29, 37, 57, 64, 81, 82, 93, 100, 102, 105]. Here, AI algorithms and agents leverage the monitoring infrastructure to rapidly and efficiently track the health, identify failures and mitigate them within the operational environment [72]. More recently, agents are able to converse with Large Language Models (LLMs) using an observe–thought–action pattern to localize problems, explore possible causes, and enact solutions to achieve recovery in a semi-autonomous manner [22, 48, 79, 95, 103]. We are at the cusp of AIOps agents being able to independently carry out these tasks in a matter of minutes within production environments, compared to several hours taken even by experts [62, 91].

The design, development, evaluation and iterative improvement of AIOps agents are challenging. While the adoption of AI has seen rapid progress for coding and programming [6, 95] due to the availability of platforms like WebArena [107], R2E [44] and benchmarks like HumanEval[21], LiveCodeBench [43], SWE-bench [47], the same is lacking for cloud operations. Existing tools address individual aspects of the AIOps lifecycle: observability and introspection tools [36, 78], application suites [28, 53], chaos engineering and fault injection tools [16, 17, 71], agent-computer interfaces [95], etc., but do not cohesively integrate them. Moreover, recent efforts on leveraging AIOps agents for cloud operations like RCAgent [91], and Zhang et al. [103] use proprietary services and datasets. Other prior works use frameworks specific to the solutions that they are building [81], or *ad hoc* and static benchmarks and metrics [60] that fail to capture the dynamic nature of real-world cloud services. Furthermore, current approaches do not agree on standard metrics or a standard taxonomy for operational tasks.

This calls for a standardised and principled framework for building, testing, comparing and improving AIOps agents. Such a framework would help AIOps agents improve in several ways. First, it would help study and standardise agent-cloud interactions, allowing researchers to focus on core agent capabilities. Analysing these interaction trajectories would help improve tools and, more generally, the *agent-cloud interface*. Secondly, like advancements in Code LLMs, introducing an execution-first environment with runtime feedback for cloud operations provides an opportunity to collect rich traces to train more domain-specific foundational models. Lastly, it also creates a "gym "-like training environment where agents and their core models can learn online [103].

In this paper, we draw upon our experiences from developing AIOps agents at Microsoft to make these contributions:

(1) We envision the requirements for a holistic framework to enable the design, development, evaluation, and enhancement of AIOps agents that, additionally, serves the purpose of reproducible, standardized, interoperable and scalable benchmarks. We also suggest key design decisions to build it (§ 3).

(2) We describe a prototype framework, AIOpsLab, that adopts this design to combine workload and fault generators to mimic production incidents and an agent-cloud interface for orchestrating the service operation lifecycle (§ 4). This is the foundation for our ongoing work on a more comprehensive framework.

(3) We report a case study on using this preliminary framework to evaluate an LLM agent for two critical operations tasks: Fault Detection and Mitigation (§ 5).

2 BACKGROUND AND RELATED WORK

Next, we identify gaps in existing work for evaluating AIOps tools in a standardized, realistic and reliable manner.

Fault Mitigation Lifecycle. A typical incident goes through four stages: (1) *Detection* [63, 98, 99]: When an anomalous system behaviour is observed, an alert is raised by monitors or users of the service (internal engineers or external customers) and reported to the Incident Management System (IcM). (2) *Triaging* [13, 19, 20]: After the detection, the incident is notified to the On-Call Engineers (OCEs) to begin investigation and then assigned to the most appropriate engineering team. (3) *Diagnosis* [22, 62, 76, 103]: Engineers inspect different aspects of the incident and have several rounds communication to identify the root cause. (4) *Mitigation* [5, 45, 77]: Several actions are taken by the team to mitigate the incident and to restore service health. The outcome is also updated postmortem in the IcM.

AIOps Benchmarks. Several benchmarks have been proposed to evaluate AIOps at various stages of the software lifecycle. For instance, ADBench [34] and AnomalyBench [42] evaluate anomaly detection, and LogParser [110] evaluates log parsing. However, these benchmarks focus only on single aspects of incident management. More recently, LLMs have powered a new wave of AIOps tools and autonomous agents, forcing evaluations to evolve. General benchmarks [4, 59] like AgentBench task LLMs to operate as autonomous agents in diverse environments, solving problems across domains like operating systems and databases.

That said, specialized AIOps evaluations have received less attention. OpsEval [60] attempts to address this with a question-answer-based evaluation for LLMs, but it disconnects from real-world operational challenges that require

complex debugging, code understanding, and multi-step fault resolution. Our proposed work on AIOpsLab bridges this gap. We believe using real services, workloads, and faults to create problems and enable executing concrete actions (e.g., run commands) is necessary for the reliable evaluation of state-of-the-art AIOps solutions, and to enhance them.

Application Benchmark Suites. Relevant to this work are benchmark suites for cloud applications [27, 35, 51, 53, 90]. Ferdman et al. [27] presented `Cloudsuite` to study the architectural implications, and TailBench [51] proposes a methodology to analyze the performance of web servers and database services. Further, the emergence of microservices has prompted recent work to study their characteristics and requirements [52, 83, 87, 108]. The popular DeathstarBench [28] differentiates from these studies by focusing on diverse large-scale applications with tens of unique microservices, allowing studying effects that only emerge at scale, such as network contention and cascading Quality of Service (QoS) violations due to dependencies between tiers. Beyond static application suites [28, 109], BluePrint [10] provides the ability to reconfigure applications and iteratively generate variants. We integrate with such application workloads to enable benchmarking of AIOps solutions.

Chaos Engineering and Fault Injection. Prior work developed fault injection techniques aimed at applications and distributed backend systems, including storage and data processing [2, 7, 8, 12, 15, 18, 23, 33, 39, 49, 54, 61, 65, 67, 70, 73, 85, 101]. However, these existing techniques fall short in providing a generic, one-click fault generator that can be universally applied across various microservices. The limitations are multifaceted: many rely on application or domain-specific knowledge to create policies and oracles, making them unsuitable for the diverse requirements of AIOps [18, 32, 54, 61, 70, 85]. Others offer mechanisms without automated policies or oracles beyond simple crashes, requiring developers to manually implement complex functionalities [1, 8, 23, 33, 39, 40, 49, 66, 69, 73, 75]. Additionally, fault injections at a single level (e.g., HTTP) do not adequately expose root-causes of failures due to: 1) their coarse-grained nature, 2) challenge of constructing meaningful error objects, and 3) a lack of consideration for dependencies between microservices [23, 26, 41, 46, 67, 68, 101].

3 AIOpsLab: A PRINCIPLED VISION

In this section, we first discuss the principles that we believe should guide the envisioned framework (3.1), followed by a discussion of the design choices that can lead to such a framework (3.2).

3.1 Requirements and Principles

(R1) Modular Design for applications, workloads, and agents. An effective evaluation system should seamlessly incorporate existing and evolving resources on application workloads, fault models, and resilience techniques for fault detection and resolution. It must be flexible, supporting easy integration of new components through standard interfaces. This allows for varied use cases, including enterprise and end-user assessments of fault and recovery strategies, AIOps research on new agents using established error models and workloads, and red-team simulations of complex faults on synthetic applications to test mitigation effectiveness.

(R2) Flexible Interface for human, digital, and AI agents. The framework must provide diverse interfaces to agents, spanning the unique requirements of humans to LLM-based agents. Humans might use a web interface for log review and command execution, digital agents need APIs for integration, and conversational LLM agents require prompt-based interaction for requests and responses.

(R3) Scalability at various operational and temporal scales. Operating at diverse spatial (single VM to clusters) and temporal scales (minutes to days) is paramount to make the framework amenable to different use cases and resource availability, e.g., from large enterprises with complex deployments and a wide fault surface to software engineering course assignment with a tiny scenario. Operating across time scales also allows faults that gradually emerge (e.g., memory leaks) or occur periodically to be detected, predicted, and preemptively mitigated.

(R4) Reproducible Setup for reliable measurements. An evaluation framework needs consistent and, ideally, automated deployments for reproducible and standardized assessment of mitigation strategies. Challenges may arise from non-deterministic elements in applications or faults, which should stem from external models, not the framework itself. While evaluation metrics can differ depending on the user or context, the framework should offer default objective metrics (e.g., accuracy, time to mitigate) and support the creation of more intricate measures with the data it provides.

(R5) Versatility in operating environments. The operating environments may vary based on the needs of the user. Chaos engineering [71], for instance, advocates for fault injections directly in production environments, necessitating a framework capable of integrating with live applications. Alternatively, a sandboxed deployment with a simpler variant of production applications may be preferred. Synthetic and emulated systems [11, 14] can also help evaluate large-scale what-if scenarios.

(R6) Comprehensive and Cross-Layer fault support. The framework should enable the introduction of faults across

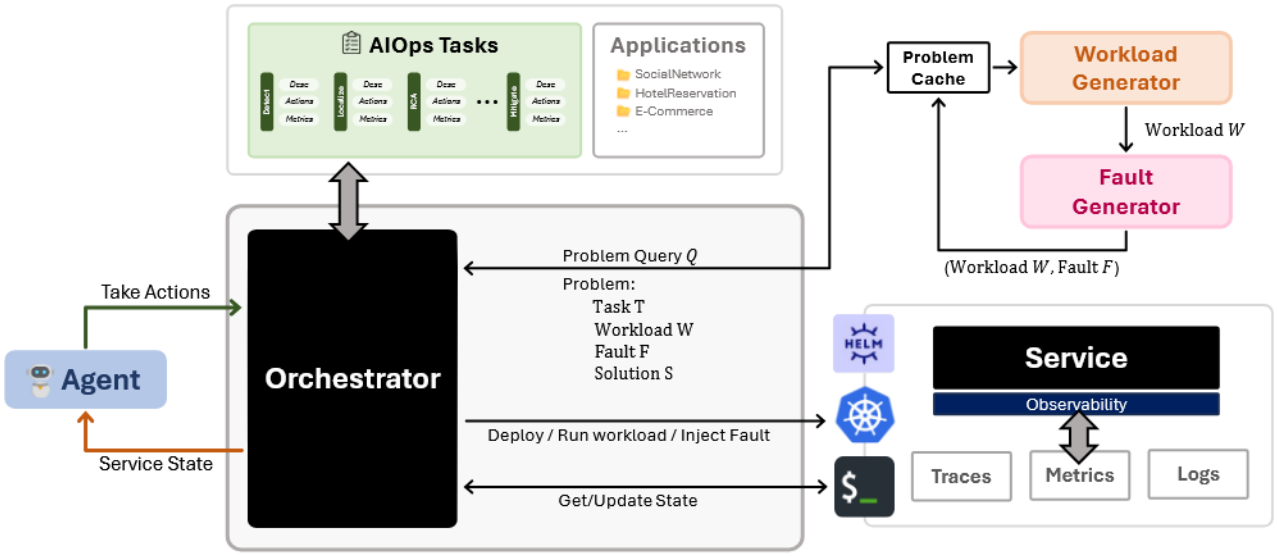

Figure 1: System Architecture of AIOpsLab. The Orchestrator coordinates interactions between various system elements and serves as the Agent-Cloud-Interface (ACI). Agents engage with the Orchestrator to solve tasks, receiving a problem description, instructions, and relevant APIs. The Orchestrator generates diverse problems using the Workload and Fault Generators, injecting these into applications it can deploy. The deployed service has observability at multiple layers, providing telemetry, traces, and logs. The Orchestrator communicates with the service and the cloud using several tools such as `Kubernetes`, `Helm`, and even a `Shell`. Agents act via the Orchestrator, which executes them and updates the service's state. The Orchestrator evaluates the final solution using predefined metrics for the task.

the entire stack, including hardware, network, OS, middleware, application, and external services. It must offer capabilities to simulate realistic faults inspired by real-world production incidents, which can cause cascading effects across distributed components, as well as synthetic faults for assessing potential future scenarios.

(R7) Diverse and realistic workload conditions. Workloads across domains have different burstiness, interarrival times, and parallelism [88]. An effective benchmarking framework should allow for generating workloads that reflect these characteristics rather than a 'one-size-fits-all' approach. Current benchmarking often overfits well-known publicly available workload traces due to the scarcity of realistic workloads available [9]. Realism is essential for effectively testing and improving AI agents, as evidenced by practitioners at CompanyX who faced delays in AI agent deployment due to the difficulty in obtaining genuine user interaction and traffic patterns.

(R8) Coverage of the operations lifecycle. The incident management process can have diverse goals and scopes, and the framework should support the different stages that correspond to them, like fault detection, root-cause analysis, and mitigation. It should allow proactive strategies to predict and preempt failures and reactive strategies to detect and correct errors.

(R9) Sufficient Observability into applications. Adequate visibility into the different aspects of the system and application environment should be available to detect faults and their impact. It includes interfaces to access logs, telemetry, traces, KPIs, and documentation such as incident reports, standard procedures, etc. Tools to explore configuration files and source code may be beneficial in acquiring the necessary context for localization and mitigation.

(R10) Adequate controls for agent operations. Fault correction may require modifying configuration files or restarting VMs or services. Even fault detection may require the agent to run test suites to gain more visibility into the possible cause. The framework should enable such actions by having hooks into different layers.

We note that these principles are intentionally comprehensive to cover the vast requirements of AIOps. However, in practice there can be variants of this envisioned framework that prioritizes certain features, balancing implementation effort and reward. That said, we outline design choices that help users satisfy these principles.

3.2 Design Decisions

To address the requirements outlined in Section 3.1 (annotated below), we recommend several design choices to build AIOpsLab, a new standardized evaluation framework:

3.2.1 ***Bring your Services Alive.*** We propose evaluating with live services, that are designed and implemented using different architectures at various scales ***(R3)***, and capable of operating in various contexts, from sandbox to production ***(R5)***. Real-world cloud services present complex behaviours, challenging AIOps agents with variability in workload patterns, diversity of faults, and intricacies of service dependencies. This approach closely mirrors the operational and reliability challenges faced in production systems, making the benchmark directly applicable to practitioners' needs. Further, we choose automation tools like `Helm` [38] and Blueprint [10] for repeatable and consistent setups ***(R4)*** and interfaces to extend to new services, preserving its applicability and rigour ***(R1)***. This approach provides a comprehensive, realistic, and relevant evaluation platform to advance AIOps research and practice.

3.2.2 ***Real Faults, Real Challenges.*** We recommend incorporating dynamic workload and fault generators to simulate real-world conditions accurately. These generators are designed to produce a wide range of realistic traffic patterns and operational challenges ***(R7)***, from typical user behaviour to peak loads and various fault scenarios, including kernel failures, network issues, and configuration bugs ***(R6)***. This approach not only tests adaptability and robustness but also mitigates the risk of "training data contamination" for LLM-powered tools. A key implication of this choice is that the state of a faulty service (say telemetry or logs) is only relevant and observed in the context of an injected fault – making it publicly unavailable to seep into training data.

3.2.3 ***Evaluate Across the Board.*** The combination of live services (§ 3.2.1) and real faults (§ 3.2.2) allow us to reliably evaluate the impact of agents on various AIOps tasks and performance dimensions. Here, one can use faults independently or in conjunction to create benchmark problems for agents. Notably, one can use a fault (e.g., network misconfiguration) to evaluate tasks across the board–from detection to resolution ***(R8)***. Evaluation involves quantitative dimensions, such as performance, time, resource usage, dollar cost, and other metrics beyond accuracy [50]. Furthermore, to reliably compare agents, we emphasize qualitative evaluation of their traces by a human or LLM-as-a-Judge [106].

3.2.4 ***Orchestrate the Agent-Cloud-Interface.*** Typically, service engineers operate cloud environments with various programming (e.g. APIs, database queries) and user interfaces (incident portals). Existing UI to the cloud are designed only for a human, and not amenable for LLMs and agents. E.g., humans reliably ignore extra information while the same can distract the context and harm performance for agents [58]. Inspired by human-computer interaction (HCI), Yang et al. [95] introduce *agent-computer-interface*, finding that agents can similarly benefit from better-designed interfaces for coding tasks. We posit the same for AIOps and envision an ***Agent-Cloud-Interface* (ACI)**. The ACI is an Orchestrator between the agent and the cloud (Figure 1) which specifies both the actions available and how the service's state is conveyed back to the agent as the observation of its actions ***(R2)***. It simplifies the action space into a concise list of APIs, each documented to ensure that agents make meaningful progress towards objectives ***(R10)***. Also, the Orchestrator takes actions on behalf of the agent and returns high-quality feedback (e.g., outputs, error messages, etc.).

3.2.5 ***Abstract Environments, not agents.*** Two sides to a real-time evaluation are the agent and the environment. For AIOps, the agent can be a DevOps engineer or an AIOps tool (e.g., LLM-powered agent). The environment is a deployed application (e.g., `SocialNetwork`) that the user interacts with. Here, we suggest providing abstractions for the environment, not the agent. This choice maximizes flexibility in implementing and evaluating various kinds of tools and agents ***(R2)***. Consequently, AIOpsLab provides sufficient information (task description, available APIs/actions, and additional instructions) to solve a problem in the benchmark.

3.2.6 ***Observe Everything, Everywhere, All at Once.*** Observability captures a system's internal states from its external outputs. It traditionally includes (1) *traces* detailing the end-to-end request paths through distributed entities; (2) *application logs* as textual records of runtime operations; and (3) *metrics* monitoring component health. We suggest an observability layer to collect not only canonical telemetry data but also other system indicators, such as cluster information, including logs, running status, and configurations ***(R9)***. But increased observability can overwhelm AIOps tools with data volume and complex data types. Therefore, we offer flexible APIs for users to select the specific information they need, ensuring tailored and comprehensive observability.

Summary. In summary, these design decisions prioritize creating a framework that is:

(1) **Realistic**: By using live services, workloads, and faults that mirror real-world operational challenges.
(2) **Scalable**: Through dynamic workload and fault generators that create new problem scenarios at varying scales.
(3) **Reliable**: Evaluating tasks and performance dimension across the operations lifecycle.
(4) **Observable**: Through rich telemetry data and actual service interactions, ensuring reliable evaluation.

(5) **Flexible**: With the Agent-Cloud-Interface (ACI) that supports plugging a diverse range of tools.
(6) **Extensible**: With the ability to easily incorporate new services, workloads, and faults.

4 SYSTEM ARCHITECTURE

This section expands on how the decisions discussed in § 3.2 lead to an prototype system architecture for AIOpsLab. Figure 1 shows the architecture consisting of 5 key pieces.

4.1 Orchestrator

AIOpsLab strictly separates the Agent and the Application Service using an intermediate Orchestrator. It provides several interfaces for other system parts to integrate and extend. First, it establishes a `session` with an Agent to share information about benchmark problems: (1) the problem description, (2) instructions (e.g., response format), and (3) available APIs to call as actions. As shown in Figure 2, the APIs are a set of documented tools, e.g., `get_logs`, `get_metrics`, `exec_shell`, designed to help the Agent solve a task. There are no restrictions on the Agent's implementation; the Orchestrator poses problems and polls it for the next action to perform given the previous result. Each action must be a valid API call, which the Orchestrator validates and carries out. The Orchestrator has privileged access to the deployment and can take arbitrary actions (e.g., scale-up, redeploy) using appropriate tools (e.g., `helm`, `kubectl`) to resolve problems on behalf of the Agent. Lastly, the Orchestrator calls workload and fault generators to create service disruptions, which serve as live benchmark problems. AIOpsLab provides additional APIs to extend to new services and generators.

4.2 Service

AIOpsLab abstracts a diverse set of services to reflect the variance in production environments. This includes live, running services implemented using various architectural principles, including microservices, serverless and monolithic. We also leverage open-sourced application suites such as DeathStarBench [28] as they provide artifacts, like source code and commit history, along with run-time telemetry. Adding tools like BluePrint [10] can help scale to other academic [25, 53, 83, 89, 97, 109] and production services [86] and also seamlessly deploy new variants of these services.

4.3 Workload Generator

The workload generator in AIOpsLab plays a crucial role by creating simulations of both faulty and normal scenarios. It receives specifications from the Orchestrator, such as the task, desired effects, scale, and duration. The generator can utilize a model trained on real production traces, to generate workloads that align with these specifications. Faulty scenarios may simulate conditions like resource exhaustion, exploit edge cases, or trigger cascading failures, inspired by real incidents. Normal scenarios mimic typical production patterns, such as daily activity cycles and multi-user interactions. When various characteristics (e.g., service calls, user distribution, arrival times) can lead to the desired effect, multiple workloads can be stored in the problem cache for use by the Orchestrator. In coordination with the Fault Generator (4.4), the workload generator can also create complex fault scenarios with workloads.

4.4 Fault Generator

AIOpsLab has a novel push-button fault generator designed for generic applicability across various cloud scenarios. Our approach integrates application and domain knowledge to create adaptable policies and "oracles" compatible with AIOps scenarios. This includes fine-grained fault injection capable of simulating complex failures inspired by production incidents. Additionally, it can inject faults at various system levels, exposing root causes while maintaining semantic integrity and considering interdependencies between cloud microservices. The fault injector's versatility can enhance the reliability and robustness of cloud systems by enabling thorough testing and evaluation of AIOps capabilities.

4.5 Observability

AIOpsLab is equipped with an extensible observability layer designed to provide comprehensive monitoring capabilities across various system layers for any AIOps tool. AIOpsLab collects a wide array of telemetry data, including (1) traces from Jaeger detailing the end-to-end paths of requests through distributed systems, (2) application logs formatted and recorded by Filebeat and Logstash, and (3) system metrics monitored by Prometheus. Additionally, AIOpsLab also captures lower-level system information such as `syscall` logs and cluster information. As mentioned, we handle the potential data overload through flexible APIs to tune the telemetry data relevant to the AIOps tools. While AIOpsLab collects several default metrics, e.g., CPU/memory/disk usage of containers and pods, users can select metrics they wish to monitor. Furthermore, AIOpsLab exposes interfaces to extend the observability layer to custom metrics, e.g., low-level syscalls or disk logs.

5 CASE STUDY AND INSIGHTS

As a proof-of-concept, we implement a prototype of AIOpsLab and evaluate an LLM agent on an AIOps incident mitigation task. Here, we aim to demonstrate the potential of AIOpsLab to standardize evaluation for AIOps tools and uncover novel insights.

Orchestrator

The service you are working with today is described below:
App: SocialNetwork
Namespace: test-social-network
...

Problem Description. You will begin by analyzing the `compose-post-service`'s state and telemetry, and submit a solution that mitigates any detected anomalies. You can use the following APIs:

`get_logs`: get Kubectl logs for a service.
Args:
service (str): The name of the service.
namespace (str): The namespace of the service.

`get_metrics`: get Prometheus metrics for a service.
...

`get_traces`: get Jaeger traces for a service.
...

`exec_shell`: Execute any command in a predefined shell.
...

`Other Instructions`: (k lines)

Agent

Thought: To begin the investigation, I need to first review the recent logs to identify any errors, warnings, or unusual activity.

```
get_logs("compose-post-service", "test-social-network")
```

Observation

... Thrift: Mon Jul 8 21:16:34 2024 TSocket::open() connect() <Host: user-service Port: 9090>: Connection refused ...

Agent

Thought: The logs indicate that the 'compose-post-service' is unable to connect to the 'user-service' on port 9090. This could be due to the 'user-service' being down or misconfigured network settings.
...

Figure 2: Orchestrator shares APIs to be used as actions and an agent detects an anomaly in the logs.

⇄ (previous attempts)

Agent

Thought: The 'user-service' seems to be running correctly with no restarts, but the 'compose-post-service' is unable to connect to it. This might be due the service being misconfigured.

```
exec_shell("kubectl describe svc user-service -n ...")
```

Observation

Service: Name: user-service
Namespace: test-social-network
Labels: app.kubernetes.io/managed-by=Helm
Annotations: meta.helm.sh/release-name: social-network
meta.helm.sh/release-namespace: test-social-network
Selector: service=user-service
...
Port: 9090 9090/TCP
TargetPort: 9999/TCP
...

Agent

Thought: Mismatch in the ports. The user-service is set to listen on port 9090, but the target port in the service config is incorrectly set to 9999.

```
exec_shell("kubectl patch service user-service -n test-social-social-network --type='json' -p='[{"op":"replace", "path":"/spec/ports/0/targetPort", "value":9090}]'")
```

Observation

Error: namespaces "test-social-social-network" not found

Agent

Thought: Need to use the right namespace!

```
exec_shell("kubectl patch service user-service -n test-social-network --type='json' -p='[{"op":"replace", "path":"/spec/ports/0/targetPort", "value":9090}]'")
```

Figure 3: Agent identifying the root-cause and mitigating as a K8S misconfiguration fault.

Setup. We deploy the `SocialNetwork` application from DeathStarBench [28] on a Kubernetes cluster. We instantiate the fault generator to induce a realistic misconfiguration fault at the virtualization layer: the target port of a microservice is misconfigured, causing connectivity issues with other microservices. We generate traffic by using an exponential workload pattern from the `wrk` tool. We study a ReAct [96] agent with a GPT-4 [3] backend. ReAct is a popular LLM-agent paradigm that leverages interleaved Thought-Action traces, utilizing LLMs' reasoning and planning abilities.

Evaluation Goals. This case study aims to showcase several of the requirements in Section 3.1. We address ***(R4)*** by employing quantitative metrics (Success, Time-to-Detect (TTD), Time-to-Mitigate (TTM), and Efficiency) and qualitative analysis of the agent's actions. We demonstrate ***(R8)*** by covering multiple stages: detection, root-cause analysis, and mitigation. ***(R9)*** is ensured through application observability (logs, metrics, traces), and the well-defined APIs for the agent, including a `shell`, fulfill ***(R10)***.

Key Insights. Figure 2 illustrates the agent's trace while attempting to first detect a service anomaly. It then takes a series of actions, as shown in Figure 3, to identify the root cause and mitigate the fault. Overall, it successfully detected the problem in 14 secs (TTD) and mitigated it in 36 secs (TTM). We measured the average resource efficiency, and found that it takes 10–12 interactions costing around $0.25 Below, we distill key insights from this case study:

① Importance of Observability. The agent's ability to detect the misconfiguration fault hinged on the detailed telemetry data provided by AIOpsLab. For example, the agent identified repeated connection refusals in the logs, which made it hypothesize a potential misconfiguration. This highlights the critical role observability will play in AIOps in the future.

② AIOps needs Efficient Actions. We found that the efficiency of the available actions influenced the agent's performance. For instance, too few APIs limited its ability to explore solutions, while too many arguments for each API hindered its performance. Also, flexible APIs (like `exec_shell`) were pivotal to the agent balancing explore and exploit actions. This reiterates the importance of a well-designed Agent-Cloud-Interface (ACI).

③ Fault Injection Reflects Real-World Complexity. Interestingly, injecting simple faults (like a K8s misconfiguration) into a real service proves challenging problems for advanced models like GPT-4 [3], requiring 10+ interaction rounds to reach a solution. This demonstrates how automated fault generators could accelerate the creation of rigorous testbeds for reliable AIOps evaluation.

④ Incorporating Service Dependency Tools. The trace in Figure 2 demonstrates the agent's implicit understanding of service dependencies just via logs. While promising, for complex applications, the agent could spend significant time traversing the call graph with only partial views of the system [55]. Concurrent efforts in coding agents have shown the power of code dependency analysis for repository-level tasks [6, 44, 104]. We believe future research can similarly look at augmenting AIOps agents with tools for explicit service dependencies and impact analysis.

⑤ Error Handling goes a long way. In Figure 3, when the agent encountered a syntax error, it quickly corrected the mistake and retried the command. We even find that poor error messages hinder the agent's performance. This insight emphasizes the need for robust error propagation for actions throughout the operations and reliability lifecycle of systems.

6 CONCLUSION

In this vision paper, we have framed the requirements and design principles for a framework to build, test, compare and improve Agents used for the operational management of cloud services. We also discuss our prototype implementation, AIOpsLab, and report preliminary results. This forms the kernel for a more comprehensive framework that will address the key gap in standardized framework for building agents to help achieve autonomous clouds.

REFERENCES

[1] 2022. AWS Fault Injection Simulator. https://aws.amazon.com/fis/.

[2] 2022. Jepsen. https://jepsen.io/.

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[4] AgentOps-AI. 2024. agentops. https://github.com/AgentOps-AI/agentops.

[5] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)*.

[6] Aider. 2024. How aider scored SOTA 26.3% on SWE Bench Lite. https://aider.chat/2024/05/22/swe-bench-lite.html.

[7] Ramnatthan Alagappan, Aishwarya Ganesan, Yuvraj Patel, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. Correlated Crash Vulnerabilities. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*.

[8] Ahmed Alquraan, Hatem Takruri, Mohammed Alfatafta, and Samer Al-Kiswany. 2018. An Analysis of Network-Partitioning Failures in Cloud Systems. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*.

[9] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. 2018. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 533–546. https://www.usenix.org/conference/atc18/presentation/amvrosiadis

[10] Vaastav Anand, Deepak Garg, Antoine Kaufmann, and Jonathan Mace. 2023. Blueprint: A Toolchain for Highly-Reconfigurable Microservice Applications. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 482–497. https://doi.org/10.1145/3600006.3613138

[11] Shrey Baheti, Shreyas Badiger, and Yogesh Simmhan. 2021. VIoLET: An Emulation Environment for Validating IoT Deployments at Large Scales. *ACM Trans. Cyber-Phys. Syst.* 5, 3, Article 25 (jul 2021), 39 pages. https://doi.org/10.1145/3446346

[12] Radu Banabic and George Candea. 2012. Fast Black-Box Testing of System Recovery Code. In *Proceedings of the 7th European Conference on Computer Systems (EuroSys'12)*.

[13] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. Decaf: Diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*.

[14] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience* 41, 1 (2011), 23–50.

[15] Marco Canini, Daniele Venzano, Peter Perešíni, Dejan Kostić, and Jennifer Rexford. 2012. A NICE Way to Test OpenFlow Applications. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*.

[16] Chaos Mesh Community. [n. d.]. ChaosMesh. https://chaos-mesh.org/. Accessed: 2024-07-08.

[17] ChaosBlade Team. [n. d.]. ChaosBlade. https://github.com/chaosblade-io/chaosblade. Accessed: 2024-07-08.

[18] Haicheng Chen, Wensheng Dou, Dong Wang, and Feng Qin. 2020. CoFI: Consistency-Guided Fault Injection for Cloud Systems. In *Proceedings of the 35th ACM/IEEE International Conference on Automated Software Engineering (ASE'20)*.

[19] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'19)*.

[20] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)*.

[21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[22] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 674–688. https://doi.org/10.1145/3627703.3629553

[23] Maria Christakis, Patrick Emmisberger, Patrice Godefroid, and Peter Müller. 2017. A General Framework for Dynamic Stub Injection. In *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*.

[24] Yuanshun Dai, Yanping Xiang, and Gewei Zhang. 2009. Self-healing and Hybrid Diagnosis in Cloud Computing. In *Cloud Computing*, Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg.

[25] Andrea Detti, Ludovico Funari, and Luca Petrucci. 2023. μBench: An Open-Source Factory of Benchmark Microservice Applications. *IEEE Transactions on Parallel and Distributed Systems* 34, 3 (2023), 968–980. https://doi.org/10.1109/TPDS.2023.3236447

[26] Envoy Docs. 2022. Envoy Fault Injection. https://www.envoyproxy.io/docs/envoy/latest/api-v3/extensions/filters/http/fault/v3/fault.proto.

[27] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices* 47, 4 (2012), 37–48.

[28] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 3–18.

[29] Vaibhav Ganatra, Anjaly Parayil, Supriyo Ghosh, Yu Kang, Minghua Ma, Chetan Bansal, Suman Nath, and Jonathan Mace. 2023. Detection Is Better Than Cure: A Cloud Incidents Perspective. In *Proceedings of the 31st Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[30] Peter Garraghan, Renyu Yang, Zhenyu Wen, Alexander Romanovsky, Jie Xu, Rajkumar Buyya, and Rajiv Ranjan. 2018. Emergent Failures: Rethinking Cloud Reliability at Scale. *IEEE Cloud Computing* 5, 5 (2018), 12–21. https://doi.org/10.1109/MCC.2018.053711662

[31] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*. 126–141.

[32] Jiawei Tyler Gu, Xudong Sun, Wentao Zhang, Yuxuan Jiang, Chen Wang, Mandana Vaziri, Owolabi Legunsen, and Tianyin Xu. 2023. Acto: Automatic End-to-End Testing for Operation Correctness of Cloud System Management. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP'23)*.

[33] Haryadi S. Gunawi, Thanh Do, Pallavi Joshi, Peter Alvaro, Joseph M. Hellerstein, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Koushik Sen, and Dhruba Borthakur. 2011. Fate and Destini: A Framework for Cloud Recovery Testing. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*.

[34] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. 2022. Adbench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems* 35 (2022), 32142–32159.

[35] Johann Hauswald, Michael A Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, et al. 2015. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. 223–238.

[36] Shilin He, Botao Feng, Liqun Li, Xu Zhang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2023. STEAM: Observability-Preserving Trace Sampling. Association for Computing Machinery, New York, NY, USA, 1750–1761. https://doi.org/10.1145/3611643.3613881

[37] Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, et al. 2022. An empirical study of log analysis at Microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[38] Helm. 2024. *Helm: The package manager for Kubernetes*.

[39] Victor Heorhiadi, Shriram Rajagopalan, Hani Jamjoom, Michael K Reiter, and Vyas Sekar. 2016. Gremlin: Systematic Resilience Testing of Microservices. In *Proceedings of the IEEE 36th International Conference on Distributed Computing Systems (ICDCS'16)*.

[40] Galen Hunt and Doug Brubacher. 1999. Detours: Binary Interception of Win32 Functions. In *Proceedings of the 3rd USENIX Windows NT Symposium*.

[41] Istio Docs. 2022. Istio Fault Injection. https://istio.io/latest/docs/tasks/traffic-management/fault-injection/.

[42] Vincent Jacob, Fei Song, Arnaud Stiegler, Yanlei Diao, and Nesime Tatbul. 2020. Anomalybench: An open benchmark for explainable anomaly detection. *CoRR abs/2010.05073* (2020).

[43] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *arXiv preprint arXiv:2403.07974* (2024).

[44] Naman Jain, Manish Shetty, Tianjun Zhang, King Han, Koushik Sen, and Ion Stoica. 2024. R2E: Turning any Github Repository into a Programming Agent Environment. In *Forty-first International Conference on Machine Learning*.

[45] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering*

Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20).

[46] Zu-Ming Jiang, Jia-Ju Bai, Kangjie Lu, and Shi-Min Hu. 2020. Fuzzing Error Handling Code using Context-Sensitive Software Fault Injection. In *Proceedings of the 29th USENIX Security Symposium*.

[47] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

[48] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, et al. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[49] Xiaoen Ju, Livio Soares, Kang G. Shin, Kyung Dong Ryu, and Dilma Da Silva. 2013. On Fault Resilience of OpenStack. In *Proceedings of the 12th ACM Symposium on Cloud Computing (SOCC'13)*.

[50] Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. 2024. AI Agents That Matter. *arXiv preprint arXiv:2407.01502* (2024).

[51] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.

[52] Nane Kratzke and Peter-Christian Quint. 2016. ppbench-A Visualizing Network Benchmark for Microservices.. In *CLOSER (2)*. 223–231.

[53] Varad Kulkarni et al. 2024. XFBench: A Cross-Cloud Benchmark Suite for Evaluating FaaS Workflow Platforms. In *24th IEEE/ACM international Symposium on Cluster, Cloud and Internet Computing (CCGRID)*.

[54] Tanakorn Leesatapornwongsa, Mingzhe Hao, Pallavi Joshi, Jeffrey F. Lukman, and Haryadi S. Gunawi. 2014. SAMC: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*.

[55] Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, et al. 2021. Fighting the fog of war: Automated incident detection for cloud systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 131–146.

[56] Wenrui Li, Pengcheng Zhang, and Zhongxue Yang. 2012. A Framework for Self-Healing Service Compositions in Cloud Computing Environments. In *2012 IEEE 19th International Conference on Web Services*. https://doi.org/10.1109/ICWS.2012.109

[57] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service*.

[58] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.

[59] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688* (2023).

[60] Yuhe Liu, Changhua Pei, Longlong Xu, Bohan Chen, Mingze Sun, Zhirui Zhang, Yongqian Sun, Shenglin Zhang, Kun Wang, Haiming Zhang, et al. 2023. OpsEval: A Comprehensive Task-Oriented AIOps Benchmark for Large Language Models. *arXiv preprint arXiv:2310.07637* (2023).

[61] Jie Lu, Chen Liu, Lian Li, Xiaobing Feng, Feng Tan, Jun Yang, and Liang You. 2019. CrashTuner: Detecting Crash-Recovery Bugs in Cloud Systems via Meta-Info Analysis. In *Proceedings of the 26th ACM Symposium on Operating System Principles (SOSP'19)*.

[62] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment (VLDB'20)* (2020).

[63] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. 2021. Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems. In *2021 USENIX Annual Technical Conference (ATC'21)*.

[64] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. 2018. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE'18)*.

[65] Rupak Majumdar and Filip Niksic. 2018. Why is Random Testing Effective for Partition Tolerance Bugs?. In *Proceedings of the 45th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'18)*.

[66] Paul D. Marinescu, Radu Banabic, and George Candea. 2010. An Extensible Technique for High-Precision Testing of Recovery Code. In *Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC'10)*.

[67] Paul D Marinescu and George Candea. 2009. LFI: A Practical and General Library-Level Fault Injector. In *Proceedings of the 39th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'09)*.

[68] Paul D. Marinescu and George Candea. 2011. Efficient Testing of Recovery Code Using Fault Injection. *ACM Transactions on Computer Systems (TOCS)* 29, 4 (Dec. 2011), 1–38.

[69] Christopher S. Meiklejohn, Andrea Estrada, Yiwen Song, Heather Miller, and Rohan Padhye. 2021. Service-Level Fault Injection Testing. In *Proceedings of the 2013 ACM Symposium on Cloud Computing (SOCC'21)*.

[70] Jayashree Mohan, Ashlie Martinez, Soujanya Ponnapalli, Pandian Raju, and Vijay Chidambaram. 2018. Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*.

[71] Netflix. [n. d.]. ChaosMonkey. https://github.com/Netflix/chaosmonkey. Accessed: 2024-07-08.

[72] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2021. A survey of aiops methods for failure management. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12, 6 (2021), 1–45.

[73] Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Samer Al Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2014. All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*.

[74] Reuters. 2023. Microsoft cloud outage hits users around the world. (2023). https://www.cnn.com/2023/01/25/tech/microsoft-cloud-outage-worldwide-trnd/index.html

[75] Patrick Reynolds, Charles Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, and Amin Vahdat. 2006. Pip: Detecting the Unexpected in Distributed Systems. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI'06)*.

[76] Manish Shetty, Chetan Bansal, Suman Nath, Sean Bowles, Henry Wang, Ozgur Arman, and Siamak Ahari. 2022. DeepAnalyze: learning to localize crashes at scale. In *Proceedings of the 44th International Conference on Software Engineering*. 549–560.

[77] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. 2022. AutoTSG: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)*.

[78] Jesper Simonsson, Long Zhang, Brice Morin, Benoit Baudry, and Martin Monperrus. 2021. Observability and chaos engineering on system calls for containerized applications in docker. *Future Generation Computer Systems* 122 (2021), 117–129.

[79] Vikramank Y. Singh, Kapil Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Balakrishnan Narayanaswamy, Rashmi Gangadharaiah, and Tim Kraska. 2024. Panda: Performance Debugging for Databases using LLM Agents. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*.

[80] Amith Singhee and Praveen Jayachandran. 2023. From Clouds to Hybrid Clouds. *ACM India Minigraphs* (2023).

[81] Gagan Somashekar, Anurag Dutt, Mainak Adak, Tania Lorido Botran, and Anshul Gandhi. 2024. GAMMA: Graph Neural Network-Based Multi-Bottleneck Localization for Microservices Applications. In *Proceedings of the ACM on Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 3085–3095. https://doi.org/10.1145/3589334.3645665

[82] Gagan Somashekar, Anurag Dutt, Rohith Vaddavalli, Sai Bhargav Varanasi, and Anshul Gandhi. 2022. B-MEG: Bottlenecked-Microservices Extraction Using Graph Neural Networks. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering* (Bejing, China) *(ICPE '22)*. Association for Computing Machinery, New York, NY, USA, 7–11. https://doi.org/10.1145/3491204.3527494

[83] Akshitha Sriraman and Thomas F Wenisch. 2018. μ suite: a benchmark suite for microservices. In *2018 ieee international symposium on workload characterization (iiswc)*. IEEE, 1–12.

[84] Laura Stevens. 2017. Amazon Finds the Cause of Its AWS Outage: A Typo. (2017). https://www.wsj.com/articles/amazon-finds-the-cause-of-its-aws-outage-a-typo-1488490506

[85] Xudong Sun, Wenqing Luo, Jiawei Tyler Gu, Aishwarya Ganesan, Ramnatthan Alagappan, Michael Gasch, Lalith Suresh, and Tianyin Xu. 2022. Automatic Reliability Testing for Cluster Management Controllers. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*.

[86] The Overleaf Team. 2024. Overleaf: An open-source online real-time collaborative LaTeX editor. https://github.com/overleaf/overleaf.

[87] Takanori Ueda, Takuya Nakaike, and Moriyoshi Ohara. 2016. Workload characterization for microservices. In *2016 IEEE international symposium on workload characterization (IISWC)*. IEEE, 1–10.

[88] Laurens Versluis, Roland Mathá, Sacheendra Talluri, Tim Hegeman, Radu Prodan, Ewa Deelman, and Alexandru Iosup. 2019. The Workflow Trace Archive: Open-Access Data from Public and Private Computing Infrastructures - Technical Report. *CoRR* abs/1906.07471 (2019). arXiv:1906.07471 http://arxiv.org/abs/1906.07471

[89] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems* (Milwaukee, WI, USA) *(MASCOTS '18)*.

[90] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. 2014. Bigdatabench: A big data benchmark suite from internet services. In *2014 IEEE 20th international symposium on high performance computer architecture (HPCA)*. IEEE, 488–499.

[91] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2023. Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. *arXiv preprint arXiv:2310.16340* (2023).

[92] Sean Wolfe. 2018. Amazon's one hour of downtime on Prime Day may have cost it up to $100 million in lost sales. (2018). https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7

[93] Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. 2023. Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. In *Proceedings of the ACM Web Conference 2023*.

[94] Xiaohan Yan, Ken Hsieh, Yasitha Liyanage, Minghua Ma, Murali Chintalapati, Qingwei Lin, Yingnong Dang, and Dongmei Zhang. 2023. Aegis: Attribution of Control Plane Change Impact across Layers and Components for Cloud Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'23)*.

[95] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793* (2024).

[96] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

[97] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 30–44. https://doi.org/10.1145/3419111.3421280

[98] Jun Zeng, Zheng Leong Chua, Yinfang Chen, Kaihang Ji, Zhenkai Liang, and Jian Mao. 2021. WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics.. In *Network and Distributed System Security Symposium (NDSS'21)*.

[99] Jun Zeng, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. 2022. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE Symposium on Security and Privacy (S&P'22)*.

[100] Zhengran Zeng, Yuqun Zhang, Yong Xu, Minghua Ma, Bo Qiao, Wentao Zou, Qingjun Chen, Meng Zhang, Xu Zhang, Hongyu Zhang, Xuedong Gao, Hao Fan, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. TraceArk: Towards Actionable Performance Anomaly Alerting for Online Service Systems. In *To appear in Proc. of ICSE*.

[101] Pingyu Zhang and Sebastian Elbaum. 2012. Amplifying Tests to Validate Exception Handling Code. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*.

[102] Qiao Zhang, Guo Yu, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, , Murali Chintalapati, Arvind Krishnamurthy, and Thomas Anderson. 2018. Deepview: Virtual Disk Failure Diagnosis and Pattern Detection for Azure. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*.

[103] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 2024. Automated Root Causing

of Cloud Incidents using In-Context Learning with GPT-4. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 266–277. https://doi.org/10.1145/3663529.3663846

[104] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. *arXiv preprint arXiv:2404.05427* (2024).

[105] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, et al. 2023. Robust Multimodal Failure Detection for Microservice Systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[106] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2024).

[107] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854* (2023). https://webarena.dev

[108] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 323–324.

[109] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 323–324. https://doi.org/10.1145/3183440.3194991

[110] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.