LogPilot: Intent-aware and Scalable Alert Diagnosis for Large-scale Online Service Systems

Zhihan Jiang[¶], Jinyang Liu[‡], Yichen Li[‡], Haiyu Huang[¶], Xiao He[‡],
Tieying Zhang[‡*], Jianjun Chen[‡], Yi Li[‡], Rui Shi[‡], Michael R. Lyu[¶],
[¶]The Chinese University of Hong Kong, {zhjiang22, hyhuang25, lyu}@cse.cuhk.edu.hk
[‡]ByteDance, {jinyang.liu, liyichen.325, xiao.hx, tieying.zhang, jianjun.chen, liyi.ly, shirui}@bytedance.com

***Abstract*—Effective alert diagnosis is essential for ensuring the reliability of large-scale online service systems. However, on-call engineers are often burdened with manually inspecting massive volumes of logs to identify root causes. While various automated tools have been proposed, they struggle in practice due to alert-agnostic log scoping and the inability to organize complex data effectively for reasoning. To overcome these limitations, we introduce LogPilot, an intent-aware and scalable framework powered by Large Language Models (LLMs) for automated log-based alert diagnosis. LogPilot introduces an intent-aware approach, interpreting the logic in alert definitions (*e.g.*, PromQL) to precisely identify causally related logs and requests. To achieve scalability, it reconstructs each request's execution into a spatiotemporal log chain, clusters similar chains to identify recurring execution patterns, and provides representative samples to the LLMs for diagnosis. This clustering-based approach ensures the input is both rich in diagnostic detail and compact enough to fit within the LLM's context window. Evaluated on real-world alerts from Volcano Engine Cloud, LogPilot improves the usefulness of root cause summarization by 50.34% and exact localization accuracy by 54.79% over state-of-the-art methods. With a diagnosis time under one minute and a cost of only $0.074 per alert, LogPilot has been successfully deployed in production, offering an automated and practical solution for service alert diagnosis.**

***Index Terms*—Log Analysis; Failure Diagnosis; AIOps; Service Reliability; Large Language Models**

I. INTRODUCTION

With the rise of cloud computing, many traditional systems have migrated to cloud platforms as large-scale online services. These services support critical business operations globally, making reliability essential. Despite rigorous testing efforts, unexpected failures still occur in production, often causing significant disruptions and financial losses [1], [2], [3], [4].

To ensure service reliability, cloud operators deploy monitoring systems that collect low-level metrics such as per-request latency and response codes. These metrics are aggregated into Service Level Indicators (SLIs), such as request success rate or 99th percentile latency [5], [6]. SLIs are then continuously evaluated against predefined Service Level Objectives (SLOs) [7], [8], [9]. When violations occur, alerts are triggered and dispatched to on-call engineers (OCEs) for investigation.

While alerts indicate that something has gone wrong, they often reveal only the symptom of a deeper issue rather than its root cause. To resolve the underlying problem, engineers must perform root cause analysis (RCA). This process demands engineers to examine vast amounts of fine-grained monitoring data. Among these data, logs, which record detailed runtime events, are the most widely used and crucial diagnostic resource [10], [11], [4], [12]. For instance, one empirical study found that 93% of failures in distributed systems are manifested in logs, and logs are instrumental in diagnosing nearly 90% of faults [13]. This finding aligns with our own practical experience, where logs consistently serve as the primary source of evidence during investigations.

In current industrial practice, this log-based diagnostic procedure typically involves two phases (§ II-B): (1) *alert-driven log scoping*, in which engineers filter massive volumes of logs to extract relevant entries; and (2) *investigative log RCA*, which infers the underlying cause of the triggered alert. This manual process is time-consuming and labor-intensive, motivating a variety of automated log-based RCA approaches from both academia [14], [15] and industry [16], [17].

However, we find that existing automated approaches fall short in complex industrial scenarios due to fundamental limitations in both phases. First, for *log scoping* (§III-A), current methods like keyword searches or anomaly detection are alert-agnostic [18], [19], [20]. Lacking the specific context of the alert, they either overwhelm engineers with irrelevant logs or miss critical evidence entirely. Second, for *investigative log RCA* (§ III-B), traditional pattern-learning techniques [14], [21], [12] are impractical. They rely on large, manually labeled datasets of past failures, which are costly to create and maintain. While recent LLM-based approaches [22], [23], [15] show promise, their practical application is hindered by ineffective log data organization. This either leads to log data volumes exceeding the LLM's context window size or presents interleaved and fragmented event information, thereby impairing the model's reasoning capabilities.

To bridge these gaps, we identify two key opportunities for building a practical and automated alert diagnostic framework. First, we recognize that an intent-aware diagnostic process is achievable by leveraging an alert's definition logic (e.g., Prometheus Query Language (PromQL) expression) to precisely scope the logs relevant to the triggered alert. Second, we observe that the diagnosis can be made scalable even when the volume of scoped logs is large. This is because the underlying execution paths are not random but follow recurring patterns [24], [17]. We can distill these patterns into a

*Tieying Zhang is the corresponding author.

few representative examples, which are both compact enough for an LLM's context window and rich enough to support accurate, holistic reasoning about the root causes.

Motivated by these insights, we propose LogPilot, to our knowledge, the first intent-aware and scalable framework powered by LLMs for automated log-based alert diagnosis in large-scale online services. LogPilot consists of three key phases. First, in the *Intent-Aware Log Scoping* phase, an alert-log correlation agent interprets the semantic intent of alerts, typically encoded in PromQL expression, to automatically generate a tailored log filtering tools. These tools extract the logs and corresponding request IDs that are causally linked to the alert, ensuring high precision in log scoping. Second, the *Request-Centric Log Chain Processing* phase reconstructs a coherent execution for each request by parsing raw logs into structured events and organizing them into spatiotemporal chains. This representation captures the end-to-end execution flow across distributed components, facilitating deeper diagnostic reasoning. Finally, in the *Clustering-Based LLM Diagnosis* phase, LogPilot groups requests based on the similarity of their log event patterns and selects a representative request from each cluster for detailed analysis by a log-based RCA agent. This approach ensures that the diagnostic input remains within the LLM's context window while retaining essential diagnostic clues. The identified root causes are then aggregated by a diagnostic summary agent, which synthesizes a final, comprehensive alert diagnosis report.

We conducted an extensive evaluation of LogPilot on real-world alert data from large-scale production services at Volcano Engine. The results show that LogPilot significantly outperforms the state-of-the-art baselines [22], [23], improving root cause summarization by 50.34% (human-evaluated Usefulness) and root cause localization by 54.79% (Exact Match). With a response time of under one-minute and an approximate cost of only $0.074 per alert, LogPilot proves both practical and scalable. Furthermore, LogPilot has been successfully deployed in our production services, and we share our experience and practical insights from this deployment.

In summary, the main contributions of this paper are:

- We investigate the gap between existing log-based diagnosis research and industrial practice, identifying key opportunities for building practical, automated solutions (§III).
- We introduces LogPilot, the first intent-aware and scalable LLM-based framework that diagnoses service alerts by correlating and analyzing massive volumes of logs (§IV).
- We extensively evaluate and successfully deploy LogPilot on our production systems, demonstrating its effectiveness, scalability, and practical value (§V).

II. Background

A. Online Service Monitoring and Automated Alerting

Modern large-scale online services are built as complex, distributed systems where reliability is essential to business success. To uphold this reliability, operators rely on advanced monitoring infrastructure that continuously tracks the health and performance of the system.

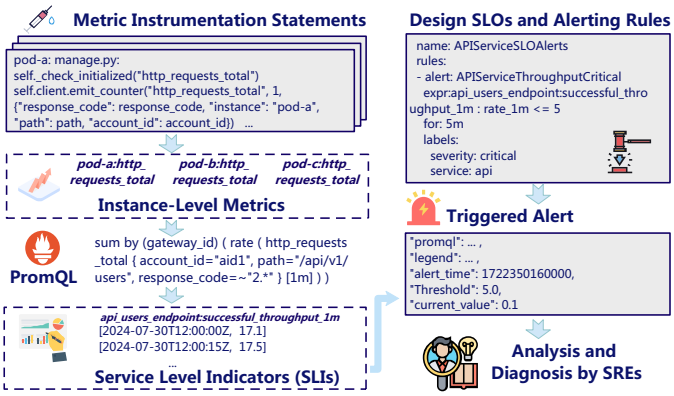

Fig. 1: An example service monitoring and alerting.

Service Metric Collection and Aggregation. As shown in Fig. 1, gaining visibility into service health begins with instrumentation, where developers embed code to emit performance metrics at runtime. For instance, a web server might expose a counter like `http_requests_total`, using labels to add dimensions such as the HTTP status code (`code="403"`) or the endpoint (`path="/api/v1/users"`). These low-level metrics are then scraped by a system like Prometheus [25] and stored in a Time-Series Database (TSDB).

However, these raw metrics from individual instances are too granular to directly reflect overall service health. A single forbidden request on one server, for example, is mostly considered as noise; its significance emerges only when aggregated with data from all instances to calculate a service-wide error rate. These aggregated signals constitute the Service Level Indicators (SLIs) [5]. Operators use tools, such as the PromQL [26], to perform this aggregation, dynamically computing critical indicators like the request success rate from the raw time-series data according to defined business logic.

Service Alerting. Automated alerting is driven by rules that continuously evaluate SLIs against their corresponding Service Level Objectives (SLOs) [7]. SLOs define the acceptable range for SLIs, expressed as a target threshold (*e.g.*, SLI $\leq$ target) or a bounded interval (*e.g.*, lower bound $\leq$ SLI $\leq$ upper bound). These alerting rules are often managed by Prometheus, which evaluates PromQL expressions in real time. As illustrated in Fig. 1, an alert is triggered if a rule's condition is met for a predefined duration, *e.g.*, the 1-minute average for the request success rate drops below 95%. Once triggered, the rule enters a "firing" state and dispatches a notification to OCEs.

B. Current Industrial Alert Diagnosis Practice

While service alerts are effective at detecting anomalous symptoms, they rarely pinpoint the underlying causes. This responsibility consequently falls to OCEs, who perform RCA by manually investigating various observability data. Among these, logs, which provide a detailed, chronological record of system runtime behavior, are the most critical and are typically the first data source engineers utilized during diagnosis in industrial environments [13], [17]. This diagnostic process typically involves the following two distinct phases.

Phase 1: Alert-Driven Log Scoping. Diagnosis begins with OCEs forming an initial hypothesis using contextual information from the alert. Alerts typically include key details such as the affected service, the type of failure (*e.g.*, low success rate), and the anomalous timeframe. Based on this context, engineers manually query distributed log storage systems to retrieve logs that are potentially relevant, such as those containing specific error codes or keywords. This targeted querying helps reduce the search space from billions of log entries to a more manageable subset that is closely related to the alert.

Phase 2: Investigative Log RCA. Following the *log scoping* phase, OCEs proceed to analyze the retrieved subset of logs to infer potential root causes. However, this task is often complicated by the interleaved nature of log entries from numerous service requests, which can obscure the execution flow of any single request. Furthermore, these logs may only capture partial execution statuses, such as error results, without detailing the preceding events that could reveal the root cause. To obtain a comprehensive picture, OCEs first extract request IDs recorded in the suspicious logs, then use them to trace and search all related log entries across components. This process relies on request IDs being embedded in the logs, which is automatically handled by the structured logging framework [27], [28]. This practice is common across modern industrial systems (*e.g.*, Google and WeChat) and widely used frameworks (*e.g.*, Spring Cloud), where unique request IDs and other metadata are routinely injected into logs via structured logging [29]. These identifiers allow OCEs to correlate all logs associated with a single request as it traverses various system components. This correlation facilitates the reconstruction of the request's end-to-end execution path, enabling OCEs to reason about the underlying issues leading up to the alert.

III. Motivation

The manual, two-phase process for investigating service alerts, while thorough, is slow and labor-intensive. Although automated solutions exist, they frequently fall short when applied to the complexities of large-scale commercial systems. This disparity between the practical demands of alert response and the capabilities of current tools motivates our research. We frame the principal challenges and our identified opportunities within the two core phases of the diagnostic workflow.

A. Alert-Driven Log Scoping

Existing Approaches. Currently, OCEs rely on two primary methods to isolate relevant logs of an alert. The most common approach involves keyword searches using terms such as *error* or the name of an affected service. More advanced methods use statistical anomaly detection, training models on historical logs to identify deviations from normal patterns [18], [19], [30], [31], [20]. Both approaches aim to extract a small subset of relevant entries from an overwhelming volume of log data.

The Industrial Gap. A fundamental gap in industrial practice lies in the inability to reliably connect an alert's symptom to its underlying log data. This requires isolating the complete set of logs for each failed request (the "signal"), from the massive stream of logs generated by successful operations (the "noise"). In a large-scale system, this signal-to-noise ratio can be exceptionally low. Existing solutions fall short due to their **lack of alert intent awareness**. Whether based on keyword search or anomaly detection, they do not account for the specific semantics of the triggering alert. As a result, they often overwhelm engineers with irrelevant logs (false positives) or miss critical information (false negatives). Crucially, they cannot perform a targeted search for the specific failure condition defined by the alert.

Opportunity. We identify a critical and underexplored asset: the semantic intent embedded in the alert itself. The alert is defined by a PromQL expression (see Sec. II-A), which encodes key metrics, labels, and aggregation logic. We observe that these contextual cues in PromQL often align well with the log data generated by our services. Our internal analysis further shows that over **96.4%** of alerts include label fields (*e.g.*, account ID, gateway, error code, pod name, and endpoint), whose meanings are reflected in the logs, even if the exact field names differ. This underscores a valuable opportunity: Rather than relying on static rules or loose keyword matching, we can use the semantic signals encoded in the alert to guide precise log extraction. By automatically interpreting the alert's intent, our approach would precisely extract the necessary logs, creating a focused and complete diagnostic picture.

B. Investigative Log RCA

Existing Approaches. Once relevant logs are scoped, the next step is RCA. Traditional RCA approaches based on pattern learning [14], [21] often require large labeled datasets and generalize poorly to evolving or unseen failures. Recently, LLMs have shown promise due to their strong reasoning capabilities and ability to operate without task-specific training, offering a more adaptable solution.

The Industrial Gap. Despite their strengths, LLMs encounter two significant gaps in real-world industrial applications. The first is **scale and complexity**: In large-scale online services, a single failure alert may involve hundreds of log lines per request, and often spans thousands of failed requests. Existing approaches [22], [23], [15] lack effective strategies for organizing this vast and heterogeneous log data. As a result, the input to LLMs either exceeds their context window or consists of fragmented, interleaved events across requests, both of which undermine the model's ability to perform accurate and comprehensive reasoning. The second gap is the implicit assumption that logs contain all necessary information for a diagnosis. In practice, logs can be incomplete, inconsistently formatted, or entirely missing critical events [13], [32]. This assumption limits the effectiveness of existing approaches when the logs are insufficient for root cause identification.

Opportunity. Although raw logs are extensive and varied, the logs of specific request types often reveal consistent execution patterns, especially for requests that fail due to the same underlying root cause [17], [16]. By identifying these patterns, we can distill the raw logs of each request into compact and informative execution patterns. Selecting representative

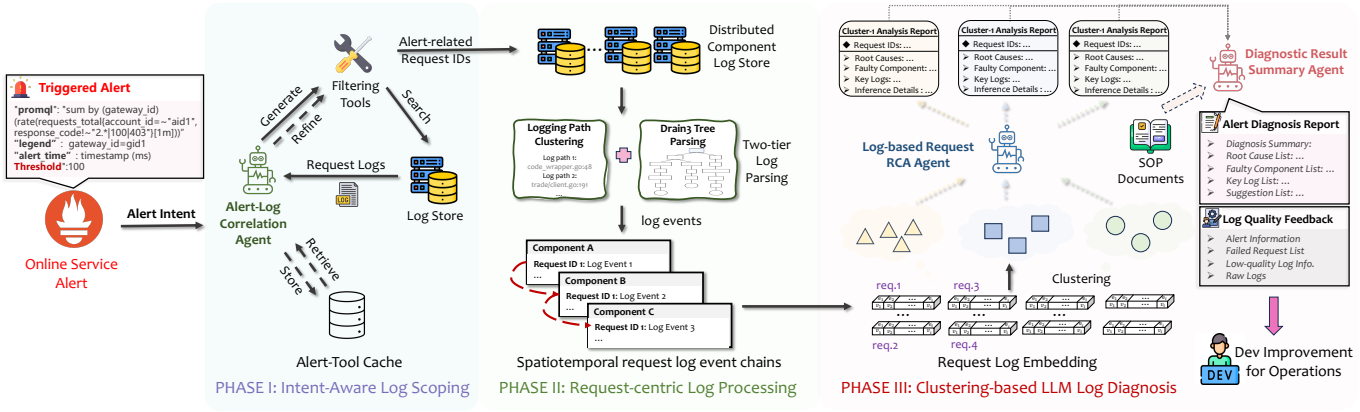

Fig. 2: The overall framework of LogPilot.

examples for each pattern offers compact diagnostic clues that well-suited to an LLM's context window, enabling more accurate and holistic reasoning. Furthermore, even if these patterns prove insufficient for diagnosis, LLMs can surface such limitations directly in the analysis. This provides actionable feedback to improve service observability.

IV. Methodology

A. Overview

To bridge the above industrial gaps, we propose LogPilot, an intent-aware and scalable log-based framework that leverages LLMs for automated and scalable alert diagnosis in large-scale online services. The overall framework of LogPilot, depicted in Fig. 2, operates in three sequential phases.

The first phase, *Intent-Aware Log Scoping*, is triggered upon alert activation. Here, an *alert-log correlation (ALC) agent* interprets the semantic intent behind the alert, typically encoded as PromQL expressions, and automatically generates customized, lightweight and executable log filtering tools for each alert. These tools retrieve logs and extract request IDs that are causally linked to the alert.

In the second phase, *Request-Centric Log Chain Processing*, LogPilot reconstructs detailed log event chains for each relevant request. This is accomplished by first parsing raw logs into structured log events using a two-tier parsing module. The parsed events are then chronologically and spatially organized into log chains that preserve inter-component execution flows, thereby enabling comprehensive request-level analysis.

Finally, the *Clustering-Based LLM Diagnosis* phase, focuses on efficient and accurate RCA. To recognize the common execution pattern of alert-related requests, LogPilot clusters them based on the similarity of their log chain embeddings. A representative request from each cluster is selected and analyzed by an *LLM-based request RCA agent* to identify underlying issues. The resulting insights are synthesized by a *diagnostic result summary agent*, which produces a comprehensive alert diagnosis report.

B. Intent-aware Log Scoping

Once an alert is triggered, the first step is to collect and filter logs that are related to the investigation. As highlighted in Sec. III-A, conventional methods that rely on static rules or anomaly detectors often produce an overwhelming volume of irrelevant logs or, conversely, omit crucial information.

To overcome these limitations, LogPilot introduces an *ALC agent* that interprets the semantic intent behind an alert and extracts log data with high precision. The key insight is that the logs from service requests that directly triggered the alert are more informative for analysis. This causal linkage can be inferred from the alert's definition logic, most commonly expressed in PromQL, as introduced in Sec.II-A. For instance, Fig. 3 illustrates an alert that triggers when the one-minute growth rate of requests with specific *gateway_id*, *account_id* and response code surpasses a threshold. As explored in Sec.III-A, such contextual information can guide the correlation between alerts and logs. However, due to diverse logging schemas across different services, naive schema-based matching is infeasible (see Fig. 3). To overcome this obstacle, we leverage LLMs to heuristically interpret and align the semantic content of alerts and logs. However, directly using LLMs to understand alert intent and process large volumes of raw log messages is computationally expensive. To address this, we instead harness their strong code generation capabilities [33], [34], [35], [36], [37] to automatically generate lightweight, customized log filtering tools tailored to each alert.

1. Tool Generation. The ALC agent is prompted to generate a runnable log filtering tool using three key inputs, as shown in Fig. 3: (1) *Alert definition*: The detailed definition and generation logic of the alert; (2) *Log Examples*: A random sampling of logs to illustrate the specific service schema. (3) *Domain Specific Language (DSL) Grammar*: The documentation details the grammar and usage of our log query language. By integrating these inputs, the agent generates a tool (*i.e.*, a Python script with DSL query) that retrieves logs from the relevant time window, matching the exact criteria specified in the alert. This tool then extracts the request IDs causally linked to the alert from these logs.

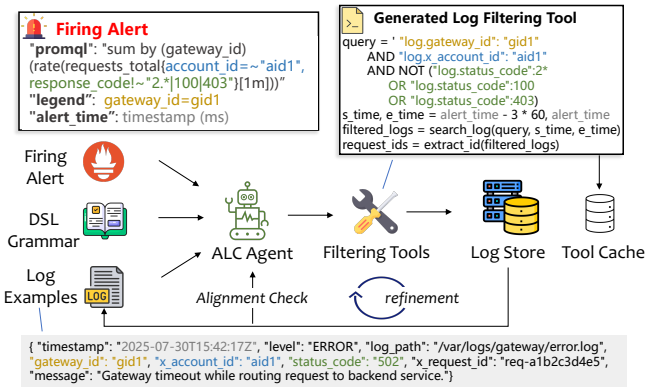

Fig. 3: An example workflow of *alert-log correlation* agent

An example of such a generated tool is shown in Fig. 3. This LLM-based approach enables dynamic adaptation to heterogeneous alert definitions and service log schema, eliminating the need for manually writing or maintaining filtering logic across a large and evolving alert surface.

2. Tool Refinement. Following the generation of a log scoping tool, it is used to query the log store to extract key logs and request IDs. However, one-shot LLM tool generation can be unreliable due to the inherent stochasticity of LLMs [38], [39]. Consequently, we introduce a feedback-based iterative refinement process to ensure the high quality of the generated tools. Specifically, both the query results and any execution errors are treated as feedback. This feedback enables the agent to perform an alignment check: it verifies whether the retrieved logs conform to the alert logic. If a mismatch occurs, the agent refines the tools accordingly. The refinement loop repeats until alignment is achieved or the maximum number of iterations (set to 3) is reached. In rare cases (less than 3% in deployment) where no valid tools emerge, human experts intervene to create tools for subsequent use and caching.

3. Tool Caching. Once a tailored log filter tool is validated, it can be cached for future reuse. Specifically, if a new alert with an identical PromQL definition is triggered, the corresponding tool can be retrieved from the cache and applied directly, bypassing the need for regeneration. This caching mechanism reduces LLM invocation overhead and enhances response latency, especially in high-frequency alerting environments.

C. Request-centric Log Processing

In real-world production systems, log data from service requests are often interleaved and scattered across distributed component log stores, which complicates analysis. To address this, we organize this complex data in a request-centric manner, mirroring the process of manual diagnosis.

1. Two-tier Log Parsing. After filtering for alert-related service requests, we use the request IDs to concurrently collect all raw logs for each request from distributed service components. Raw logs are typically unstructured and substantial, impeding automated analysis. Consequently, log parsing has been widely adopted as a critical prerequisite to convert them into structured log events [40], [11], [41], [42], [43].

In production systems, we observe that in most log sources, logging paths with the exact code lines are contained within the logs. However, logs with the same logging path can still record different log events due to the dynamic propagation and construction of logging variables [44], [45], [46], [15], [43], [47], [48]. Therefore, we employ a two-tier log parsing approach. First, we extract the logging path from each raw log and perform a coarse-grained clustering based on this path. Log messages without a path are grouped together. Then, within each coarse-grained group, we use the Drain [49] algorithm to heuristically parse the raw logs into structured log events.

2. Request Log Event Chain Construction. To support effective reasoning by LLMs about system behavior, it is critical to structure log data in a manner that captures the interaction patterns among system components. To this end, we construct a spatiotemporal log event chain for each request, leveraging log timestamps and component identifiers. This representation preserves both the chronological execution flow and the invocation relationships across components.

Formally, let the log event list for a single request be denoted as $L = \{e_1, e_2, \cdots\}$. To reduce the volume of log data and ensure it remains within the context window of modern LLMs, we deduplicate this list by retaining only the first occurrence of each unique log event and then sort the result chronologically: $L' = \{e'_1, e'_2, \cdots\}$. We then partition the sorted list L' into segments based on the associated service components, yielding a chain structure that spans multiple components: $C_1 = \{e'_1, e'_2, \cdots, e'_i\}$, $C_2 = \{e'_{i+1}, e'_{i+2}, \cdots\}$, and so on. The ordering of these component segments is determined by the timestamp of their earliest log event, thereby preserving the actual invocation sequence across services.

D. Clustering-based LLM Diagnosis

After processing the logs for each request into an event chain, the next step is RCA. However, diagnosing all requests is impractical due to the cost, inefficiency, as well as the limited LLM context issue. Based on the observation that requests with similar log event patterns often stem from the same underlying issue. This insight allows for a significant reduction in the diagnostic workload, as a single alert can typically be traced to a small number of root causes [17], [50], [51], [23]. Consequently, LogPilot focuses on analyzing representative request for each distinct execution pattern.

LogPilot achieves this by employing a clustering-based approach, which consists of several stages. First, LogPilot vectorizes the log event chain of each request to create a log pattern embedding. Next, it groups these requests into clusters based on embedding similarity. Then, within each cluster, a *log-based request RCA agent* analyzes one representative request to identify the root cause. Finally, the *diagnostic result summary agent* aggregates the results from all clusters to produce a final comprehensive diagnosis report for the alert.

1. Request Log Embedding. To quantitatively represent each request's pattern for similarity measurement, we transform the log data into a vector embedding following previous work [52], [53], [54]. Specifically, the embedding for a single request is represented as a vector $V = [v_1, v_2, \cdots, v_n]$, where n represents the total number of distinct log events across all requests, and v_i denotes the count of the i-th log event. This representation captures both the presence and frequency of log events, providing a holistic view of request log patterns.

To assess the similarity between these embeddings, we use a log-scaled cosine similarity metric. This approach mitigates the issue where high-frequency events can disproportionately dominate the similarity score, thereby better capturing the underlying structural log patterns. First, for any request embedding vector $V = [v_1, v_2, \cdots, v_n]$, we apply a logarithmic scaling transformation to produce a new vector $V' = [v'_1, v'_2, \cdots, v'_n]$, where each element is calculated as: $v'_i = \log(1 + v_i)$. Subsequently, the similarity between two requests, represented by their scaled vectors V'_A and V'_B, is computed using the standard cosine similarity formula:

$$\text{Similarity}(V'_A, V'_B) = \frac{V'_A \cdot V'_B}{\|V'_A\| \|V'_B\|} \tag{1}$$

This ensures that requests are represented based on their fundamental log patterns, making the process less susceptible to variations in system load.

2. Request Clustering. To identify distinct groups of alert-related issues, we employ the Hierarchical Agglomerative Clustering (HAC) algorithm on the request log embeddings. The algorithm initializes each request as a separate cluster and progressively merges the most similar pairs until their similarity score falls below a predefined threshold. We chose this algorithm because it does not require predefining the number of clusters. The resulting clusters represent groups of requests with similar log patterns, from which we can infer a shared underlying root cause. Based on our experience, the number of resulting clusters is typically small (under 10), as a specific alert usually has only a few root causes [51], [55].

3. Log-based Request RCA Agent This LLM agent performs RCA for each alert-related request cluster with a common execution pattern. Since analyzing every request in a large cluster is impractical, our framework mimics manual troubleshooting by selecting a single, representative request for analysis. To identify this request from a cluster of m requests, we first calculate the average log embedding vector (the centroid): $\bar{V} = \frac{1}{m} \sum_{i=1}^{m} V_i$. We then select the request whose log embedding is closest to this centroid, $\bar{V}$, as representative request for further detailed analysis.

Leveraging the analytical and reasoning capabilities of LLMs, the *Log-based Request RCA Agent* analyzes the selected request's lifecycle to identify the root cause of the issue. As shown in the example prompt in Fig. 4, the agent takes the detailed alert information, the request's log chain, and service component information as input. It then generates an analysis report that includes the identified root causes, the faulty component, key log evidence, and inference details. Furthermore, considering that logging quality can vary, the agent is designed to handle cases where logs are insufficient or inconsistent to determine the root cause (*e.g.*, error information is not transmitted or logged). In such situations, it identifies and reports this poor logging quality in its analysis.

Log-based Request RCA Prompt

You are an expert specializing in log analysis and root cause diagnosis for online service systems.
Context
You are diagnosing a production alert based on the following data:
1. **Triggering Alerts**: *{detailed alert information}*
2. **Alert-related Request Log Chain**: *{request log chain across distributed components}*
3. **Service Component Information**: *{service component functionality}*
Instruction
1. **Analyze the Request Execution Path**: Analyze the Request Log Chain from start to finish. Find any abnormal behaviors like high latency, error status codes, or unexpected responses.
2. **Identify the Root Cause**: Pinpoint the specific service component that is the source of the failure. Isolate the key log entries that serve as direct evidence of the root cause.
3. **Handling Low Logging quality**: If the provided logs are insufficient to determine the root cause, or if there are clear inconsistencies, please provide the feedback with detailed information.
4. **Generate the Analysis Report**: Synthesize your findings into the RCA Report Format specified below. Your reasoning should be clear, concise, and directly supported by the provided log data.
Output Format
{analysis report format} and {feedback format}

Diagnostic Result Summary Prompt

You are an expert specializing in alert diagnosis for online service systems.
Context
You are diagnosing a production alert with the following information: *{detailed alert information}*
An automated analysis system has analyzed the root cause issues of the alert-related service requests, with several scattered request clusters: *{analysis reports from all request clusters}*
Here are also some related standard operation procedure documents: *{retrieved historical SOP}*
Task
1. **Review report and summarize root causes**: Review all analysis reports, merge root causes that are fundamentally identical in nature, and summarize the root causes for the alert.
2. **Generate final alert diagnosis report**: Synthesize your findings into the final diagnosis report for this alert following the format specified below.
3. **Generate log quality feedback**: If there are any low log quality issues in the analysis reports, please also give a summarized feedback for developer to improve.
Output Format
{alert diagnosis report format} and {log quality feedback format}

Fig. 4: Example prompt templates for *log-based request RCA* and *diagnostic result summary* agent.

4. Diagnostic Result Summary Agent The RCA for each cluster is performed independently and in parallel. Consequently, the findings from different clusters can be interrelated or contain overlapping information (*e.g.*, they may stem from the same root cause but exhibit different symptoms). To address this, LogPilot incorporates a *Diagnostic Result Summary Agent* to aggregate these disparate results. As illustrated in Fig. 4, this agent takes alert information and all individual analysis reports as input, and synthesizes them into a final, comprehensive alert diagnostic report, including a summary, a unified list of root causes, faulty components, and key logs.

Furthermore, to provide domain-specific insights and actionable recommendations, we use a knowledge base of historical Standard Operating Procedure (SOP) documents. Employing the Retrieval-Augmented Generation (RAG) paradigm, the agent retrieves the most relevant SOPs based on the identified root causes of all request clusters. This context is then supplied to an LLM to generate a list of suggestions for resolving the alert. Additionally, the agent identifies instances of poor logging quality into a feedback report for developers, highlighting opportunities for improving future DevOps processes. More detailed cases can be found in Sec. VII.

V. Evaluation Setup

We first evaluate LogPilot by answering the following research questions (RQs):

- **RQ1:** How effective is LogPilot in alert diagnosis?
- **RQ2:** How robust are the components of LogPilot?
- **RQ3:** What is the efficiency and cost of LogPilot?

We also share our deployment experience of LogPilot in the Volcano Engine Cloud systems (§ VII).

A. Dataset Collection

To assess the effectiveness of our proposed method, we collected alert data from four large-scale online services at our company, each serving millions of users globally. The data collection period spanned from June 15 to July 15, 2025. From this initial set, we randomly sampled a subset of alerts that can be manually analyzed. This resulted in a final dataset of 202 alerts, with detailed statistics presented in Table I.

TABLE I: Evaluated alerts and unique root causes (URCs)

Datasets	Service $\mathcal{A}$	Service $\mathcal{B}$	Service $\mathcal{C}$	Service $\mathcal{D}$	Total
# Alerts	35	40	52	75	202
# URCs	27	29	43	62	161

B. Dataset Labeling

The complete log data corresponding to each alert's timeframe is archived in our internal log storage, and a large portion of the selected alerts were manually diagnosed by engineers at the time of their occurrence and formally documented in failure review reports. The manual annotation of our dataset was conducted by a team of four annotators. This team comprised two Ph.D. students with over two years of research experience in system operations, and two industry engineers, each with more than five years of experience in software development and maintenance. For each alert, following previous work [15], the annotators carefully reviewed the available diagnostic data and labeled from two aspects:

- **Root Cause Summarization**: A concise description of the root cause was formulated through a collaborative discussion among the annotators. The primary objective of this summary is to provide a clear and actionable guide for system maintainers to diagnose similar issues in the future.
- **Root Cause Localization**: The specific components of the service identified as the root cause were pinpointed. These components are also reflected in the fields of the log messages. While most cases were attributed to a single root cause component, some alerts involved two or three components being identified as the source of the issue.

C. Evaluation Metrics

- **Root Cause Summarization:** To evaluate the quality of root cause summary, following prior research [56], [57], [58], [15], we use the following metrics: (1) *Automated Evaluation*: *METEOR* [59] and *ROUGE* [60] are employed to measure the textual similarity between generated summaries and the ground truth. To capture contextual meaning beyond lexical overlap, we also computed *Semantic Similarity* by embedding the summaries using the *Doubao-1.5-Embedding* model [61]. (2) *Human Evaluation*: We assessed the *Usefulness* of the summaries, defined as how accurately they explain the cause of an issue. Evaluations were conducted by highly experienced production engineers. For each issue, evaluators reviewed both the ground truth and the corresponding summaries from all methods, assigning a usefulness score ranging from 0 to 1. The final score for each baseline is averaged across all evaluators.
- **Root Cause Localization:** We measure localization accuracy using: (1) *Exact Match*: This metric measures the percentage of instances where the predicted set of root cause components is exactly identical to the ground-truth set. (2) *Top-3 Accuracy*: This metric assesses whether the set of ground-truth components is fully contained within the top-3 most likely components proposed by the model.

D. Baselines

We exclude traditional log diagnosis methods [62], [18], [19], [31] due to their reliance on labeled training data and limited explainability, two critical drawbacks that make them unsuitable for production systems. Recently, many studies [63], [64], [65] have explored the use of LLM for RCA in online services. We compare LogPilot against the following state-of-the-art methods with the same LLM backend:

- **LLM with sampling** [23]: This method directly feeds downsampled log data to an LLM for RCA.
- **RCA Agent** [23]: A multi-agent system that enables an LLM to write and execute code to process large volumes of log data, identifying root causes from the processed results.
- **RCACopilot** [22]: This approach uses alert information, corresponding log data, and retrieved historical SOPs to perform RCA. To accommodate the LLM's context window, we only input logs with a *warning* or *error* level.

E. Implementation Details

LogPilot is implemented as an internal service that integrates with our alert system. It leverages an internal log store search engine to query and filter log data from distributed services components. The similarity threshold for request clustering is empirically set to 0.7, with its performance impact evaluated in RQ2. For the LLM backend selection, LogPilot is powered by the latest *Doubao* series on the Volcano Engine platform through the official API provided. Specifically, the *alert-log correlation* and *log-based request RCA* agents utilize the *Doubao-Seed-1.6-thinking* model to leverage its strong reasoning capabilities, while the *diagnostic result summary* agent defaults to the *Doubao-Seed-1.6-flash* model which prioritizes speed for lightweight tasks.

VI. Evaluation Results

A. RQ1: Effectiveness of Alert Diagnosis

The evaluation results of both root cause summarization and localization dimensions are presented in Tab. II.

Root Cause Summarization. In the task of root cause summarization, LogPilot demonstrates remarkable performance improvements over existing baselines across automated and human-evaluated metrics. Regarding automated metrics, compared to the strongest baseline, RCA Agent, LogPilot achieves average gains of 20.75% in ROUGE-1, 27.37% in METEOR,

TABLE II: Root cause analysis results from both *summarization* and *localization* dimensions for each service.

System	Model	Root Cause Summarization				Root Cause Localization	
		ROUGE-1	METEOR	Semantics	Usefulness	Exact Match	Top-3 Acc.
Service $\mathcal{A}$	LLM w/ BS	0.381	0.303	0.713	0.454	0.343	0.543
	RCACopilot	0.468	0.374	0.795	0.563	0.429	0.686
	RCA Agent	0.475	0.369	0.812	0.640	0.457	0.743
	LogPilot	**0.587**	**0.522**	**0.894**	**0.823**	**0.743**	**0.943**
Service $\mathcal{B}$	LLM w/ BS	0.375	0.322	0.719	0.473	0.400	0.475
	RCACopilot	0.483	0.397	0.804	0.570	0.450	0.725
	RCA Agent	0.512	0.408	0.831	0.668	0.475	0.775
	LogPilot	**0.602**	**0.497**	**0.913**	**0.851**	**0.725**	**0.950**
Service $\mathcal{C}$	LLM w/ BS	0.352	0.309	0.688	0.427	0.346	0.481
	RCACopilot	0.413	0.346	0.755	0.494	0.423	0.673
	RCA Agent	0.432	0.367	0.808	0.588	0.442	0.747
	LogPilot	**0.538**	**0.460**	**0.852**	**0.785**	**0.712**	**0.885**
Service $\mathcal{D}$	LLM w/ BS	0.401	0.356	0.748	0.492	0.387	0.533
	RCACopilot	0.517	0.437	0.821	0.609	0.507	0.733
	RCA Agent	0.538	0.446	0.846	0.643	0.560	0.787
	LogPilot	**0.631**	**0.539**	**0.924**	**0.895**	**0.800**	**0.960**

and 8.66% in semantic similarity. These results demonstrate the high quality of the summaries generated by LogPilot. Furthermore, in our human evaluation, the Usefulness metric directly measures the practical benefit of a summary to system maintainers. In this crucial assessment, LogPilot obtains an average improvement of 50.34% over RCACopilot, underscoring its ability to produce not just accurate, but also clear and actionable diagnostic information.

Root Cause Localization LogPilot's performance in root cause localization also achieves state-of-the-art, where it markedly surpasses all baseline methods in identifying the faulty components associated with an alert. The best-performing baseline, RCA Agent, is consistently outperformed by LogPilot across all systems and metrics. On average, LogPilot delivers a 54.79% higher score in Exact Match and a 22.49% higher score in Top-3 Accuracy. The most significant performance gap is observed in the Top-3 Accuracy for Service A, where LogPilot's score is 26.92% greater than RCA Agent's. These findings highlight the precision of our approach in pinpointing the faulty components.

B. RQ2: Robustness of Individual Components

Robustness of Intent-aware Log Scoping. To evaluate the robustness of the log scoping process, we randomly sample 60 alerts from our datasets and manually assess the quality of the generated log filter tools. Each tool is assigned a score of 0 if it fails to execute or returns empty results (*e.g.*, due to error conditions). Otherwise, a human annotator rate its quality on a scale from 0 to 1, based on inspection of both its logic and execution results. We compared the original alert-log correlation agent with two ablation settings: LogPilot *w/o log examples* and LogPilot *w/o tool refinement*.

Fig. 5-(a) summarizes the results: (1) Without illustrative log examples for the LLMs, the quality of the generated tool is significantly diminished and largely unusable. This is primarily due to a lack of knowledge regarding the specific service's log schema. (2) In the absence of the feedback-based tool refinement procedure, the agent is prone to generating unreliable tools, for instance, by mismatching the log schema with PromQL conditions. This results in an average quality score of only 0.727. (3) With all designs integrated, our LogPilot achieves a high average score of 0.984 in the log filtering tool quality assessment, demonstrating its effectiveness in correlating alerts with logs by generating executable tools.

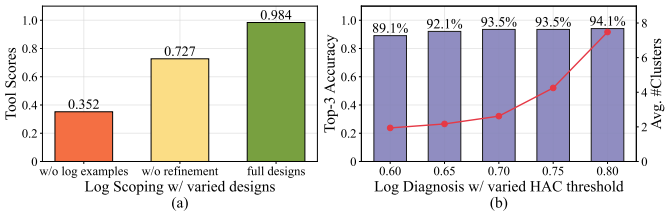

Fig. 5: Robustness evaluation of different phases in LogPilot.

Robustness of Clustering-based LLM Diagnosis. The clustering step is critical for accurate LLM-based diagnosis and is controlled by a manually set threshold θ_{HAC}. We performed a sensitivity analysis to examine its impact, measuring both the system's top-3 RCA accuracy and the average number of request clusters as θ_{HAC} varies.

As shown in Fig. 5-(b): (1) The top-3 accuracy of LogPilot remains stable and high (around 94%) for θ_{HAC} values ranging from 0.60 to 0.80. (2) Increasing θ_{HAC} produces more fine-grained clusters, leading to a greater number of analyzed requests and potentially more LLM invocations. Thanks to the aggregation capabilities of our summarization agent, overall RCA performance is maintained. (3) Reducing θ_{HAC} results in fewer clusters and a potential loss of valuable diagnostic information, slightly lowering accuracy (*e.g.*, to 89.1% when θ_{HAC}=0.60). Based on this trade-off between diagnostic cost and performance, we select $\theta_{\text{HAC}} = 0.7$ as the default threshold.

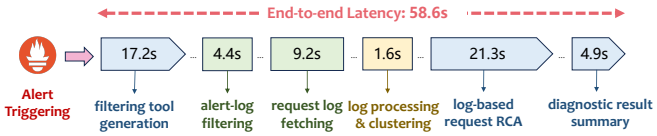

Fig. 6: Average E2E and stage-wise latency of LogPilot.

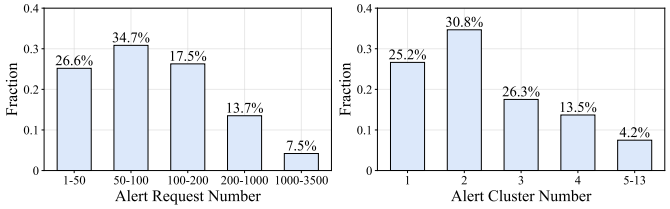

Fig. 7: The distribution of request and cluster number.

C. RQ3: Efficiency and Cost of LogPilot

In production systems, the high volume of alerts and the demand for timely responses make RCA efficiency critical. To evaluate LogPilot in this context, we scale the number of alerts to 548. Notably, this assessment does not require ground-truth root causes, allowing us to focus solely on efficiency and cost.

Efficiency. We evaluate efficiency by measuring the average processing time across the different stages of LogPilot, with a breakdown shown in Fig. 6. Our results show that LLM calls dominate the overall runtime. Specifically, generating the log filtering tool and executing the log-based RCA procedure take an average of 17.2s and 21.3s, respectively. The final summarization step, using a streamlined model with shorter prompts, completes in just 4.9s on average. Despite this reliance on LLMs, the total end-to-end latency remains modest at 58.6 seconds per alert, staying well below the one-minute mark. This represents a substantial improvement over manual diagnosis, which can take several minutes to tens of minutes. Moreover, caching the log filtering tool for recurring alert types can further reduce latency in future diagnoses.

Cost. To evaluate economic viability, we first analyze the number of filtered requests and resulting request clusters per alert (Fig. 7). Although a single alert may associated with over 3,000 requests (mean: 198.65), clustering reduces the number of required LLM invocations to a maximum of 13 (mean: 2.56), achieving a 98.71% reduction. Furthermore, token usage statistics show that, per alert, LogPilot consumes an average of 69.73K prompt tokens and 8.08K response tokens. According to current pricing [66], this results in an estimated cost of $0.074 per alert for diagnosis. Given the critical importance of rapid and accurate alert handling in production systems, we consider this cost both acceptable and justified.

VII. Deployment Experience

Performance in Production. Since June 2025, LogPilot has been deployed across 12 production services within the Volcano Engine Cloud systems. Fig. 8 shows how we have integrated LogPilot with existing service monitoring systems.

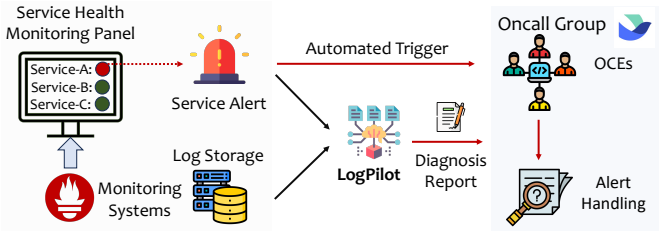

Fig. 8: The production deployment workflow of LogPilot.

Upon an alert trigger, a dedicated on-call group is automatically created via Lark to notify related OCEs. Simultaneously, LogPilot autonomously initiates its three-stage RCA process, generating a diagnostic report with actionable insights to assist OCEs in alert triage and handling. By July 31, 2025, LogPilot had analyzed over 3,500 production alerts and provided RCA results to assist OCEs. The practical effectiveness of LogPilot is demonstrated by user feedback from system architects and engineers. The overall acceptance rate for the generated reports is **84.21%**. Of these, 60.53% of the RCA results were identified completely correct (an exact match with the ground truth) and 23.68% were partially correct (we identified a subset of the ground-truth root causes). Even in the remaining 15.79% of cases, the inference details of LogPilot were still reported to offer valuable insights for subsequent manual investigation. These statistics highlight LogPilot 's practical effectiveness in production environments.

Enhancing Log Quality and Bridging Dev-Ops. Beyond alert diagnosis, LogPilot provides feedback on log quality, helping to identify observability gaps and recurring logging anti-patterns for developers. A common scenario identified is the "silent failure", where a request returns an error code, yet the corresponding logs across all traversed components are recorded at the info level with no indication of abnormality. Such logging practices severely hinder manual diagnosis. Furthermore, LogPilot detects log inconsistencies within the request execution lifecycle. For instance, as depicted in Fig. 9, LogPilot identified a case where the cloud control plane's request to create a node pool failed due to an ECS image type validation error. This specific error, however, was not propagated to the originating service and was instead logged as a *info* level by the Kubernetes Engine API Server. Concurrently, the EBS component, despite returning a success code, anomalously logged at the *warn* level with exit code 3. LogPilot successfully flagged these discrepancies and surfaced potential reliability and observability issues that merit further developer attention. This automated feedback loop represents a step toward identifying software reliability issues and bridging the gap between Development (Dev) and Operations (Ops). To formalize this process, we have incorporated log quality into the evaluation of engineering discipline across development teams from diverse services. Over time, this encourages more consistent and comprehensive logging practices, ultimately improving service maintainability.

Enriching Alert Context and Future Work. We believe

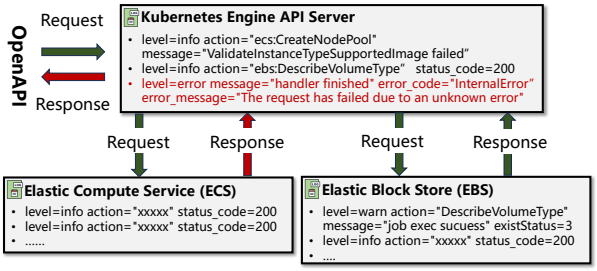

Fig. 9: A real-world example of inconsistent logging case.

LogPilot, as an automated framework that integrates high-level alerts with low-level logs, fundamentally enriches alerts with deeper system context and analytical insights. This capability paves the way for improving alert management in large-scale online service systems. Specifically, for a given alert type, the system state at each trigger can vary considerably, even if the triggering rules are identical. By augmenting alerts with the identified root causes derived from real-time underlying log data, we can provide more granular contextual information, *e.g.*, the actual root causes. This, in turn, facilitates more precise alert aggregation and the development of more fine-grained alert-handling SOPs. This area constitutes a primary focus for our future research and development efforts.

VIII. Discussion

A. Lessons Learned

From Definitive Conclusions to Actionable Insights. Prevailing studies in RCA [67], [14], [23] traditionally focus on delivering a single, definitive conclusion. However, our practical experience in industrial production environments indicates that the inherent complexity of these systems precludes any single RCA method from achieving complete reliability. This limitation often hinders the trust of OCEs for automated diagnoses. Consequently, we advocate for a paradigm shift: moving beyond sole root cause verdicts to the provisioning of actionable diagnostic insights. We argue that the ability to profile system behavior and distill critical clues from vast telemetry data is of paramount importance. Our experience and feedback from LogPilot's users confirm this: the failure clues and inference details LogPilot provides enable OCEs to rapidly comprehend the system's state and derive actionable insights. The advent of LLMs presents a valuable opportunity to further develop and scale this paradigm.

From Isolated Signals to Correlated Observability Data. While LogPilot effectively bridges the gap between high-level alerts and underlying runtime logs, modern online services generate a diverse source of observability and failure management data, including metrics, traces, oncall records, and so on. These data sources often exist in silos. The dynamic correlation and fusion of these heterogeneous signals are crucial for comprehensive system diagnosis and operations, yet this remains a persistent challenge in the industry. Addressing this gap effectively necessitates the co-design of both data infrastructure and analytical algorithms. We believe the strong pattern recognition and reasoning capabilities of LLMs open promising new research directions toward achieving unified and intelligent system observability.

B. Generalizability

Our proposed LogPilot, is designed for broad applicability across diverse online service systems, offering insights to both the research and practitioner communities. First, LogPilot is founded on widely adopted observability data sources: alert information (*e.g.*, PromQL definitions [26]), and fundamental runtime logs. Given that Prometheus has become the de facto standard for monitoring in modern cloud systems [25], and runtime logs represent the most basic source of observability data, LogPilot can be readily integrated into various online services. Second, LogPilot is architected around request-centric analysis, a methodology that reconstructs log event chains at the request level and employs clustering techniques to extract meaningful runtime patterns. This methodology is broadly applicable, as request-centric observability remains a core paradigm in diagnosing complex online services [68], [69], [67]. Third, the generalization capabilities (*e.g.*, in-context learning ability) of LLMs [70], [63] further enhance LogPilot's adaptability. By incorporating system-specific documentation into the agent's retrieval-augmented context, LogPilot can be tailored to new systems with minimal human effort. Finally, while our current deployment of LogPilot utilizes our internal Doubao-series LLMs due to privacy considerations, prior work has shown that various LLMs offer strong comprehension and reasoning capabilities [22], [71]. Consequently, LogPilot can be flexibly deployed with various open-source or commercial LLMs, facilitating its broader adoption in industrial settings.

IX. Related Work

Log-based Diagnosis. Alert or failure diagnosis using logs is vital for ensuring the reliability of software systems. It generally involves two stages: log scoping and RCA. Log scoping reduces the vast volume of logs to a relevant subset for analysis. Existing approaches [18], [19], [30], [31], [20] often apply anomaly detection to identify logs that deviate from normal patterns. For example, LogRobust [18] uses an attention-based Bi-LSTM to detect anomalies, while LogAnomaly [19] leverages a template2Vec embedding to capture log semantics. However, these methods typically ignore the context provided by alerts, limiting their ability to scope logs aligned with specific alert conditions. The second stage, log-based RCA, identifies the underlying issue of an alert. Prior work [14], [21] commonly relies on statistical correlation or pattern mining. For instance, FDiag [72] uses statistical metrics to associate anomalies with potential root causes, and LogRCA [21] adopts a semi-supervised approach for anomaly localization. Nonetheless, these techniques often struggle with the scale and complexity of log data in production environments.

Log Analysis and RCA in the LLM Era. Recent advances in LLMs have spurred efforts to enhance log analysis and RCA. Several works leverage LLMs' semantic understanding to interpret log data. For example, LogGPT [73] learns normal

log patterns using a GPT-based model and applies reinforcement learning (RL) to improve detection. Knowlog [74] integrates domain knowledge into pretrained LMs to enhance log comprehension. Other studies fine-tune LLMs for domain-specific RCA tasks. OASIS [75] adapts GPT-3 to summarize cloud incidents, while ThinkFL [76] employs RL to equip lightweight LLMs with reasoning abilities. To better utilize observability data, some approaches adopt retrieval-augmented generation (RAG) or agent-based architectures. RCACopilot [22] aggregates multi-source data for LLM-based RCA, and RCA-agent [23] generates diagnostic tools using LLMs. Despite promising results, these methods often struggle to structure the substantial diagnostic data effectively, leading to context window overflows and fragmented or interleaved inputs, which limits their scalability in real-world systems.

X. Conclusion

This paper presented LogPilot, the first intent-aware and scalable LLM-based framework for automated alert diagnosis via logs in large-scale cloud service systems. We identified key limitations in existing automated solutions, including imprecise, alert-agnostic log scoping and the inability to effectively organize massive volumes of log data for reasoning. To address these, LogPilot introduces two core innovations. First, it is intent-aware: it leverages the semantics of alert definitions to precisely extract causally related logs. Second, it is scalable: it reconstructs user requests into log chains, clusters them to uncover common execution patterns, and feeds compact, representative samples to the LLM for efficient diagnosis. Evaluations on production data show that LogPilot outperforms state-of-the-art baselines in both root cause summarization and localization. With diagnoses under a minute at $0.074, and successful production deployment, LogPilot offers a practical solution for enhancing service reliability.

XI. Acknowledgment

This work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. SRFS2425-4S03 of the Senior Research Fellow Scheme and No. CUHK 14209124 of the General Research Fund).

References

[1] "Aws post-event summaries," https://aws.amazon.com/cn/premiumsupport/technology/pes/, [Online; accessed 30 April 2025].

[2] "Google cloud status dashboard," https://status.cloud.google.com/summary, [Online; accessed 30 April 2025].

[3] "Azure status history," https://azure.status.microsoft/en-us/status/history/, [Online; accessed 30 April 2025].

[4] J. Huang, J. Liu, Z. Chen, Z. Jiang, Y. Li, J. Gu, C. Feng, Z. Yang, Y. Yang, and M. R. Lyu, "Faultprofit: Hierarchical fault profiling of incident tickets in large-scale cloud systems," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, 2024, pp. 392–404.

[5] S. Frey, C. Reich, and C. Lüthje, "Key performance indicators for cloud computing slas," in *The fifth international conference on emerging network intelligence, EMERGING*, 2013, pp. 60–64.

[6] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu, "Adaptive performance anomaly detection for online service systems via pattern sketching," in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 61–72.

[7] J. Ding, R. Cao, I. Saravanan, N. Morris, and C. Stewart, "Characterizing service level objectives for cloud services: Realities and myths," in *2019 IEEE International Conference on Autonomic Computing, ICAC 2019, Umeå, Sweden, June 16-20, 2019*. IEEE, 2019, pp. 200–206.

[8] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Wen, X. Ling, Y. Yang, and M. R. Lyu, "Graph-based incident aggregation for large-scale online service systems," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 430–442.

[9] Y. Li, X. Zhang, S. He, Z. Chen, Y. Kang, J. Liu, L. Li, Y. Dang, F. Gao, Z. Xu *et al.*, "An intelligent framework for timely, accurate, and comprehensive cloud incident detection," *ACM SIGOPS Operating Systems Review*, vol. 56, no. 1, pp. 1–7, 2022.

[10] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang *et al.*, "An empirical investigation of practical log anomaly detection for online service systems," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 1404–1415.

[11] S. He, X. Zhang, P. He, Y. Xu, L. Li, Y. Kang, M. Ma, Y. Wei, Y. Dang, S. Rajmohan, and Q. Lin, "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 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022.

[12] J. Huang, Z. Jiang, J. Liu, Y. Huo, J. Gu, Z. Chen, C. Feng, H. Dong, Z. Yang, and M. R. Lyu, "Demystifying and extracting fault-indicating information from logs for failure diagnosis," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*.

[13] W. Yuan, S. Lu, H. Sun, and X. Liu, "How are distributed bugs diagnosed and fixed through system logs?" *Inf. Softw. Technol.*, vol. 119, 2020.

[14] T. Wittkopp, P. Wiesner, and O. Kao, "Logrca: Log-based root cause analysis for distributed services," in *Euro-Par 2024: Parallel Processing - 30th European Conference on Parallel and Distributed Processing, Madrid, Spain, August 26-30, 2024, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. Carretero, S. Shende, J. García-Blas, I. Brandic, K. Olcoz, and M. Schreiber, Eds., 2024.

[15] Y. Li, Y. Wu, J. Liu, Z. Jiang, Z. Chen, G. Yu, and M. R. Lyu, "COCA: generative root cause analysis for distributed systems with code knowledge," in *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2025, pp. 1346–1358.

[16] Z. Jiang, J. Huang, G. Yu, Z. Chen, Y. Li, R. Zhong, C. Feng, Y. Yang, Z. Yang, and M. R. Lyu, "L4: diagnosing large-scale LLM training failures via automated log analysis," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, FSE Companion 2025, Clarion Hotel Trondheim, Trondheim, Norway, June 23-28, 2025*, L. Montecchi, J. Li, D. Poshyvanyk, and D. Zhang, Eds. ACM, 2025, pp. 51–63.

[17] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin, M. Chintalapati *et al.*, "Onion: identifying incident-indicating logs for cloud systems," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021.

[18] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds., 2019.

[19] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 4739–4745.

[20] B. Yu, J. Yao, Q. Fu, Z. Zhong, H. Xie, Y. Wu, Y. Ma, and P. He, "Deep learning or classical machine learning? an empirical study on log-based anomaly detection," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 35:1–35:13.

[21] P. Notaro, S. Haeri, J. Cardoso, and M. Gerndt, "Logrule: Efficient structured log mining for root cause analysis," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 4, pp. 4231–4243, 2023.

[22] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen *et al.*, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 2024, pp. 674–688.

[23] J. Xu, Q. Zhang, Z. Zhong, S. He, C. Zhang, Q. Lin, D. Pei, P. He, D. Zhang, and Q. Zhang, "Openrca: Can large language models locate the root cause of software failures?" in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[24] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," *IEICE Transactions on Communications*.

[25] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[26] "Prometheus query language," 2025. [Online]. Available: https://prometheus.io/docs/prometheus/latest/querying/basics/

[27] "Structured logging and how to use it," 2025. [Online]. Available: https://www.loggly.com/use-cases/what-is-structured-logging-and-how-to-use-it/

[28] "Structured logging, google observability," 2025. [Online]. Available: https://cloud.google.com/logging/docs/structured-logging

[29] G. Yu, P. Chen, Y. Li, H. Chen, X. Li, and Z. Zheng, "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*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 553–565.

[30] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *CoRR*, vol. abs/2107.05908, 2021.

[31] V. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 492–504.

[32] J. Bogatinovski, S. Nedelkoski, A. Acker, J. Cardoso, and O. Kao, "Qulog: data-driven approach for log instruction quality assessment," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022*, A. Rastogi, R. Tufano, G. Bavota, V. Arnaoudova, and S. Haiduc, Eds. ACM, 2022, pp. 275–286.

[33] J. Jiang, F. Wang, J. Shen, S. Kim *et al.*, "A survey on large language models for code generation," *CoRR*, vol. abs/2406.00515, 2024.

[34] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, and Q. Wang, "Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 2332–2354, 2024.

[35] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Evaluating large language models in class-level code generation," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 81:1–81:13.

[36] P. Di, J. Li, H. Yu, W. Jiang, W. Cai, Y. Cao, C. Chen, D. Chen, H. Chen, L. Chen *et al.*, "Codefuse-13b: A pretrained multi-lingual code large language model," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 418–429.

[37] R. Zhong, Y. Huo, W. Gu, J. Kuang, Z. Jiang, G. Yu, Y. Li, D. Lo, and M. R. Lyu, "Ccisolver: End-to-end detection and repair of method-level code-comment inconsistency," *arXiv preprint arXiv:2506.20558*, 2025.

[38] Z. Bi, Y. Wan, Z. Wang, H. Zhang, B. Guan, F. Lu, Z. Zhang, Y. Sui, H. Jin, and X. Shi, "Iterative refinement of project-level code context for precise code generation with compiler feedback," in *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024.

[39] Y. Peng, A. D. Gotmare, M. R. Lyu, C. Xiong, S. Savarese, and D. Sahoo, "Perfcodegen: Improving performance of LLM generated code with execution feedback," in *IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering, Forge@ICSE 2025, Ottawa, ON, Canada, April 27-28, 2025*. IEEE, 2025, pp. 1–13.

[40] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 130:1–130:37, 2022.

[41] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu, "A large-scale evaluation for log parsing techniques: How far are we?" in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, M. Christakis and M. Pradel, Eds. ACM, 2024, pp. 223–234.

[42] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Lilac: Log parsing using llms with adaptive parsing cache," *Proceedings of the ACM on Software Engineering*, no. FSE, 2024.

[43] J. Huang, Z. Jiang, Z. Chen, and M. Lyu, "No more labelled examples? an unsupervised log parser with llms," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 2406–2429, 2025.

[44] Y. Huo, Y. Li, Y. Su, P. He, Z. Xie, and M. R. Lyu, "Autolog: A log sequence synthesis framework for anomaly detection," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023.

[45] Y. Li, Y. Huo, R. Zhong, Z. Jiang, J. Liu, J. Huang, J. Gu, P. He, and M. R. Lyu, "Go static: Contextualized logging statement generation," *Proceedings of the ACM on Software Engineering*, no. FSE, 2024.

[46] Y. Li, Y. Huo, Z. Jiang, R. Zhong, P. He, Y. Su, L. C. Briand, and M. R. Lyu, "Exploring the effectiveness of llms in automated logging statement generation: An empirical study," *IEEE Transactions on Software Engineering*, 2024.

[47] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Ieee, 2019, pp. 121–130.

[48] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 355–366.

[49] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, I. Altintas and S. Chen, Eds. IEEE, 2017, pp. 33–40.

[50] J. Liu, J. Huang, Y. Huo, Z. Jiang, J. Gu, Z. Chen, C. Feng, M. Yan, and M. R. Lyu, "Scalable and adaptive log-based anomaly detection with expert in the loop," *CoRR*, 2023.

[51] J. Liu, S. He, Z. Chen, L. Li, Y. Kang, X. Zhang, P. He, H. Zhang, Q. Lin, Z. Xu *et al.*, "Incident-aware duplicate ticket aggregation for cloud systems," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 2299–2311.

[52] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *Proceedings of the 2010 USENIX Annual Technical Conference, USENIX ATC 2010, Boston, MA, USA*, P. Barham and T. Roscoe, Eds. USENIX Association, 2010.

[53] Z. Li, J. Shi, and M. van Leeuwen, "Graph neural networks based log anomaly detection and explanation," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 306–307.

[54] S. Zhang, Y. Ji, J. Luan, X. Nie, Z. Chen, M. Ma, Y. Sun, and D. Pei, "End-to-end automl for unsupervised log anomaly detection," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, V. Filkov, B. Ray, and M. Zhou, Eds. ACM, 2024, pp. 1680–1692.

[55] Z. Yu, Q. Ouyang, C. Pei, X. Wang, W. Chen, L. Su, H. Jiang, X. Wang, J. Li, and D. Pei, "Causality enhanced graph representation learning for alert-based root cause analysis," in *24th IEEE International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2024, Philadelphia, PA, USA, May 6-9, 2024*. IEEE, 2024, pp. 77–86.

[56] H. Wang, X. Xia, D. Lo, J. C. Grundy, and X. Wang, "Automatic solution summarization for crash bugs," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 1286–1297.

[57] A. Saha and S. C. H. Hoi, "Mining root cause knowledge from cloud service incident investigations for aiops," in *44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2022, Pittsburgh, PA, USA, May 22-24*, 2022.

[58] S. Kang, G. An, and S. Yoo, "A quantitative and qualitative evaluation of llm-based explainable fault localization," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1424–1446, 2024.

[59] S. Banerjee and A. Lavie, "METEOR: an automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL 2005, Ann Arbor, Michigan, USA, June 29, 2005*, J. Goldstein, A. Lavie, C. Lin, and C. R. Voss, Eds. Association for Computational Linguistics, 2005.

[60] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[61] "Doubao embedding langchain," 2025. [Online]. Available: https://v03.api.js.langchain.com/classes/_langchain_community.embeddings_bytedance_doubao.ByteDanceDoubaoEmbeddings.html

[62] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1285–1298.

[63] X. Zhang, S. Ghosh, C. Bansal, R. Wang, M. Ma, Y. Kang, and S. Rajmohan, "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, FSE 2024, Porto de Galinhas, Brazil*, M. d'Amorim, Ed.

[64] D. Roy, X. Zhang, R. Bhave, C. Bansal, P. H. B. Las-Casas, R. Fonseca, and S. Rajmohan, "Exploring llm-based agents for root cause analysis," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, M. d'Amorim, Ed.

[65] X. Huang, T. Zhang, and W. Zhao, "Logrules: Enhancing log analysis capability of large language models through rules," in *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, L. Chiruzzo, A. Ritter, and L. Wang, Eds. Association for Computational Linguistics, 2025.

[66] "Doubao model pricing," 2025. [Online]. Available: https://www.volcengine.com/pricing?product=ark_bd&tab=1

[67] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang *et al.*, "Practical root cause localization for microservice systems via trace analysis," in *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*.

[68] R. R. Sambasivan, A. X. Zheng, M. D. Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger, "Diagnosing performance changes by comparing request flows," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*, D. G. Andersen and S. Ratnasamy, Eds. USENIX Association, 2011.

[69] J. Zhou, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu, "A data set for user request trace-oriented monitoring and its applications," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 699–712, 2018.

[70] A. Zeng, M. Liu, R. Lu, B. Wang, X. Liu, Y. Dong, and J. Tang, "Agenttuning: Enabling generalized agent abilities for llms," in *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds.

[71] S. Parashar, B. Olson, S. Khurana, E. Li, H. Ling, J. Caverlee, and S. Ji, "Inference-time computations for LLM reasoning and planning: A benchmark and insights," *CoRR*, vol. abs/2502.12521, 2025.

[72] E. Chuah, S.-h. Kuo, P. Hiew, W.-C. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne, "Diagnosing the root-causes of failures from cluster log files," in *2010 International Conference on High Performance Computing*, 2010, pp. 1–10.

[73] X. Han, S. Yuan, and M. Trabelsi, "Loggpt: Log anomaly detection via gpt," 2023. [Online]. Available: https://arxiv.org/abs/2309.14482

[74] L. Ma, W. Yang, B. Xu, S. Jiang, B. Fei, J. Liang, M. Zhou, and Y. Xiao, "Knowlog: Knowledge enhanced pre-trained language model for log understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024.

[75] P. Jin, S. Zhang, M. Ma, H. Li, Y. Kang, L. Li, Y. Liu, B. Qiao, C. Zhang, P. Zhao, S. He, F. Sarro, Y. Dang, S. Rajmohan, Q. Lin, and D. Zhang, "Assess and summarize: Improve outage understanding with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1657–1668.

[76] L. Zhang, Y. Zhai, T. Jia, C. Duan, S. Yu, J. Gao, B. Ding, Z. Wu, and Y. Li, "Thinkfl: Self-refining failure localization for microservice systems via reinforcement fine-tuning," 2025.