When LLMs Listen to Experts: Accurate Failure Diagnosis in Operating Systems

Yongxin Zhao
Nankai University
Tianjin, China

Shenglin Zhang*
Nankai University
Tianjin, China

Yuxin Sun
Nankai University
Tianjin, China

Wenwei Gu
Nankai University
Tianjin, China

Yongqian Sun
Nankai University
Tianjin, China

Luping Wang
Alibaba Group
Hangzhou, China

Li Shi
Alibaba Group
Hangzhou, China

Cheng Huang
Alibaba Group
Hangzhou, China

Guodong Yang
Alibaba Group
Hangzhou, China

Liping Zhang
Alibaba Group
Hangzhou, China

Dan Pei
Tsinghua University
Beijing, China

Abstract

Efficient failure diagnosis is critical to maintaining the stability and reliability of operating systems (OSes) in modern industrial environments. In practice, manual failure analysis by on-call engineers (OCEs) faces increasing challenges due to the growing complexity of OS failures, while recent automated diagnosis methods often suffer from low explainability, limiting their practical adoption. Large Language Models (LLMs) hold promise for advancing automated failure diagnosis through their sophisticated reasoning and language generation capabilities. However, traditional LLM-based solutions struggle to integrate domain knowledge and lack the effective interaction mechanisms required for industrial troubleshooting. To address these practical challenges, we present *OScope*, an automated, explainable failure diagnosis framework powered by LLMs. *OScope* leverages historical failure cases to enhance the semantic understanding of anomalies, enabling precise retrieval of relevant troubleshooting guides. The framework further structures the diagnosis process using standard operating procedure (SOP) templates, supporting step-by-step verification and correction with corresponding SOP documents. Importantly, *OScope* facilitates human-in-the-loop collaboration, allowing OCEs to interact with the system for report refinement and practical feedback. We have evaluated *OScope* on a real-world OS failure dataset collected from *Alibaba*. Results show that *OScope* achieves an *AC*@5 of 90%, significantly outperforming baseline methods and demonstrating high diagnostic value in production settings. The diagnostic reports generated by *OScope* have also received positive feedback from OCEs for readability and practical usefulness. Since deployment at *Alibaba*, *OScope* has substantially improved the efficiency of engineers in resolving failures, highlighting its tangible impact as a successful case of applying automated software engineering methods in industry.

CCS Concepts

• **Software and its engineering → Software maintenance tools**.

Keywords

Failure Diagnosis, Operating System, Large Language Model

*Shenglin Zhang is the corresponding author. Email: zhangsl@nankai.edu.cn

1 Introduction

Operating systems serve as an intermediary between user-facing applications and the underlying hardware, facilitating user interaction with infrastructure [12]. Operating system (OS) provides essential services such as resource management, process scheduling, input/output control, and security enforcement, thereby establishing a dependable execution environment for applications.

However, the growing complexity of modern OSes introduces substantial operational challenges. The intricate dependencies among system components can lead to degraded performance, increased failures, and reduced availability. The issues are further magnified in large-scale cloud-native environments, where an OS-level failure on a node may cascade across interconnected services, triggering widespread alerts and causing significant disruptions to service availability [46]. In extreme cases, such disruptions can result in considerable financial losses for service providers [27]. In this context, robust and intelligent troubleshooting mechanisms are indispensable for ensuring system stability. Timely and accurate failure diagnosis is critical for root cause identification, informed remediation planning, and accelerated system recovery, thereby minimizing application performance degradation and preserving system reliability [14].

Despite the increasing complexity and scale of modern OSes, practical fine-grained diagnostic tools for OS failures remain limited. As a result, accurate failure diagnosis continues to be highly labor-intensive and primarily depends on manual investigation by on-call engineers (OCEs). Our investigation of failure tickets at *Alibaba* reveals a mean time to diagnosis of approximately two hours, with more complex cases often requiring several days to resolve.

In recent years, numerous automated failure diagnosis techniques have been proposed, leveraging either unimodal [25, 30, 36, 42, 46, 47, 53] or multimodal [4, 17, 44, 49] data, particularly in microservice systems. These methods primarily focus on identifying anomalous patterns in observability data to localize system failures. However, most of these approaches merely output failure types or root causes without offering a transparent diagnostic process, resulting in limited explainability. The "black box" nature of such methods poses significant challenges for their adoption in real-world operational environments [45].

The rapid advancements in natural language processing, especially the emergence of large language models (LLMs), offer new opportunities for improving failure diagnosis [2, 7, 14, 15, 34, 48, 50]. While LLMs have demonstrated strong reasoning and representation capabilities, their practical application in failure diagnosis remains constrained by ongoing challenges.

Challenge 1: Deficiency of troubleshooting knowledge in LLM-based reasoning. Troubleshooting guides (TSGs) derived from historical failure cases are essential resources for failure diagnosis, offering documentation of initial mitigation procedures [2, 13]. By referencing relevant TSGs based on observed failure symptoms, OCEs can expedite the resolution process [13]. However, in cloud-native systems, the wide variety and high frequency of OS failures often result in TSGs being authored by multiple engineers with differing expertise and writing styles, which leads to inconsistencies in terminology, symptom descriptions, and procedural details. The heterogeneity results in diverse expressions of the same failure symptoms, posing significant challenges for traditional information retrieval methodologies and impeding the effective reuse of valuable diagnostic knowledge.

Challenge 2: Unreliable and Non-Actionable LLM Responses in Failure Diagnosis. In OS failure diagnosis, identical symptoms may arise from diverse underlying root causes, posing significant challenges for accurate failure localization. Moreover, most LLMs are pre-trained on general-purpose corpora and thus lack sufficient exposure to domain operational knowledge. The limitation is exacerbated by the probabilistic nature of LLMs, which often leads to inconsistent or ambiguous outputs [27], undermining their reliability and applicability in real-world diagnostic tasks. Consequently, general-purpose LLMs frequently fail to generate accurate and actionable diagnostic conclusions, necessitating repeated trial-and-error procedures and resulting in prolonged recovery times.

Challenge 3: Limitations in Engineer–LLM Interaction Mechanism. The interaction between engineers and the LLM also influences the effectiveness of LLM-based diagnostic frameworks. Upon receiving diagnostic reports, engineers may find certain output aspects ambiguous and seek clarification. Furthermore, LLM-generated reports sometimes provide excessive detail, which can obscure the most pertinent information and hinder efficient comprehension. Therefore, engineers often require the LLM to deliver targeted responses and distill essential insights, enhancing the readability of diagnostic reports. Consequently, optimizing interactive mechanisms between engineers and LLMs is critical to improve the clarity, relevance, and precision of automated failure diagnosis.

To address the aforementioned challenges, we propose *OScope*, a practical framework for failure diagnosis in OSes that harnesses the reasoning capabilities of LLMs. (1) To mitigate the inherent deficiency of troubleshooting knowledge in LLMs, we employ supervised fine-tuning on domain-specific data. In particular, anomalous features observed during ongoing failures are standardized and aligned with symptom descriptions in historical failure cases, enabling precise retrieval of relevant TSGs. (2) To further enhance the actionability of diagnostic reports, we introduce standard operating procedure (SOP) templates, which serve as structured scaffolds for generating initial diagnostic reports. The reports are subsequently segmented, each chunk being validated and enriched by retrieving contextually relevant SOPs, ensuring accuracy and actionability in the final recommendations. (3) To improve adaptability and foster human–system collaboration, we incorporate an OCE-LLM-in-the-loop mechanism that facilitates interactive refinement of diagnostic outputs. Such an iterative process enables OCEs to efficiently identify and mitigate failures with the support of diagnostic reports generated by *OScope*. In return, the feedback and domain expertise contributed by OCEs continuously refine and enhance the diagnostic performance of *OScope*.

Our main contributions are summarized as follows:

- We propose *OScope*, a novel LLM-based framework for automated failure diagnosis in OS, significantly improving OCEs' efficiency in identifying and mitigating system failures.
- *OScope* enhances the explainability, usability, and readability of diagnostic results by extracting anomalous features from multimodal data, integrating domain-specific operational knowledge, including TSGs and SOPs to enhance diagnostic reasoning, and enabling an interactive human-in-the-loop (HITL) diagnostic process.
- To validate the effectiveness of *OScope*, we conduct a comprehensive evaluation on a real-world dataset collected from *Alibaba*. The results demonstrate that, compared to baselines, *OScope* achieves the top 5 accuracy (*AC*@5) of 90%, representing a 20% improvement over the best-performing baseline. Furthermore, the diagnostic reports generated by *OScope* have garnered positive feedback from experienced OCEs for readability and practical usefulness. Moreover, during a three-month deployment within three teams in *Alibaba*, *OScope* successfully diagnosed critical failures reported by engineers and significantly improved diagnostic efficiency compared to manual troubleshooting approaches.

2 Preliminaries

2.1 Multimodal Observability Data

As illustrated in Figure 1, the assessment of OS state typically relies on three primary categories of observability data: metrics, logs, and stack traces. Each modality offers complementary perspectives on system behavior and plays a crucial role in failure diagnosis.

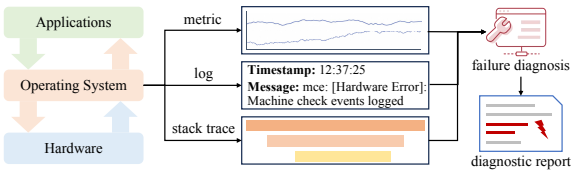


Figure 1: Multimodal observability data (*i.e.*, metrics, logs, and stack traces) are used to diagnose failure.

2.1.1 Metric. Metrics, often captured as time series data, offer rapid insights into system performance and resource usage [45]. Metrics are quantitative indicators used to monitor system resource utilization, workload performance, and kernel-level activities. Collected at regular intervals, these time series data are susceptible to performance fluctuations and can offer early signals of abnormal system states. For example, OCEs can identify potential resource exhaustion by analyzing trends in metrics. However, metric data may also contain misleading variations due to normal behaviors [18]. For instance, a temporary decrease in file cache utilization, which

may result from routine cache eviction, can appear anomalous in monitoring data but does not indicate a system failure.

2.1.2 Log. Logs provide rich, semi-structured textual descriptions of system events, capturing key execution information such as timestamps, log levels, and detailed messages [30]. The log details typically consist of static templates authored by developers to describe specific events, and dynamic variables that reflect real-time system state [9]. Logs are essential for understanding the state of the system [30, 46–48], and various techniques have been proposed for parsing log templates [3, 8, 26, 38]. However, excessive logging may introduce performance overhead, resulting in transient I/O bottlenecks that are often insufficiently captured due to constraints on logging granularity.

2.1.3 Stack Trace. Stack traces (aka call stacks) utilized for CPU profiling offer a hierarchical view of function call sequences, revealing the code-path ancestry at the time of an exception or during debugging sessions [6, 32]. They provide a precise snapshot of the runtime execution context, enabling deep insight into failure propagation paths. In modern OS, stack traces are particularly valuable for diagnosing critical issues such as kernel panics, segmentation failures, and deadlocks. Nonetheless, stack traces are limited in their ability to convey temporal trends in resource consumption, which are often critical for detecting performance degradation.

2.2 Manual Failure Diagnosis

Figure 2 shows the manual failure diagnosis workflow. After receiving a failure alert, OCEs initiate the diagnostic procedure by collecting multimodal data related to the failure. Subsequently, collected data undergoes a comprehensive analysis to identify and extract distinctive anomalous features that serve as diagnostic indicators of the failure. The features are utilized to perform similarity-based queries against a historical failure case repository, enabling the identification of analogous failures from previous operational experiences. By consulting the corresponding TSGs associated with historical cases, the OCEs perform a comparative analysis to infer potential root causes. Diagnostic actions are guided by SOPs to ensure that root cause localization adheres to prescribed and systematic procedures.

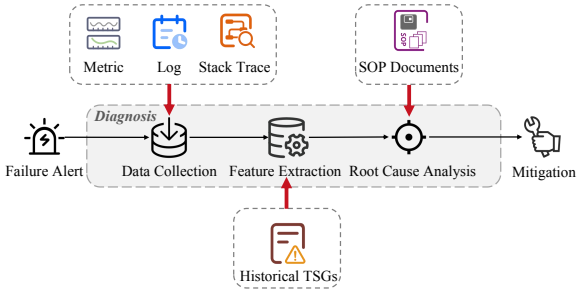


Figure 2: The workflow of the manual failure diagnosis.

3 Motivation

The rapid advancement of information technology has significantly increased the architectural and functional complexity of OSes, particularly within cloud-native environments. The heightened complexity has led to a proliferation of failure types. Concurrently, the explosion of heterogeneous monitoring data and system alerts has rendered manual diagnosis by OCEs increasingly labor-intensive and error-prone.

To address these challenges, recent research has explored automated failure diagnosis using machine learning and deep learning techniques, which analyze patterns in observability data to infer potential root causes [4, 17, 25, 30, 36, 42, 44, 46, 47, 49, 53]. While these methods demonstrate considerable promise, they tend to provide coarse-grained classifications or ranked lists of candidate causes, which lack explainability and actionable insights. As a result, OCEs may struggle to trust or effectively utilize these outputs during failure resolution.

LLMs offer a potential alternative given their strong reasoning and generative capabilities [2, 7, 14, 15, 34, 48, 50]. However, leveraging LLMs for failure diagnosis introduces two significant challenges: (1) how to integrate domain-specific knowledge, including TSGs and SOPs, effectively, and (2) how to enable effective interaction between OCEs and the diagnostic system.

3.1 How to Integrate Domain Knowledge?

Historical failure cases often contain descriptions of failure symptoms and corresponding TSGs [13], which serve as valuable troubleshooting resources during manual diagnosis. A straightforward approach is to retrieve relevant TSGs using keyword-based or vector-based retrieval methods. However, our empirical analysis of failure tickets in *Alibaba*'s production environment reveals considerable inconsistency in how different OCEs describe similar symptoms. These inconsistencies hinder the accurate retrieval of relevant cases and prevent effective knowledge transfer from historical TSGs.

To address this, *OScope* harnesses the semantic alignment capabilities of LLMs [54] to align current failure features with historical case symptoms, enabling more robust and accurate TSG retrieval. We conducted an empirical study to evaluate the effectiveness of the semantic alignment strategy, using a representative set of 20 failure cases sampled from *Alibaba*. Each case is paired with a ground-truth TSG, which domain experts review and validate as the correct resolution reference. We employ *AC@k*, quantifying the proportion of failure cases in which the ground-truth TSG appears among the top k retrieved candidates. As presented in Table 1, the semantic alignment strategy substantially enhances the retrieval performance of vector-based methods, effectively capturing the latent semantic relationships between failure descriptions and historical diagnostic knowledge. A detailed analysis of a representative case is provided in Section §5.3.

Table 1: Comparison of retrieval accuracy

Method	AC@1	AC@3	AC@5
vector retrieval	0.2	0.4	0.75
semantic alignment with vector retrieval	**0.45**	**0.7**	**0.9**

Moreover, SOPs, authored by domain experts or generated through automated tools, present formalized diagnostic workflows (as shown in Figure 3) that help reduce trial-and-error and improve the efficiency of failure resolution [16]. When diagnosing failures that

present with similar symptoms, SOPs employ branching logic to guide engineers through step-by-step procedures that help identify the underlying cause. The domain structured reasoning capability is largely lacking in most LLMs. In addition, LLMs pre-trained on general-purpose corpora often produce recommendations that, while linguistically sound, fail to align with the specific operational context. This limitation stems from their inability to internalize and reason over domain-specific procedural knowledge, such as the sequential logic and decision paths encoded in SOPs, thereby constraining their effectiveness in real-world diagnostic scenarios. For instance, in diagnosing issues related to kernel parameter configurations, an LLM might provide solutions tailored to mainstream OSes, which are often inapplicable to *Alibaba*'s AliOS. Inspired by Chain-of-Thought (CoT) [35], *OScope* integrates SOP guidance to generate structured and actionable diagnostic reports that adhere to operational procedures.

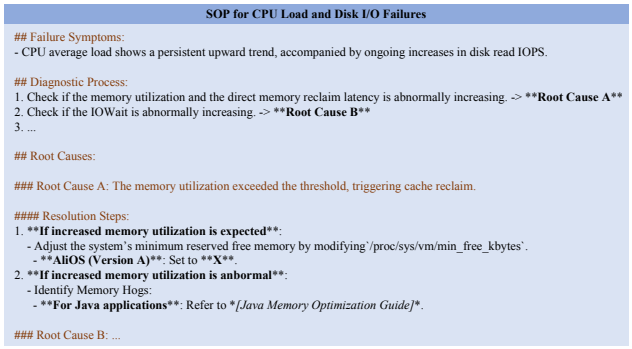


Figure 3: Example SOP for CPU load and disk I/O failures.

3.2 How to Facilitate Effective Interaction with OCEs?

Existing automated diagnosis solutions typically lack interactive capabilities [2, 27], thereby limiting their practical utility. The absence of interaction mechanisms means that OCEs often present with lengthy or overly generic outputs, which may obscure critical information. Furthermore, LLMs may provide conclusions that are difficult for engineers to understand and verify, especially when the diagnostic reasoning is obscure or misaligned with current observations.

To overcome these limitations, *OScope* supports an interactive diagnostic process by generating concise, explainable, and context-sensitive reports that enhance the transparency and conciseness of LLM outputs. More importantly, we design an OCE-LLM-in-the-loop mechanism that supports multi-turn dialogues, preserving historical context and allowing engineers to refine queries or explore diagnostic hypotheses iteratively. The mechanism improves diagnostic accuracy and reduces cognitive burden on OCEs.

4 Approach

4.1 Overview

In this section, we introduce *OScope*, a novel LLM-augmented failure diagnosis framework, *OScope*. As shown in Figure 4, *OScope* consists of three key modules:

(1) Data Analysis with Knowledge Retrieval (§4.2). *OScope* incorporates the *Knowledge Aligner* component to address the deficiency of troubleshooting knowledge in LLM-based reasoning, particularly the issue of inconsistent, ambiguous, and non-standardized symptom descriptions in historical failure documentation. The *Knowledge Aligner* standardizes unstructured symptom narratives into structured, machine-interpretable formats that align with operational semantics. When an alert is raised, *OScope* automatically collects relevant multimodal data (*e.g.*, metrics, logs, stack traces), and performs modality-specific preprocessing and anomaly detection to extract anomalous features. The features are subsequently summarized by the *Knowledge Aligner* into a coherent semantic representation encoded into a query vector. The query vector is used to retrieve semantically similar historical cases from the knowledge base, enabling efficient and informed failure diagnosis grounded in past operational experience.

(2) SOP-Guided Failure Diagnosis (§4.3). *OScope* incorporates a failure diagnosis mechanism explicitly guided by SOPs to address the lack of accurate and actionable outputs in LLM-based diagnostic reasoning, which often stems from limited internalization of procedural knowledge. The *Diagnosis Analyst* component generates an initial diagnostic report by integrating the extracted multimodal anomalous features with retrieved historical cases, and structuring the output according to predefined SOP templates. The diagnostic report is then iteratively refined through the *Report Validator*, which aligns the proposed diagnosis with the relevant SOP documents, thereby ensuring that the validated report is accurate and executable operationally.

(3) OCE-LLM-in-the-Loop (§4.4). *OScope* adopts an interactive HITL paradigm to mitigate the explainability gap stemming from limited domain overlap between OCEs and LLMs. Specifically, the *Diagnosis Analyst* annotates each diagnostic conclusion with confidence scores and provides traceable hyperlinks to supporting evidence, including retrieved failure records and SOP documents, thereby improving transparency and explainability. Furthermore, *OScope* enables effective interaction between OCEs and the *Diagnosis Analyst*, allowing engineers to request elaborations on specific reasoning steps or contextual clarifications. Such an interaction mechanism fosters trust and supports nuanced, scenario-specific diagnostic decisions that align with real-world operational constraints.

It is important to note that the *Knowledge Aligner* is an independently fine-tuned model, whereas the roles of *Diagnosis Analyst* and *Report Validator* are executed by a single model through prompting with retrieval. This approach offers greater resource efficiency compared to deploying three separate models, as it reduces both computational overhead and memory usage, thereby enabling a more streamlined framework without compromising functionality.

4.2 Data Analysis with Knowledge Retrieval

4.2.1 Knowledge Base Construction. Historical failure documents compiled by OCE comprise detailed descriptions of failure symptoms and corresponding troubleshooting procedures, serving as valuable references for subsequent failure diagnosis. Discrepancies in terminology, variations in descriptive styles among engineers, and the time-sensitive nature of online incident response contribute to inconsistent narrations for similar failures. Such inconsistency

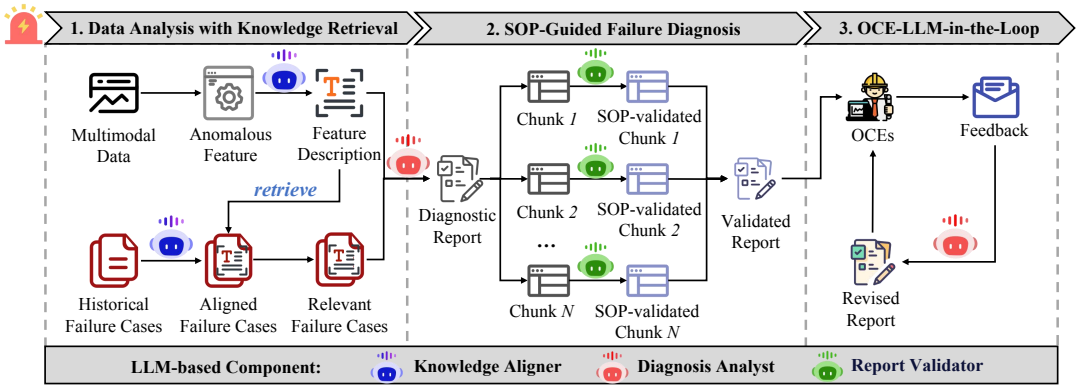


Figure 4: The framework of *OScope*.

presents a substantial obstacle to information retrieval and compromises the accuracy of knowledge reuse in diagnostic workflows.

To address the above limitation, *OScope* employs a systematic methodology to align failure symptom descriptions semantically. The procedure begins by segmenting the failure documents into individual cases, after which all 157 cases from the historical records are selected for standardization. Three senior OCEs participate in this process: two independently rewrite each failure symptom using a predefined set of terminology rules, and a third OCE resolves any discrepancies to finalize the narration.

The constructed dataset, which includes the original symptom descriptions as inputs, the standardized narrations as outputs, and explicit semantic alignment instruction, serves as the foundation for supervised fine-tuning of the *Knowledge Aligner*. To augment the training data, additional samples are synthesized using semantic similarity heuristics, thereby capturing a wider range of expression variants. After fine-tuning, the *Knowledge Aligner* processes the broader corpus of historical failure documents to produce standardized descriptions, which are then embedded using BGE [37] and stored in a vector database, facilitating efficient and precise semantic retrieval during diagnostic tasks.

4.2.2 Data Collection and Preprocess. Upon alert triggering, *OScope* initiates the collection of multimodal observability data to support comprehensive failure diagnosis. Specifically, metrics, logs, and stack traces are collected from *Alibaba*'s unified monitoring platform according to the timestamp and machine IP, spanning the three hours preceding the alert up to the moment of the alert. Notably, each failure case generates approximately 200 KB of multimodal data, which is automatically purged after analysis. As a result, the data collection incurs minimal resource overhead, with memory utilization increasing by 0.25% and CPU utilization by 3%.

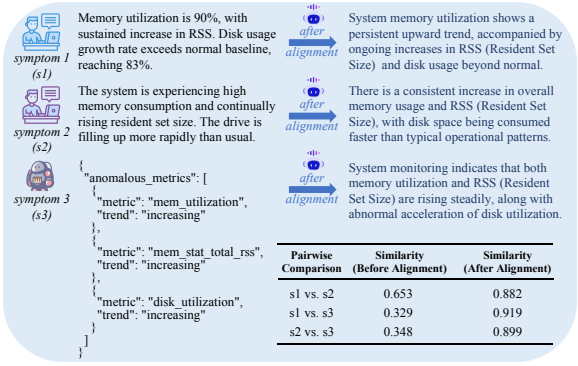
Each modality of data offers distinct and complementary insights into the system's state. Metric data, typically formatted as time series, provides quantitative measurements of system performance, such as resource utilization [18]. The preprocessing of metric data involves Min-max normalization to ensure consistency across different scales and time series alignment to enable effective correlation across multiple sources. Log data, usually semi-structured, records key events and system states, offering contextual information crucial for failure diagnosis [30]. To retain the core semantics of log entries, we employ the Drain log parsing algorithm [8] to extract log templates. The templates encapsulate the essential structural patterns of log messages, abstracting away variable content. Stack traces provide detailed records of function call sequences preceding failures, making them particularly effective for pinpointing execution bottlenecks [5]. The preprocessing of stack trace data includes extracting hierarchical relationships among function calls, enabling the provision of execution paths for analytical tasks.

4.2.3 Feature Extraction and Knowledge Retrieval. To facilitate effective failure diagnosis, we perform anomaly detection across multimodal data to extract discriminative features that signify anomalous OS behavior. Different data modalities are processed using tailored detection strategies, as detailed below:

- Metric Data. We adopt the 3-sigma rule as the anomaly detection criterion [1]. Due to its simplicity, computational efficiency, and robust performance, this method is widely employed in time series anomaly detection. Moreover, we extract temporal trends (*e.g.*, increasing or decreasing patterns) of the anomalous metrics. When interpreted alongside the physical meanings of the corresponding metrics, these trend features offer contextual information that enhances diagnostic accuracy [45].
- Log Data. We first embed log templates into semantic vectors and apply clustering based on cosine similarity to group semantically similar templates. For each cluster, we extract two key features: (1) the centroid embedding, representing the dominant semantic pattern, and (2) the aggregated count of templates within the cluster, reflecting its frequency. They are fed into the Isolation Forest algorithm [23] to identify anomalous clusters. The combination of semantic and statistical attributes enhances the model's effect [52].
- Stack Trace Data. We analyze stack traces by computing the frequency distribution of function calls, focusing on leaf functions located at the terminal positions of call chains. Elevated invocation frequencies of specific leaf functions often indicate potential performance bottlenecks [6]. Furthermore, we quantify the occurrence of specific keywords (*e.g.*, "do_anonymous_page") within the traces, which serve as potential indicators of system failures.

To facilitate semantic alignment between the extracted anomalous features and historical failure records, we incorporate the *Knowledge Aligner* component, synthesizing concise failure symptom summaries from the multimodal data. The summaries are encoded using BGE and employed as query embeddings to retrieve the *topK* most semantically similar cases from the historical TSGs.

Figure 5 illustrates a representative failure case that includes two symptom descriptions provided by OCEs ($s1$ and $s2$) and the corresponding anomaly detection output from *OScope* ($s3$). These descriptions, which vary in terminology, expression style, and structural form, are semantically aligned through the *Knowledge Aligner*. As shown, the alignment process not only standardizes heterogeneous descriptions into consistent representations but also significantly increases pairwise similarity scores, thus improving the reliability of knowledge retrieval and the effectiveness of case matching.



Pairwise Comparison	Similarity (Before Alignment)	Similarity (After Alignment)
s1 vs. s2	0.653	0.882
s1 vs. s3	0.329	0.919
s2 vs. s3	0.348	0.899

Figure 5: Effect of semantic alignment on symptom representation similarity.

4.3 SOP-Guided Root Cause Analysis

While LLMs exhibit robust general-purpose reasoning capabilities, their direct application to specialized domains such as failure diagnosis often yields outputs that are either inaccurate or non-actionable. This limitation arises from the domain's reliance on specialized technical knowledge and the need for precise procedural execution [27, 39]. To address this challenge, we draw inspiration from Retrieval Augmented Thoughts [33] and propose a mechanism grounded in SOPs. The central concept of our approach is a two-stage process designed to enhance diagnostic accuracy. The first stage involves generating a draft diagnostic report utilizing SOP templates. Subsequently, it iteratively validates each chunk of the report against the SOP knowledge base.

4.3.1 Initial Diagnosis CoT Generation. Given the anomaly features extracted from multimodal system data, we initiate the process with the *Diagnosis Analyst* component. Guided by CoT prompts derived from SOP templates, this module produces a sequential diagnostic reasoning chain, denoted as $C = \{C_1, C_2, ..., C_N\}$, which forms the initial diagnostic report. Due to the limitations of general-purpose LLMs in capturing domain-specific knowledge, the preliminary output may contain imprecise assumptions or flawed reasoning steps, necessitating further validation.

4.3.2 Iterative SOP-Guided Validation and Correction. To enhance the accuracy of the initial diagnostic reasoning, we introduce the *Report Validator*, which incrementally verifies and revises each step in C. Instead of verifying the entire report, the *Report Validator* implements a sequential strategy, which has been demonstrated to be more effective in rectifying errors [33]. For each step C_i, the *Report Validator* formulates a retrieval query composed of C_i and the previously validated subsequence $C_{1:i-1}$. The query is subsequently used to retrieve relevant SOPs from a knowledge base.

The SOP knowledge base is a compiled repository of documents authored by domain experts or extracted from technical manuals, ensuring high accuracy and operational reliability. For the retrieval process, the titles of the SOP documents are encoded into vector representations and indexed within a vector database to facilitate efficient semantic retrieval. The content of these documents serves as the grounding reference against which each step is validated.

The iterative process of embedding and sequentially correcting each reasoning step C_i produces a validated diagnostic report that is both accurate and executable. The proposed methodology ensures that every step in the diagnostic reasoning is explicitly aligned with established SOP knowledge, thereby substantially improving the reliability and practical applicability of the LLM-generated output. Figure 6 demonstrates that SOP-guided correction can accurately identify the underlying cause while providing resolution steps that are adapted to the current OS version. The specific prompts employed are detailed in Figures 7 and 8.

Diagnostic Report

Diagnostic Process:
1. High CPU load average indicates that the system is overloaded with processes, either actively executing or waiting for access to resources.
2. High disk read IOPS indicates that one or more applications are continuously performing intensive read operations on a large volume of files or data from the disk.

Root Causes:
The system is experiencing high CPU load due to processes running or waiting for resources, combined with high disk read activity from I/O-bound applications.

Resolution Steps:
1. Use the iotop or iostat command to monitor disk I/O activity in real time and identify which specific process or application is generating the unusually high read I/O.

Validated Report

Diagnostic Process:
1. High CPU load average indicates that the system is overloaded with processes, either actively executing or waiting for access to resources. Elevated memory utilization and the resulting increase in direct memory reclaim activity can further exacerbate CPU load, as threads attempting to allocate memory enter an uninterruptible sleep state while awaiting memory availability.
2. High disk read IOPS indicates that one or more applications are continuously performing intensive read operations on a large volume of files or data from the disk. This I/O activity may be further intensified under memory pressure, as slower memory reclaim operations prolong the time threads spend waiting on disk access.

Root Causes:
The memory utilization has likely exceeded a critical threshold, triggering aggressive direct memory reclaim. This causes threads attempting memory allocation to block in uninterruptible sleep, which increases CPU load and amplifies disk I/O delays.

Resolution Steps:
1. Adjust the system's minimum reserved free memory (/proc/sys/vm/min_free_kbytes) to X according to the current OS version.

Figure 6: Comparison between the original diagnostic report and the validated report following SOP-guided validation and correction. Red text indicates corrections of errors, while orange text highlights supplementary information.

Role: As a Senior Research Assistant specializing in rapid information verification, your primary responsibility is to formulate precise and concise search queries to validate the claims.
Task: You will be provided with content containing a statement requiring verification. Your task is to identify the primary assertion, typically located toward the conclusion of the content, and construct an optimal search query to facilitate fact-checking.
Requirements:
1. The query must not exceed 20 words to ensure efficiency and focus.
2. The query must be presented directly, without accompanying explanations.

Figure 7: Prompt for verification query generation.

Role: As a Technical Knowledge Editor, your primary responsibility is to ensure that all provided responses strictly adhere to the guidelines outlined in the SOPs.
Task: You will be provided with two key inputs: a `[Source Text]` extracted from the SOPs and a `[Draft Answer]` intended to address a specific query. Your task is to refine the [Draft Answer] to ensure complete alignment with the `[Source Text]`.
Requirements:
1. If the answer is incorrect: Rewrite the erroneous parts to match the `[Source Text]`.
2. If the answer is incomplete: Integrate the missing key steps or essential details from the `[Source Text]` to make the answer comprehensive and safe to follow.
3. If the answer is consistent with the SOPs: Return it without modification.

Figure 8: Prompt for answer validation.

4.4 OCE-LLM-in-the-Loop

In industrial settings, OCEs typically possess deep but narrowly focused expertise specific to the services they manage. Consequently, the diagnostic reports they receive frequently contain complex

reasoning processes and technical jargon, which can hinder the efficient extraction of critical information, particularly when there is minimal overlap in knowledge. The complexity underscores the fundamental requirement for diagnostic systems to provide transparent, explainable, and actionable explanations. Enhanced explainability strengthens the credibility of diagnostic outputs and facilitates informed decision-making by OCEs. To address these challenges, the *Diagnosis Analyst* is engineered to assign a confidence score to each inference. Moreover, it provides hyperlinks to the specific source documents referenced during the reasoning process, thereby ensuring traceability of the diagnostic conclusions.

To further augment the reliability and usefulness of the diagnostic reports, the framework facilitates direct interaction between the OCEs and the *Diagnosis Analyst*. The interactive mechanism allows for real-time clarification, refinement, and contextual adaptation of the diagnostic findings. To support the dynamic interaction, the *Diagnosis Analyst* maintains a comprehensive dialogue history for each diagnostic session. Furthermore, anomalous features identified in the current failure are integrated with a knowledge base of historical TSGs and SOP documents. The unified knowledge base facilitates a more accurate and context-aware diagnostic process.

5 Evaluation

In this section, we aim to answer the following research questions (RQs):

- RQ1: How effective is *OScope* in failure diagnosis?
- RQ2: Does each component of *OScope* contribute significantly to *OScope*'s performance?
- RQ3: How beneficial are the reports of *OScope* for OCEs?

5.1 Experimental Setup

5.1.1 Dataset. To comprehensively evaluate the performance of *OScope*, we collect a total of 101 failure cases from the production environment of *Alibaba*. These cases relate to Linux-based systems, including standard Linux distributions and their customized variant, AliOS, which is widely deployed in our infrastructure. It is important to note that each failure represents typical issues within the OS, thereby providing valuable insights for diagnosis. The dataset has been manually labeled with root causes by experienced OCEs, which serves as the ground truth for evaluation.

5.1.2 Evaluation Metrics. We employ multiple metrics to comprehensively assess the effectiveness of *OScope*.

To assess the accuracy of root cause localization, we employ the metrics *AC@k* and *Avg@k*, which are widely adopted in recent literature [19, 31, 49]. Specifically, *AC@k* measures the probability that the actual root cause is included among the top k results returned by each method for all cases. *Avg@k* summarizes the overall performance by calculating the average *AC@k* across all cases.

To evaluate the usability of the diagnostic reports, we adopt the human-centered evaluation methodology proposed in LogPrompt [24]. Traditional metrics based on lexical or semantic similarity fail to adequately capture the practical utility of outputs in the context of OCEs. Therefore, we assess the clarity and actionability of each report using expert-based evaluation. Specifically, three domain experts independently rate each report on two 5-point Likert scales, reflecting its readability and usefulness for guiding operational decisions [1]. The detailed criteria are listed in Table 2.

- Readability: assesses the clarity, structure, and technical fluency of the report.
- Usefulness: reflects how helpful the diagnosis facilitates failure resolution by OCEs.

5.1.3 Baselines. We intentionally excluded comparisons with AIOps methodologies [17, 19, 44, 53] that depend on failure-free data. Due to the scenario-specific nature of current LLM-based diagnostic methods, which are not tailored for OS failure diagnosis, these approaches are neither open-source nor easily transferable. Therefore, we compare *OScope* with two general-purpose open-source frameworks representative of few-shot learning: (1) ReAct [40] integrates reasoning and action by allowing the model to generate thoughts and interact with the environment to arrive at a solution. (2) CoT [35] encourages the model to decompose diagnostic tasks into intermediate reasoning steps to improve accuracy. Similar to COCA [21], both models will retrieve historical TSGs as knowledge sources to augment the effectiveness of failure diagnosis.

5.1.4 Implementation. We conduct all the experiments with four NVIDIA H20 GPUs, PyTorch 2.7.1, and CUDA 12.1. We utilize LoRA [10] with a rank of 8 and an alpha of 16 for fine-tuning, and the dropout rate is set to 0.05.

5.2 RQ1: Overall Performance

Table 3 presents the diagnostic performance of *OScope* and baseline methods on real-world failures collected from a production environment at *Alibaba*. *OScope* consistently outperforms all baseline methods, achieving a notable *AC@5* of 0.901, representing a 20% absolute improvement over ReAct, the best-performing baseline. Regarding overall performance, *OScope* also leads with an *Avg@5* score of 0.683, corresponding to a 13% relative improvement compared to the second-best method. The results of different seed LLMs demonstrate that *OScope* maintains robustness and generalizability, validating its applicability across different architectures.

Although CoT [35] benefits from the inference capabilities of LLMs, it only performs well on relatively simple or common failure cases. Even experienced OCEs struggle to identify accurate diagnostic steps in more complex or less frequent scenarios, resulting in limited CoT effectiveness. ReAct [40], while capable of autonomously generating inference traces, often suffers from prompt bias and hallucinations due to the lack of integration of domain knowledge. In contrast, *OScope* is guided by SOPs and enriched with operational knowledge throughout the reasoning process, which is essential for achieving high diagnostic accuracy in complex scenarios.

5.3 RQ2: Ablation Study

To validate the effectiveness of the core components of *OScope*, we conduct an ablation study by removing key modules, ensuring that their removal does not impair the functionality of *OScope*. The results, summarized in Table 4, highlight the critical contributions of each component. When the *Knowledge Aligner* is removed, *OScope* relies solely on semantic vector retrieval for historical case

[1]The use of three experts is commonly adopted in similar research [7, 39] to strike a balance between the diversity of opinions and the reliability of the evaluation process.

Table 2: Criteria for readability and usefulness evaluation

Scores	Readability	Usefulness
1	The report is unclear, verbose and hard to read; the terminology is inconsistent or unexplained.	The report is not actionable and untrustworthy; it has no reference value for future cases.
2	The report contains noticeable ambiguity or redundancy; the terminology is partially inconsistent or unexplained.	The report is hard to apply and lacks supporting evidence; it offers limited reference value.
3	The report is generally readable but somewhat disorganized; the terminology is mostly consistent but occasionally unclear.	The report is somewhat actionable and partially supported; it may be useful for similar future cases with interpretation.
4	The report is clear and well-structured; the terminology is consistent and generally well explained.	The report is generally actionable and well-supported; it provides good reference value for future diagnosis.
5	The report is concise, easy to read, and well-organized; the terminology is precise, consistent, and clearly explained.	The report is highly actionable and reliable; it serves as a clear and reusable reference for similar cases.

Table 3: Effectiveness of root cause localization

Seed LLM	Method	AC@3	AC@5	Avg@5
Qwen3-8B	*OScope*	**0.713**	**0.901**	**0.683**
	ReAct [40]	0.564	0.703	0.549
	CoT [35]	0.515	0.594	0.463
InternLM3-8B	*OScope*	0.683	0.861	0.618
	ReAct [40]	0.426	0.634	0.434
	CoT [35]	0.228	0.544	0.269
Llama3.1-8B	*OScope*	0.663	0.861	0.626
	ReAct [40]	0.287	0.574	0.349
	CoT [35]	0.297	0.505	0.255

matching, leading to a substantial decline in retrieval precision. This observation highlights the essential role of the *Knowledge Aligner* in improving the relevance of retrieval results. Similarly, removing the *Report Validator* eliminates the guided reasoning process, causing the LLM to fail to correctly diagnose rare or complex failures due to the lack of structured diagnosis steps.

Table 4: The contributions of the main components in *OScope*

Method	AC@3	AC@5	Avg@5
OScope	**0.713**	**0.901**	**0.683**
OScope w/o *Knowledge Aligner*	0.406	0.732	0.465
OScope w/o *Report Validator*	0.525	0.782	0.525

[a]The LLM backbone we use is Qwen3-8B.

To illustrate the effectiveness of *OScope*, we present a case involving a memory leak failure. As shown in Figure 9, the top half of the heatmap displays the similarity scores between the anomalous feature and historical cases retrieved by *OScope* without the *Knowledge Aligner*. In contrast, the bottom half shows the results enabled by the *Knowledge Aligner*. The results show that relying solely on semantic vector-based similarity tends to retrieve irrelevant cases (*e.g.*, CPU-related or network-related failures), which may mislead the diagnosis process. In contrast, *OScope*, augmented with the *Knowledge Aligner*, significantly improves retrieval accuracy by identifying memory-related cases, thus facilitating more effective root cause localization.

5.4 RQ3: Usability for OCEs

Leveraging the LLM's capability to generate insightful responses, *OScope* delivers clear and structured diagnostic reports. Guided by prompt engineering and SOPs, *OScope* provides detailed feedback on anomalous symptoms, root causes, troubleshooting processes, and resolution strategies. To assess the usability of these reports in real-world production environments, we conduct a user study involving three experts from *Alibaba*. The experts evaluate 30 randomly selected failure cases from the dataset, assessing readability and usefulness according to the criteria defined in Table 2.

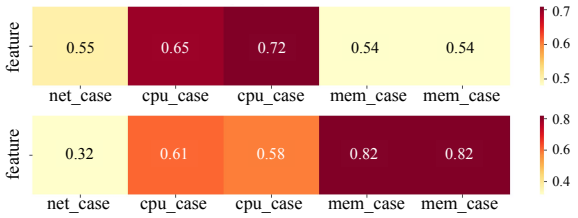


Figure 9: Case similarity heatmaps for a memory leak failure. The top heatmap shows the retrieval results produced by *OScope* w/o *Knowledge Aligner*, while the bottom heatmap shows the results produced by *OScope*.

As listed in Table 5, the diagnostic reports achieved a mean readability score of 4.14, with a HIP (percentage rated four or higher [24]) of 87.78%, indicating that reports are generally clear, well-structured, and easy to understand. For usefulness, the mean score reached 3.84 with a HIP of 75.56%, demonstrating that most reports are effective in diagnosing failures. These results provide strong evidence for the applicability of *OScope* in supporting real-world failure diagnosis.

Table 5: Practical utility of *OScope* rated by experts

Raters	Readability		Usefulness	
	Mean	HIP	Mean	HIP
R1	4.13	90.00%	3.83	73.33%
R2	4.03	83.33%	3.77	73.33%
R3	4.27	90.00%	3.93	80.00%
Avg.	4.14	87.78%	3.84	75.56%

[a]The LLM backbone we use is Qwen3-8B.

6 Discussion

6.1 Deployment

We have deployed *OScope* within the *Foresight* operations and maintenance (O&M) system at *Alibaba* for trial implementation across

three teams. During a stable operational period spanning over three months, *OScope* successfully diagnosed 67 OS failures that engineers had initially reported due to their considerable business impact [2]. Based on internal evaluations, the average time required by *OScope* to complete a diagnosis was approximately 1.5 minutes, compared to an average of 112 minutes taken by OCEs. The results highlight an improvement in diagnostic efficiency and demonstrate *OScope*'s potential to reduce the time and manual effort involved in diagnosis.

Through empirical evaluations conducted with three teams, we demonstrate that fine-tuning the *Knowledge Aligner* using a lightweight LLM (7B) produces highly effective results. Leveraging our experimental configuration (see §5.1), the fine-tuning process for 1,000 instructions is completed in under 10 minutes. Once fine-tuned, the *Knowledge Aligner* serves as a reusable component. Additionally, the *Diagnosis Analyst* and *Report Validator* modules are accessed via API, eliminating the need for extensive local computational infrastructure without compromising system performance. To promote wider adoption within *Alibaba*, we have made *OScope* internally available, allowing teams to deploy *OScope* easily.

6.2 Case Study

To comprehensively demonstrate the workflow of *OScope*, we present a detailed case study involving a memory leak failure that occurred during deployment. All sensitive information has been anonymized to ensure security and compliance with privacy policies.

As shown in Figure 10, the diagnostic process began when an engineer observed an abnormal increase in disk utilization on machine *A*. (1) The engineer immediately reported the *A*'s IP address and the timestamp to *OScope*, which then collected multimodal data from the monitoring system for analysis. (2) During the analysis, *OScope* detected anomalous trends in memory-related metrics and an unusually high frequency of memory allocation function calls in the stack trace. Subsequently, these findings were processed by the *Knowledge Aligner*, which characterized the anomalous features to enable semantic alignment with historical failure records, facilitating the retrieval of relevant TSGs based on the aligned features. (3) Leveraging the identified anomalous features and TSGs from the retrieved cases, the *Diagnosis Analyst* generated an initial diagnostic report, following SOP templates. (4) The report was subsequently reviewed by the *Report Validator*, which systematically verified the accuracy and completeness of each diagnostic step. (5) Upon reviewing the report, the engineer notes that the memory utilization remained below the configured threshold. In response, the *Diagnosis Analyst* re-examined the anomalous features and highlighted an abnormal increase in memory recovery time. Guided by the critical insight, the engineer investigated and identified the memory leak.

6.3 Generalizability and Scalability

6.3.1 Generalizability. This study is based on data collected from the production environment of *Alibaba*'s e-commerce platform, thereby ensuring the representativeness and generalizability of our findings and methodologies. The datasets encompass a broad spectrum of business scenarios within *Alibaba* are derived from Linux and its various distributions, highlighting the broad applicability and adaptability of *OScope* across heterogeneous OS environments.

[2]This scale of deployment and diagnosis compares favorably with existing industry benchmarks (*e.g.*, RCACopilot [2], covering 653 incidents over one year at Microsoft).

6.3.2 Robustness. *OScope* exhibits notable robustness under varying data availability conditions. Although *OScope* is designed to perform comprehensive failure diagnosis by jointly analyzing metrics, logs, and stack traces, it retains core diagnostic capabilities even without specific data modalities. The resilience enables *OScope* to operate effectively in data-constrained environments while remaining compatible with real-world operational settings.

6.3.3 Flexibility. The modular design of *OScope* provides significant flexibility, enabling users to tailor the framework to specific application requirements without compromising overall diagnostic effectiveness. For instance, the retrieval strategy can be customized to align with the characteristics of domain-specific data. Notably, the *Knowledge Aligner* requires only a lightweight fine-tuning process using a small-scale model, which can be reused across diverse diagnostic scenarios. Moreover, the components within *OScope* are designed to be compatible with more advanced LLMs, facilitating seamless integration and supporting future scalability.

6.4 Threats to Validity

6.4.1 Internal Threats. To mitigate the impact of potential instability of LLMs and to account for variability across evaluation runs, we conducted three independent experimental rounds for *OScope* and the baselines. Moreover, all baseline implementations strictly adhered to the corresponding code repositories and papers, carefully verifying their correctness by two independent authors. Furthermore, *OScope* adopts a data-driven approach. While *OScope* can operate with any individual modality, accurate diagnosis becomes challenging if all data sources are unavailable or if specific failures do not produce observable anomalies in the collected data. This limitation is widely acknowledged in prior works [17, 53]. In addition, we leverage a CoT reasoning strategy, employing SOPs as inference templates, supplemented with few-shot examples and confidence scoring to iteratively test and refine prompts [27]. While our manually designed prompts have demonstrated effectiveness, a direction for future work is the exploration of prompt optimization techniques, such as incorporating self-reflection mechanisms [29].

6.4.2 External Threats. We conducted a comprehensive evaluation of *OScope* and baselines using a real-world failure dataset provided by *Alibaba*. The dataset was prepared with domain experts from the Infrastructure and Stability Engineering team, who have substantial experience and practices in failure labeling. To ensure the reliability of the ground-truth labels, each failure case was independently reviewed by at least three experts. The dataset comprises failure cases collected across multiple businesses, offering diverse failure scenarios. In future work, we plan to expand this dataset further to enable a more thorough assessment of generalization performance.

7 Related Work

7.1 Traditional Methods

Traditional failure diagnosis techniques can be broadly categorized based on the modality of data they utilize, encompassing metrics, logs, traces, and multimodal data.

- Metric-based Methods [11, 19, 25, 36]: Some methods [25, 36] typically involve extracting statistical or temporal features from

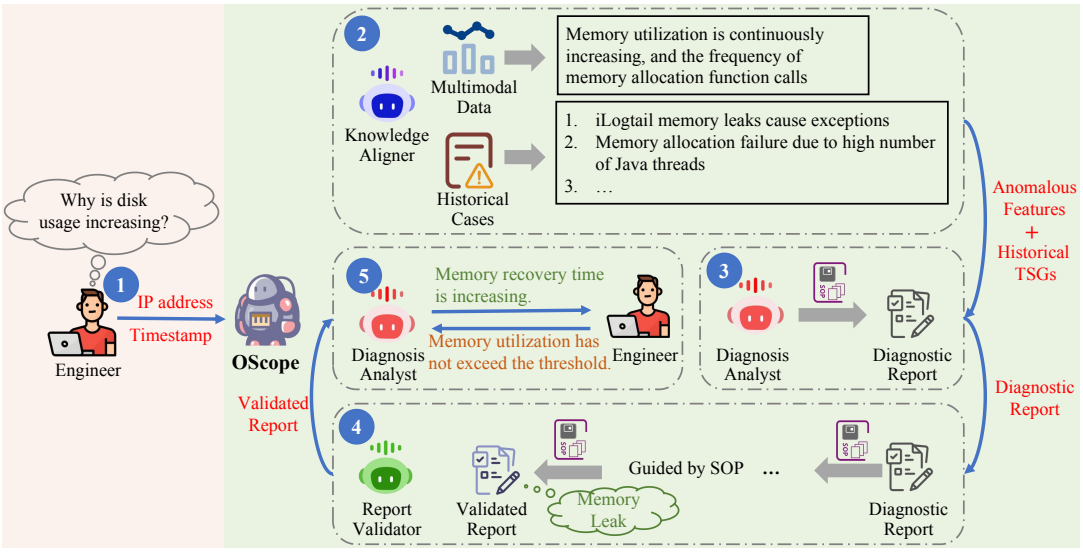


Figure 10: A case: memory leak failure diagnosed by *OScope*.

time series data to train classification models for failure type identification and root cause localization. In addition, graph-based techniques [11, 19] leverage telemetry data, such as Remote Procedure Call (RPC) data, to construct causality graphs, employing algorithms like random walk to identify root causes.

- Log-based Methods [22, 30, 46, 47, 51]: Log analysis is widely used in diagnosing failures by detecting anomalous or infrequent log patterns. For instance, LogCluster [22] applies clustering algorithms to detect anomalous events. Cloud19 [46] investigates cloud logs to identify anomalous patterns during failed executions of cloud OS.
- Trace-based Methods [20, 42, 43, 53, 55]: These methods focus on analyzing traces to capture anomalous patterns. For example, MEPFL [53] extracts features reflecting service interactions to predict latent errors, localize faulty microservices, and determine fault types in production microservice applications. Other approaches [20, 55] utilize stack trace mining to diagnose software crashes and identify failure code paths.
- Multimodal Methods [4, 17, 31, 41, 44, 49]: To improve diagnostic robustness, multimodal methods integrate heterogeneous observability data into a unified diagnostic framework. Typically, these methods [41, 44, 49] involve transforming all data modalities into event sequences to facilitate anomaly detection and root cause analysis.

Nevertheless, none of the approaches offer explainable insights into the reasoning process behind results within the OS context, thereby limiting their practical utility for OCEs in failure diagnosis.

7.2 LLM-based Methods

With the advancement of LLMs, a growing body of research has explored their application in failure diagnosis, leveraging their strong capabilities in natural language understanding, reasoning, and multi-source information synthesis.

ReAct-based agents have been employed autonomously to orchestrate root cause analysis [28, 34]. mABC [50] adopts agents organized in a blockchain-inspired polling structure to diagnose failures through workflows. RCACopilot [2] aggregates runtime diagnostic information and employs LLMs to infer the root cause category of incidents. Flow-of-Action [27], a system based on SOPs, guides the diagnosis process by summarizing typical diagnostic steps followed by engineers to assist in root cause identification. L4 [14] is a diagnostic framework trained on extensive log data, designed to identify failure-indicating log content and streamline the process. ScalaLog [48] is a log-based diagnosis method for IIoT systems, leveraging LLM-based summarization, sample augmentation, and CoT prompting to improve accuracy without requiring model retraining or log parsing. AutoFL [15] applies LLMs to analyze stack traces, enabling failure localization across software repositories. Additionally, LasRCA [7] leverages the collaboration of the LLM and the classifier to localize the root cause. However, their limited incorporation of domain operational knowledge and absence of interactive mechanisms significantly constrain their practical applicability.

8 Conclusion

This study investigates the problem of failure diagnosis in OSes and presents *OScope*, an LLM-based framework designed to enhance the accuracy and efficiency of diagnosis. *OScope* identifies relevant TSGs by extracting anomalous features from observable data and performing semantic alignment with historical failure symptoms. Furthermore, *OScope* leverages SOP templates to guide the diagnostic process, retrieving relevant SOPs in chunks for verification and correction, thereby improving the reliability of diagnostic results. Finally, *OScope* generates diagnostic reports for OCEs and supports HITL interactions to refine and validate the results. To assess its effectiveness, we conducted extensive evaluations on 101 OS failures collected from *Alibaba*'s production environment. *OScope* has been successfully deployed in a specific O&M system at *Alibaba*, significantly improving the efficiency of failure diagnosis and resolution by OCEs.

References

[1] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. 2022. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Comput. Surv.* 54, 3, Article 56 (April 2022), 33 pages. doi:10.1145/3444690

[2] 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* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 674–688. doi:10.1145/3627703.3629553

[3] Min Du and Feifei Li. 2016. Spell: Streaming Parsing of System Event Logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 859–864. doi:10.1109/ICDM.2016.0103

[4] Chiming Duan, Yong Yang, Tong Jia, Guiyang Liu, Jinbu Liu, Huxing Zhang, Qi Zhou, Ying Li, and Gang Huang. 2025. FAMOS: Fault diagnosis for Microservice Systems through Effective Multi-modal Data Fusion. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 610–610.

[5] Mohamad Gebai and Michel R Dagenais. 2018. Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–33.

[6] Brendan Gregg. 2016. The flame graph. *Commun. ACM* 59, 6 (2016), 48–57.

[7] Yongqi Han, Qingfeng Du, Ying Huang, Jiaqi Wu, Fulong Tian, and Cheng He. 2024. The Potential of One-Shot Failure Root Cause Analysis: Collaboration of the Large Language Model and Small Classifier. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 931–943. doi:10.1145/3691620.3695475

[8] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. 33–40. doi:10.1109/ICWS.2017.13

[9] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.* 54, 6, Article 130 (July 2021), 37 pages. doi:10.1145/3460345

[10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.

[11] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root Cause Analysis of Failures in Microservices through Causal Discovery. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 31158–31170. https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Paper-Conference.pdf

[12] Shian Jia, Xinbo Wang, Mingli Song, and Gang Chen. 2024. Agent Centric Operating System – a Comprehensive Review and Outlook for Operating System. arXiv:2411.17710 [cs.DC] https://arxiv.org/abs/2411.17710

[13] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 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* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1410–1420. doi:10.1145/3368089.3417054

[14] Zhihan Jiang, Junjie Huang, Zhuangbin Chen, Yichen Li, Guangba Yu, Cong Feng, Yongqiang Yang, Zengyin Yang, and Michael R Lyu. 2025. L4: Diagnosing Large-scale LLM Training Failures via Automated Log Analysis. In *Companion Proceedings of the 33nd ACM International Conference on the Foundations of Software Engineering*.

[15] Sungmin Kang, Gabin An, and Shin Yoo. 2024. A quantitative and qualitative evaluation of LLM-based explainable fault localization. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1424–1446.

[16] Jinxi Kuang, Jinyang Liu, Junjie Huang, Renyi Zhong, Jiazhen Gu, Lan Yu, Rui Tan, Zengyin Yang, and Michael R. Lyu. 2024. Knowledge-aware Alert Aggregation in Large-scale Cloud Systems: a Hybrid Approach. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (Lisbon, Portugal) *(ICSE-SEIP '24)*. Association for Computing Machinery, New York, NY, USA, 369–380. doi:10.1145/3639477.3639745

[17] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1750–1762. doi:10.1109/ICSE48619.2023.00150

[18] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R. Lyu. 2023. Heterogeneous Anomaly Detection for Software Systems via Semi-Supervised Cross-Modal Attention. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1724–1736. doi:10.1109/ICSE48619.2023.00148

[19] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 3230–3240. doi:10.1145/3534678.3539041

[20] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*. 169–180.

[21] Yichen Li, Yulun Wu, Jinyang Liu, Zhihan Jiang, Zhuangbin Chen, Guangba Yu, and Michael Lyu. 2025. COCA: Generative Root Cause Analysis for Distributed Systems with Code Knowledge . In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE, 770–770. doi:10.1109/ICSE55347.2025.00234

[22] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 102–111. doi:10.1145/2889160.2889232

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. 413–422. doi:10.1109/ICDM.2008.17

[24] Yilun Liu, Shimin Tao, Weibin Meng, Feiyu Yao, Xiaofeng Zhao, and Hao Yang. 2024. LogPrompt: Prompt Engineering Towards Zero-Shot and Interpretable Log Analysis. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 364–365. doi:10.1145/3639478.3643108

[25] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, Feifei Li, Changcheng Chen, and Dan Pei. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proc. VLDB Endow.* 13, 8 (April 2020), 1176–1189. doi:10.14778/3389133.3389136

[26] Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. 2024. LLMParser: An Exploratory Study on Using Large Language Models for Log Parsing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 99, 13 pages. doi:10.1145/3597503.3639150

[27] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, Gaogang Xie, and Dan Pei. 2025. Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis. In *In Companion Proceedings ofthe ACMWeb Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 422–431. doi:10.1145/3701716.3715225

[28] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring LLM-Based Agents for Root Cause Analysis. 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, 208–219. doi:10.1145/3663529.3663841

[29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., 8634–8652.

[30] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, Dan Pei, Xiao Yang, and Li Yu. 2023. LogKG: Log Failure Diagnosis Through Knowledge Graph. *IEEE Transactions on Services Computing* 16, 5 (2023), 3493–3507. doi:10.1109/TSC.2023.3293890

[31] Yongqian Sun, Zihan Lin, Binpeng Shi, Shenglin Zhang, Shiyu Ma, Pengxiang Jin, Zhenyu Zhong, Lemeng Pan, Yicheng Guo, and Dan Pei. 2025. Interpretable Failure Localization for Microservice Systems Based on Graph Autoencoder. *ACM Trans. Softw. Eng. Methodol.* 34, 2, Article 52 (Jan. 2025), 28 pages. doi:10.1145/3695999

[32] Roman Vasiliev, Dmitrij Koznov, George Chernishev, Aleksandr Khvorov, Dmitry Luciv, and Nikita Povarov. 2020. TraceSim: a method for calculating stack trace similarity. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation* (Virtual, USA) *(MaLTeSQuE 2020)*. Association for Computing Machinery, New York, NY, USA, 25–30. doi:10.1145/3416505.3423561

[33] Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. 2024. RAT: Retrieval Augmented Thoughts Elicit Context-Aware Reasoning and Verification in Long-Horizon Generation. In *NeurIPS 2024 Workshop on Open-World Agents*. https://openreview.net/forum?id=5QtKMjNkjL

[34] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. RCAgent: Cloud Root Cause Analysis by Autonomous Agents with Tool-Augmented Large Language

Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management* (Boise, ID, USA) *(CIKM '24)*. Association for Computing Machinery, New York, NY, USA, 4966–4974. doi:10.1145/3627673.3680016

[35] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[36] Canhua Wu, Nengwen Zhao, Lixin Wang, Xiaoqin Yang, Shining Li, Ming Zhang, Xing Jin, Xidao Wen, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. Identifying Root-Cause Metrics for Incident Diagnosis in Online Service Systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 91–102. doi:10.1109/ISSRE52982.2021.00022

[37] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packed Resources For General Chinese Embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 641–649. doi:10.1145/3626772.3657878

[38] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 199, 12 pages. doi:10.1145/3597503.3639155

[39] Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. 2025. OpenRCA: Can Large Language Models Locate the Root Cause of Software Failures?. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=M4qNIzQYpd

[40] 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)*.

[41] Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. 2024. 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* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 50–61. doi:10.1145/3663529.3663827

[42] Zhenhe Yao, Haowei Ye, Changhua Pei, Guang Cheng, Guangpei Wang, Zhiwei Liu, Hongwei Chen, Hang Cui, Zeyan Li, Jianhui Li, Gaogang Xie, and Dan Pei. 2024. SparseRCA: Unsupervised Root Cause Analysis in Sparse Microservice Testing Traces. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. 391–402. doi:10.1109/ISSRE62328.2024.00045

[43] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3087–3098. doi:10.1145/3442381.3449905

[44] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 553–565. doi:10.1145/3611643.3616249

[45] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. 2024. MonitorAssistant: Simplifying Cloud Service Monitoring via Large Language Models. 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, 38–49. doi:10.1145/3663529.3663826

[46] Yue Yuan, Wenchang Shi, Bin Liang, and Bo Qin. 2019. An Approach to Cloud Execution Failure Diagnosis Based on Exception Logs in OpenStack. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 124–131. doi:10.1109/CLOUD.2019.00031

[47] Lingzhe Zhang, Tong Jia, Mengxi Jia, Hongyi Liu, Yong Yang, Zhonghai Wu, and Ying Li. 2025. Towards Close-to-Zero Runtime Collection Overhead: Raft-Based Anomaly Diagnosis on System Faults for Distributed Storage System. *IEEE Transactions on Services Computing* 18, 2 (2025), 1054–1067. doi:10.1109/TSC.2024.3521675

[48] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Hongyi Liu, and Ying Li. 2025. ScalaLog: Scalable Log-Based Failure Diagnosis Using LLM. In *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1–5. doi:10.1109/ICASSP49660.2025.10888670

[49] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, et al. 2023. Robust failure diagnosis of microservice system through multimodal data. *IEEE Transactions on Services Computing* 16, 6 (2023), 3851–3864.

[50] Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, Chaoran Yan, Zhoujun Li, Tongliang Li, Xu Shi, Liangfan Zheng, et al. 2024. mABC: multi-Agent Blockchain-Inspired Collaboration for root cause analysis in micro-services architecture. *arXiv preprint arXiv:2404.12135* (2024).

[51] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, 1253–1263.

[52] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1404–1415. doi:10.1145/3468264.3473933

[53] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 683–694. doi:10.1145/3338906.3338961

[54] Wenhao Zhu, Yunzhe Lv, Qingxiu Dong, Fei Yuan, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2023. Extrapolating large language models to non-english by aligning languages. *arXiv preprint arXiv:2308.04948* (2023).

[55] Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D Ernst, and Lu Zhang. 2019. An empirical study of fault localization families and their combinations. *IEEE Transactions on Software Engineering* 47, 2 (2019), 332–347.