Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.077908

ARTICLE

Graph-Augmented Multi-Agent Robust Root Cause Analysis in AIOps

Haodong Zou[1,*], **Yichen Zhao**[1], **Xin Chen**[1], **Ling Wang**[1], **Jinghang Yu**[1] **and Long Yuan**[2,*]

[1]Information & Telecommunication Branch, State Grid Jiangsu Electric Power Co., Ltd., Nanjing, China
[2]School of Computer Science and Technology, Wuhan University of Technology, Wuhan, China

*Corresponding Authors: Haodong Zou. Email: haodongz88@163.com; Long Yuan. Email: longyuan@whut.edu.cn

Received: 19 December 2025; Accepted: 13 March 2026; Published: 08 May 2026

ABSTRACT: Root cause analysis (RCA), which leverages multi-modal observability data (including metrics, traces, and logs) to identify the fundamental source of system failures, is critical for ensuring the reliability of complex microservice systems. Traditionally, RCA has relied on human engineers to manually correlate these fragmented signals, which is a labor-intensive and error-prone process. Although recent AIOps advancements, particularly those leveraging Large Language Models (LLMs), aim to automate this workflow, they remain constrained by limitations. Existing methods often rely on single-modal data, restricting diagnostic comprehensiveness. Furthermore, approaches that utilize multi-modal data typically depend on simplistic temporal alignment, which fails to capture complex semantic relationships, or directly employ LLMs, which are prone to hallucinations and lack reliability. To address these issues, we propose a novel Graph-Augmented Multi-Agent Framework that synergizes the structural rigor of graph topology with the advanced semantic reasoning capabilities of LLMs. Our approach operates in two distinct phases designed to mimic human expert problem-solving. First, in the Anomaly Fusion Graph Construction phase, we employ a hybrid alignment strategy to bridge the gap between unstructured logs and structured traces. An LLM serves as a "semantic arbitrator" to resolve ambiguities in high-concurrency scenarios, creating a unified knowledge environment where each node is enriched with comprehensive health insights. Second, the Multi-Agent Collaborative Reasoning phase deploys a team of specialized agents to simulate human Site Reliability Engineering (SRE) workflows. A *Navigator Agent* efficiently guides the search space via calculated fault gradients, while a *Diagnoser Agent* performs deep semantic analysis. Crucially, a *Verifier Agent* enforces an Adversarial Validation Protocol to mitigate hallucinations through rigorous counterfactual reasoning. Extensive experiments conducted on five diverse datasets demonstrate the robustness and effectiveness of our approach. The results show that our framework achieves an average F1-score of 88.4%, significantly outperforming state-of-the-art baselines by 4.6%, proving its ability to synthesize multi-modal information into actionable diagnostic insights.

KEYWORDS: Data fusion; anomaly fusion graph; AIOps

1 Introduction

Modern microservice systems have become increasingly complex due to dynamic interactions and evolving runtime environments [1–4]. These systems often consist of hundreds or even thousands of fine-grained, interdependent subsystems, where issues in any one component can easily lead to performance problems at the top level. Therefore, to ensure system reliability, it is crucial to localize the root cause of these issues in a timely manner [5,6]. However, localizing the root cause is challenging due to the intricate dependencies between subsystems [7,8]. Each request typically follows complex invocation chains involving multiple components, such as services, service instances, and hosts, and the interactions between them.

Moreover, the dynamic nature of these interactions, combined with the heterogeneity of system components, makes pinpointing the root cause a highly non-trivial task [9,10]. As illustrated in Fig. 1, the traditional RCA workflow relies heavily on human engineers to manually correlate fragmented signals from multi-modal data, a process that is often labor-intensive and error-prone.

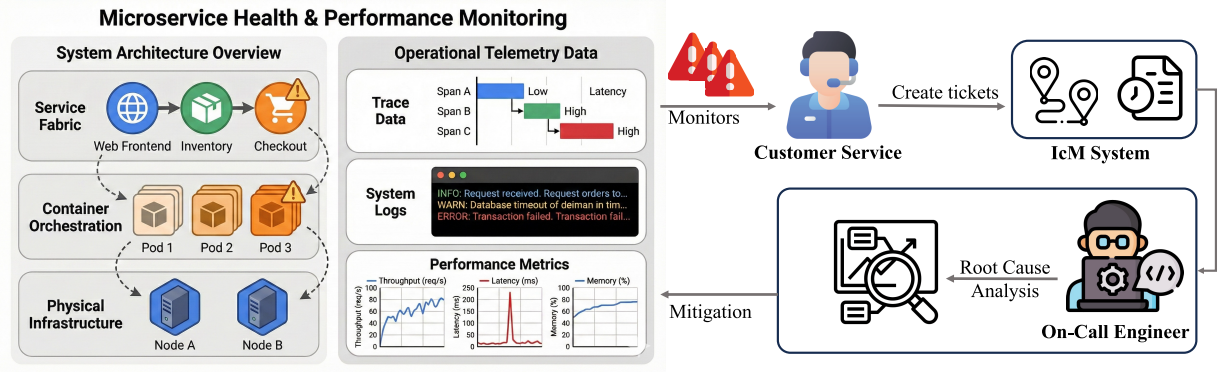

Figure 1: The traditional, human-centric workflow for microservice root cause analysis (RCA). Multi-modal data (Traces, Logs, Metrics) from multiple infrastructure layers (Service, Container, Physical) triggers alerts. These alerts are converted into tickets for on-call engineers, who must manually correlate fragmented signals to localize the root cause.

To mitigate the inefficiencies of manual diagnosis, Artificial Intelligence for IT Operations (AIOps) has emerged as a paradigm shift, aiming to automate anomaly detection and root cause localization. Extensive research has been devoted to automated root cause analysis based on multi-modal data [7,10–13]. These approaches leverage multi-modal signals to capture different aspects of system behavior: *metrics* provide quantitative indicators of system health (e.g., CPU utilization, response time), *traces* reveal the invocation paths and dependencies across services, and *logs* record detailed event sequences and error messages. By integrating these multi-modal data sources, AIOps methods can enable comprehensive diagnosis of complex failures in microservice environments. Recent advancements include graph-based approaches that model service dependencies [14,15], machine learning techniques for anomaly detection [16,17], and causal inference methods that distinguish root causes from symptoms [6,9].

Challenges. Despite these advancements, achieving robust RCA remains challenging, primarily due to two key obstacles:

- **The Intricacy of Multi-Modal Semantic Alignment.** While multi-modal data is available, effectively fusing it remains an open problem due to the significant *semantic gap* between multi-modal signals [18,19]. Metrics provide quantitative trends, traces offer structural dependencies, while logs contain unstructured semantic details. Existing approaches often rely on rigid timestamp matching or shallow feature concatenation, which are insufficient to resolve the ambiguity when mapping unstructured logs to specific trace spans. This misalignment fails to capture the true causal dependencies between system anomalies and error logs, resulting in a fragmented understanding of the failure context [14,20].
- **Cognitive Deficits of LLMs in Structural Reasoning.** Although LLMs demonstrate impressive general knowledge, they fundamentally lack the *spatial reasoning* and *rigorous logic* required for traversing complex service topologies [2]. In scenarios with deep fault propagation chains, unconstrained LLMs often suffer from "causal confusion"—mistaking high-level symptoms for root causes due to their reliance on surface-level textual patterns rather than underlying structural logic. Furthermore, without

a mechanism for self-correction or adversarial verification, LLMs are prone to confident hallucinations, generating plausible but non-existent failure explanations that are unacceptable in critical IT operations [5,8].

Our Approach. To address these challenges, we propose a Graph-Augmented Multi-Agent Framework for automated Root Cause Analysis. Unlike black-box models that directly predict root causes from raw data, our approach simulates the cognitive workflow of a human SRE team, combining the structural rigor of graph topology with the semantic reasoning of LLMs. The framework operates in two phases:

- **Anomaly Fusion Graph Construction.** To overcome the semantic gap between multi-modal data sources, we construct the Anomaly Fusion Graph (AFG) as a unified knowledge environment. This phase employs a novel multi-modal alignment strategy that integrates semantic similarity with spatio-temporal consistency to precisely map unstructured error logs to their corresponding trace spans. An LLM serves as a "semantic arbitrator" to resolve ambiguities in high-concurrency scenarios, ensuring accurate data association. The resulting graph transforms fragmented telemetry into a navigable topology, where each node is enriched with an LLM-synthesized health report that encapsulates both statistical anomalies and semantic failure contexts.
- **Multi-Agent Collaborative Reasoning.** We model the root cause localization as a sequential decision-making process executed by a team of specialized agents. The *Diagnoser Agent* performs deep-dive analysis on local nodes, utilizing Chain-of-Thought reasoning to differentiate between root causes and propagated symptoms. The *Navigator Agent* functions as a topology scout, calculating a "fault gradient" based on latency, error rates, and traffic volatility to prioritize the search path. To mitigate LLM hallucinations, the *Verifier Agent* adopts an Adversarial Validation Protocol, challenging hypotheses through consistency checks and counterfactual reasoning. Orchestrating this collaboration is the *Coordinator Agent*, which manages a global shared context and implements a stack-based backtracking mechanism to dynamically prune incorrect branches and ensure robust convergence.

Contributions. The main contributions of this paper are summarized as follows:

- We propose a novel Graph-Augmented Multi-Agent Framework that integrates the structural rigor of graph topology with the semantic reasoning of LLMs for robust RCA.
- We introduce an Adversarial Verification Mechanism and Dynamic Backtracking Strategy, significantly reducing LLM hallucinations and preventing the search from getting trapped in local optima.
- We construct a high-quality Anomaly Fusion Graph via a hybrid Log-Trace alignment technique, effectively solving the data fragmentation problem in high-concurrency scenarios.
- Extensive experiments demonstrate that our framework achieves superior accuracy and robustness compared to state-of-the-art baselines.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the preliminaries and formalizes the RCA problem. Section 4 details the AFG construction and Multi-Agent reasoning framework. Section 5 presents the experimental evaluation. Finally, Section 6 concludes the paper.

2 Related Work

2.1 Multi-Modal Data Fusion and Graph Construction

In microservice observability, the integration of multi-modal data—metrics, logs, and traces—is fundamental for constructing a comprehensive system view. Early approaches often analyzed these modalities in isolation, which limits the ability to capture complex fault propagation across service boundaries. To address this, recent research has increasingly focused on graph-based representations that fuse structural and behavioral data.

DeepTraLog [7] and LMGD [11] represent a class of methods that construct dependency graphs from distributed traces and log sequences, employing Graph Neural Networks (GNNs) to detect abnormal execution patterns. Twin-Graph [14] advances this paradigm by introducing attention mechanisms to weigh the importance of different observability sources within the graph. Similarly, Wang et al. [10] explore hybrid graph architectures augmented with multi-modal data to enhance detection accuracy. However, a common limitation of these approaches is their reliance on implicit fusion. Data sources are often concatenated into feature vectors for black-box deep learning models, lacking an explicit, navigable topology that preserves the semantic relationships between unstructured logs and structured traces. In contrast, our Anomaly Fusion Graph (AFG) explicitly aligns logs with trace spans using LLM Arbitration, creating a transparent knowledge environment that supports interpretable reasoning.

2.2 Automated Reasoning Strategies

Once the system data is modeled, the core challenge of Root Cause Analysis (RCA) becomes the reasoning strategy used to pinpoint the fault origin. Traditional AIOps methods largely rely on statistical correlation and causal inference. Approaches like Chain-of-Event [8] and MULAN [9] automatically derive event graphs or causal graphs to infer fault propagation paths. Yu et al. [21] employ sophisticated causal learning to localize root causes in multi-modal data. Han et al. [5] and Chen et al. [6] further advocate for holistic analysis, applying causal inference across the complete observability spectrum.

While these methods excel at processing numerical metrics and structured events, they often struggle with the semantic gap inherent in unstructured data. They can identify where a correlation break occurs but often fail to explain why, as they cannot comprehend the rich semantic context provided by error logs. Furthermore, they typically lack the interactive reasoning capabilities of human operators. Our work bridges this gap by introducing a Multi-Agent Framework. Unlike static causal models, our agents simulate the dynamic cognitive processes of SREs, employing Chain-of-Thought reasoning, hypothesis verification, and backtracking, to provide not just a localization result, but a semantically grounded diagnosis.

3 Preliminaries

In this section, we formally define the multi-modal data generated by microservice systems, including metrics, logs, and traces. We then introduce the construction of the Anomaly Fusion Graph (AFG) and formally state the root cause localization problem.

We consider a microservice system $\mathcal{S}$ consisting of N services (or components), denoted as $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$. The system is monitored continuously over a time window T. Microservice observability relies on three primary data modalities: metrics, logs, and traces. Metrics provide quantitative time-series data reflecting resource usage and performance (e.g., CPU, latency). Logs capture discrete, unstructured events recording runtime behaviors. Traces track the propagation of requests across distributed components. We formally define these data types as follows:

Definition 1 (Metric Data): *Let $\mathcal{M} = \{\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(N)}\}$ be the set of metric data for all services. For each service v_i, $\boldsymbol{X}^{(i)} \in \mathbb{R}^{T \times D}$ represents a multivariate time series matrix, where T is the number of time steps and D is the number of metric dimensions. We denote $\boldsymbol{x}_t^{(i)} \in \mathbb{R}^D$ as the metric vector of service v_i at time t.*

Definition 2 (Log Data): *Let $\mathcal{L} = \{l_1, l_2, \ldots, l_M\}$ be the set of raw log entries collected during the time window. Each log entry is defined as a tuple:*

$$l_k = (t_k, v_k, c_k) \tag{1}$$

where t_k is the timestamp, $v_k \in \mathcal{V}$ is the source service generating the log, and c_k is the raw text content (e.g., "Connection timeout" or "DB error"). Distinct from traditional methods that rely on log templates, we treat c_k as semantic text sequences to preserve rich runtime information.

Definition 3 (Trace Data): *A trace τ represents a single request's workflow, consisting of a set of spans $S = \{s_1, s_2, \dots\}$. Each span $s_k \in S$ is defined as a tuple:*

$$s_k = (ID_k, ParentID_k, Service_k, Duration_k, Status_k) \tag{2}$$

where ID_k and $ParentID_k$ identify the calling relationship, $Service_k \in \mathcal{V}$ is the component executing the span, $Duration_k$ is the latency, and $Status_k$ denotes the execution state (e.g., success or error). The collection of all traces is denoted as $\mathcal{T}$.

The aforementioned data modalities provide complementary views of the system status: metrics and logs capture the local health of individual services, while traces reveal the dynamic invocation dependencies between them. However, analyzing these data in isolation is insufficient for effective Root Cause Analysis (RCA), as faults often propagate across services through these dependencies.

Problem Definition. Given the multi-modal observability data $\mathcal{D} = \{\mathcal{M}, \mathcal{L}, \mathcal{T}\}$ collected during an anomaly time window, the **Root Cause Analysis (RCA)** problem is defined as identifying the specific service node $v^* \in \mathcal{V}$ that is the origin of the failure. Formally, we aim to find a mapping $\mathcal{F} : \mathcal{D} \to \mathcal{R}$, where $\mathcal{R}$ is a ranked list of candidate services sorted by their probability of being the root cause. The goal is not only to pinpoint the faulty service v^* but also to identify the failure type (e.g., resource saturation, code exception) and provide interpretable reasoning for the diagnosis.

4 Methodology

We propose a Graph-Augmented Multi-Agent Framework for root cause analysis (RCA) in microservice systems. Unlike traditional pipeline approaches that treat data processing and reasoning as isolated stages, our framework establishes a dynamic symbiosis between a structured knowledge environment—the Anomaly Fusion Graph (AFG)—and a collaborative team of LLM-based Agents. The AFG serves as a comprehensive "world model," consolidating fragmented observability data into a unified semantic topology, while the agents act as autonomous entities that perceive this environment, reason about fault propagation, and iteratively refine their diagnosis.

The framework operates on a Perceive-Reason-Act loop. The architecture is composed of two core layers:

- Anomaly Fusion Graph Construction: This layer transforms raw, multi-modal observability streams (metrics, logs, traces) into a structured graph $\mathcal{G}$. It addresses the challenge of data fragmentation by aligning multi-modal signals and enriching them with semantic context. This graph provides the agents with a navigable representation of the system's health state.
- Multi-Agent Collaborative Reasoning: We employ a team of specialized agents to emulate human SRE roles: the Diagnoser assesses local service states, the Navigator determines the propagation path, the Verifier enforces robust decision-making through multi-perspective consensus, and the Coordinator manages the global shared context and backtracking logic. This collaborative architecture enables the system to autonomously navigate the complex dependency topology, perform root cause analysis, and accurately pinpoint the origin of cascading failures.

4.1 Anomaly Fusion Graph Construction

To facilitate the agents' diagnostic process, we construct the Anomaly Fusion Graph (AFG), denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}_V, \mathcal{A}_E)$. This graph serves as a semantic bridge, projecting discrete and multi-modal anomaly events into a unified dependency topology. As depicted in Fig. 2, the construction pipeline is designed to process raw Abnormal Data through a structured flow. It consists of two primary phases: (1) *Multi-modal Data Alignment*, which fuses Semantic Similarity and Spatio-Temporal Consistency to associate logs with traces, employing an LLM Arbitration mechanism for ambiguous cases to form Fused Anomaly Groups; and (2) *Graph Construction*, which performs Topology Extraction from traces and utilizes an LLM Summarizer to synthesize node attributes, ...ultimately generating the navigable AFG.

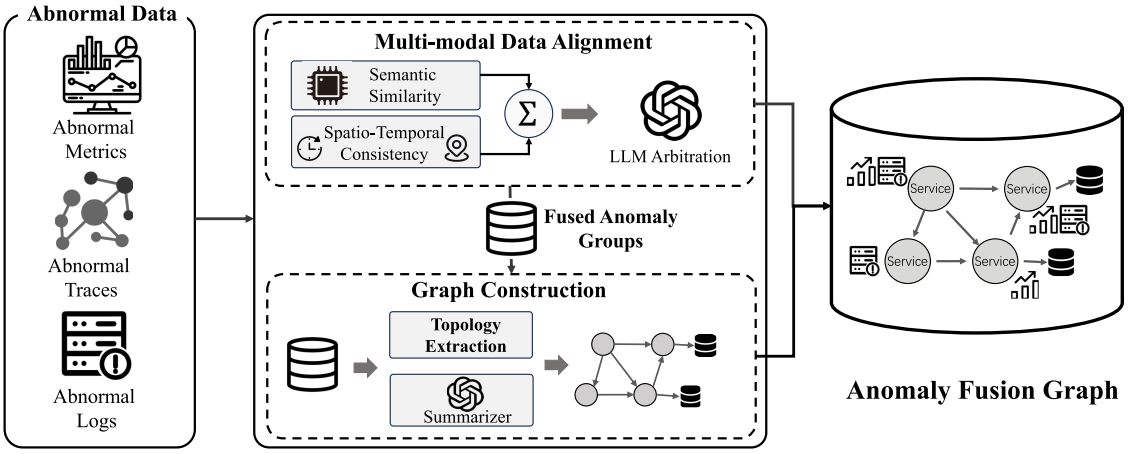

Figure 2: The construction process of the anomaly fusion graph. It illustrates the transformation from raw abnormal data to a structured graph via multi-modal data alignment (integrating semantic similarity and spatio-temporal consistency with LLM arbitration) and graph construction (Topology extraction and LLM-based summarizer).

Definition 4 (Anomaly Fusion Graph): *The Anomaly Fusion Graph is defined as a directed attributed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}_V, \mathcal{A}_E)$. $\mathcal{V}$ *denotes the set of nodes representing microservices and components.* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *represents the set of edges derived from invocation dependencies in traces* $\mathcal{T}$*, where an edge* $(v_i, v_j) \in \mathcal{E}$ *exists if a call from* v_i *to* v_j *is observed.* $\mathcal{A}_V = \{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_N\}$ *is the set of node attributes, where each* $\boldsymbol{h}_i$ *represents a multimodal health state for node* v_i*. Unlike traditional numerical vectors,* $\boldsymbol{h}_i$ *encapsulates an LLM-synthesized health report fusing quantitative metric anomalies with qualitative log semantics:*

$$\mathbf{h}_i = LLM\_Synthesize(MetricFeat(v_i), LogFeat(v_i)) \tag{3}$$

Finally, $\mathcal{A}_E$ *is the set of edge attributes. For each edge* (v_i, v_j)*, the attribute vector contains traffic statistics (e.g., latency, error rate) and interaction logs derived from traces* $\mathcal{T}_{ij}$ *passing through this edge.*

4.1.1 Multi-Modal Data Alignment

The primary objective of data alignment is to overcome the fragmentation inherent in raw observability data, where logs, traces, and metrics lack explicit cross-references. For instance, a log entry often does not natively point to its corresponding trace span, and metric anomalies are not automatically linked to the requests that triggered them. To bridge this semantic gap, we employ a two-stage alignment strategy. This approach aims to fuse scattered signals into unified anomaly events, thereby providing a complete

context for the subsequent graph construction and agent reasoning. The strategy progressively narrows down associations from coarse spatial-temporal grouping to fine-grained semantic matching.

Proximity Evaluation. To efficiently handle high volumes of telemetry data, we first perform a coarse-grained grouping based on service locality and temporal proximity. Let $\mathcal{A} = \{a_1, a_2, \dots\}$ denote the set of all detected anomaly events. Each event $a_k \in \mathcal{A}$ corresponds to an anomaly instance derived from the raw data: it can be a metric anomaly detected in $\mathcal{M}$, an error log $l \in \mathcal{L}$, or an abnormal span s from a trace in $\mathcal{T}$. This step acts as a pre-filter, clustering anomalies that likely belong to the same incident. As shown in the alignment module, we extract time and service identifiers from each anomaly event. Two events $a_i, a_j \in \mathcal{A}$ are merged into a unified group if their composite proximity score $\mathcal{P}_{st}(a_i, a_j)$ exceeds a threshold τ_{prox}:

$$\mathcal{P}_{st}(a_i, a_j) = w_s \cdot \mathbb{I}(a_i.\text{svc} = a_j.\text{svc}) + w_t \cdot \exp\left(-\frac{\Delta t_{ij}^2}{2\sigma_t^2}\right) \tag{4}$$

where Δt_{ij} is the temporal distance and σ_t controls sensitivity.

Hybrid Score Calculation & LLM Arbitration. While proximity grouping is effective for aggregating continuous metric signals, it is insufficient for discrete log events. In high-concurrency environments, multiple spans often execute simultaneously on the same service, as shown in Fig. 3. A simple temporal overlap could erroneously link a failure log to an unrelated successful span, thereby corrupting the causal evidence required for diagnosis. To resolve this *log-span association problem* and map logs to their exact execution context, we calculate a hybrid association score $\mathcal{P}_{assoc}(l, s)$ for each log entry l and candidate span s. This score fuses semantic similarity (via embedding vectors $\mathbf{z}_l, \mathbf{z}_s$) with spatiotemporal consistency:

$$\mathcal{P}_{assoc}(l, s) = \alpha \cdot \frac{\mathbf{z}_l^\top \mathbf{z}_s}{\|\mathbf{z}_l\| \|\mathbf{z}_s\|} + (1 - \alpha) \cdot \left(\beta e^{-\frac{\Delta t^2}{2\sigma_t^2}} + (1 - \beta)\mathbb{I}(l.\text{svc} = s.\text{svc})\right) \tag{5}$$

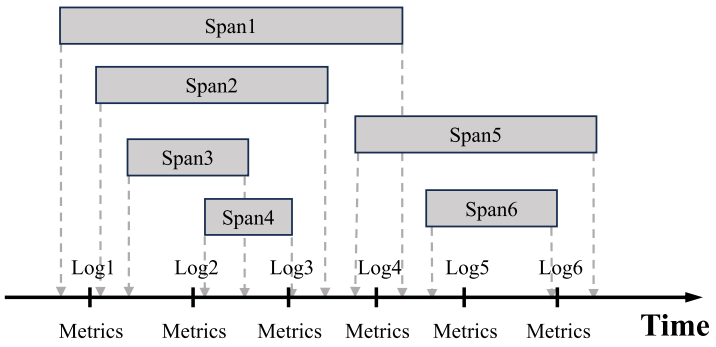

Figure 3: The challenge of log-trace alignment in high-concurrency scenarios. Multiple spans (e.g., Span 2, 3, 4) execute concurrently on the same service. A simple temporal match is ambiguous; for instance, Log 4 could temporally belong to either Span 1 or Span 5, necessitating semantic analysis for precise association.

Matches with scores exceeding a high-confidence threshold τ_{high} are automatically accepted. However, statistical methods may struggle with ambiguous cases where multiple spans exhibit similar characteristics. For such cases (where $\tau_{low} \leq p_{max} < \tau_{high}$), we introduce an **LLM Arbitration** module (detailed prompts are provided in the Appendix A). The LLM acts as a semantic judge, leveraging its contextual understanding to select the best match from candidates, thus ensuring high-precision alignment even for obscure or generic log messages.

Algorithm 1 details the execution flow of this multi-modal alignment strategy. The process begins with *Anomaly Extraction and Coarse-grained Grouping* (Lines 1–14). The algorithm first identifies anomaly instances from raw metrics, logs, and traces to form a unified anomaly set $\mathcal{A}$. These events are then clustered into fused groups $\mathcal{F}$ based on the spatiotemporal proximity score $\mathcal{P}_{st}$, effectively isolating related events into manageable contexts. Subsequently, the algorithm performs *Fine-grained Log-Span Association* (Lines 15–29). Unlike global search methods, this process iterates through each group G_k. For every error log l within the group, it evaluates candidate spans $\mathcal{T}_k$ restricted to the same group. A hybrid association score $\mathcal{P}_{assoc}$ is calculated for each candidate. High-confidence matches ($p_{max} \geq \tau_{high}$) are accepted immediately, while ambiguous cases are queued. Finally, the system executes *LLM Arbitration* (Lines 30–34). For the queued ambiguous pairs, the algorithm constructs a context-rich query $\mathcal{K}$ and leverages the LLM to infer the optimal span s^*, ensuring precise alignment even for generic log messages.

Algorithm 1: Multi-modal data alignment

Input: Metric Data $\mathcal{M}$, Log Data $\mathcal{L}$, Trace Data $\mathcal{T}$, Thresholds $\tau_{prox}, \tau_{high}, \tau_{low}$
Output: Fused Groups $\mathcal{F}$, Log-Span Association $\mathcal{M}_{align}$
1: $\mathcal{A}_{met} \leftarrow \{x \in \mathcal{M} \mid \text{IsAnomaly}(x)\}$; $\mathcal{A}_{log} \leftarrow \{l \in \mathcal{L} \mid \text{IsError}(l)\}$
2: $\mathcal{A}_{trc} \leftarrow \{s \in \text{Spans}(\mathcal{T}) \mid \text{IsAbnormal}(s)\}$; $\mathcal{A} \leftarrow \mathcal{A}_{met} \cup \mathcal{A}_{log} \cup \mathcal{A}_{trc}$
3: $\mathcal{F} \leftarrow \varnothing$
4: **for** each event $a_i \in \mathcal{A}$ **do**
5: $matched \leftarrow$ False
6: **for** each group $G_k \in \mathcal{F}$ **do**
7: Let a_j be the representative of G_k
8: $p_{score} \leftarrow w_s \cdot \mathbb{I}(a_i.\text{svc} = a_j.\text{svc}) + w_t \cdot \exp\left(-\frac{(a_i.t - a_j.t)^2}{2\sigma_t^2}\right)$
9: **if** $p_{score} \geq \tau_{prox}$ **then**
10: $G_k \leftarrow G_k \cup \{a_i\}$; $matched \leftarrow$ True; **break**
11: **if not** *matched* **then**
12: $\mathcal{F} \leftarrow \mathcal{F} \cup \{\{a_i\}\}$
13: $\mathcal{M}_{align} \leftarrow \varnothing$, $\mathcal{Q}_{amb} \leftarrow \varnothing$
14: **for** each group $G_k \in \mathcal{F}$ **do**
15: $\mathcal{L}_k \leftarrow G_k \cap \mathcal{A}_{log}$; $\mathcal{T}_k \leftarrow G_k \cap \mathcal{A}_{trc}$
16: **for** each log $l \in \mathcal{L}_k$ **do**
17: $\mathbf{z}_l \leftarrow \Phi(l.\text{content})$
18: $C_{cand} \leftarrow \mathcal{T}_k$ ▷ Candidates restricted to current group
19: **if** $C_{cand} = \varnothing$ **then continue**
20: $s^*, p_{max} \leftarrow \text{argmax}_{s \in C_{cand}} \mathcal{P}_{assoc}(l, s)$
21: **if** $p_{max} \geq \tau_{high}$ **then**
22: $\mathcal{M}_{align} \leftarrow \mathcal{M}_{align} \cup \{(l, s^*)\}$
23: **else if** $p_{max} \geq \tau_{low}$ **then**
24: $\mathcal{Q}_{amb} \leftarrow \mathcal{Q}_{amb} \cup \{(l, C_{cand})\}$
25: **for** each ambiguous instance $(l, C_{cand}) \in \mathcal{Q}_{amb}$ **do**
26: $\mathcal{K} \leftarrow \text{Contextualize}(l, C_{cand})$ ▷ Formulate query context
27: $s^* \leftarrow \text{LLM}(\mathcal{K})$ ▷ Infer optimal span association
28: $\mathcal{M}_{align} \leftarrow \mathcal{M}_{align} \cup \{(l, s^*)\}$
29: **return** $\mathcal{F}, \mathcal{M}_{align}$

4.1.2 Graph Construction

With the fused anomaly groups established, we synthesize the final graph structure and attributes. Our framework constructs a semantic graph where nodes and edges store explicit natural language descriptions and statistical facts. This design allows LLM-based agents to reason directly over the graph content without information loss.

Topology Extraction. The graph topology is derived directly from trace data $\mathcal{T}$. Nodes $\mathcal{V}$ represent services, and edges $\mathcal{E}$ represent invocation dependencies extracted from parent-child span relationships. This ensures the graph faithfully reflects the runtime interaction paths.

Node Attribute Synthesis. We enrich each node v with a descriptive health report R_v. We aggregate the raw descriptions of all anomalies within the service's fused group $\mathcal{F}_v$ (e.g., metric alerts, error logs) and employ an **LLM Summarizer** to synthesize a concise diagnosis. This summary is stored directly as a text attribute, avoiding the compression loss associated with embeddings:

$$R_v = \text{LLM}\left(\bigcup_{a_i \in \mathcal{F}_v} \text{desc}(a_i) \right) \tag{6}$$

This report serves as the primary context for the Diagnoser Agent to analyze the service's local health state.

Edge Attribute Extraction. For each dependency edge (u, v), we compute explicit statistical metrics and attach relevant log context. The edge attributes $\mathcal{A}_E(u, v)$ include: (1) **Traffic Statistics:** Request count, error rate, and latency derived from traces $\mathcal{T}_{uv}$; and (2) **Interaction Logs:** A collection of error logs linked to the specific calls between u and v (via the mapping $\mathcal{M}$). These attributes provide the Navigator agent with quantitative evidence to assess fault propagation probabilities.

Algorithm 2 formalizes the complete graph construction pipeline. It starts by invoking the Multi-modal Data Alignment (Line 1) to generate fused anomaly groups $\mathcal{F}$ and the log-span mapping $\mathcal{M}_{align}$. Next, it extracts the system topology directly from trace data (Lines 2–6), identifying services $\mathcal{V}$ and dependency edges $\mathcal{E}$ from parent-child span relationships. The algorithm then synthesizes node attributes (Lines 7–10): for each service, it aggregates descriptions from all corresponding anomaly groups and employs an LLM to generate a health report R_v. Finally, edge attributes are computed (Lines 11–15) by calculating traffic statistics from traces $\mathcal{T}_{uv}$ and attaching error logs $\mathcal{L}_{uv}$ linked via the alignment mapping. This results in a fully attributed graph $\mathcal{G}$.

Algorithm 2: Anomaly fusion graph construction

Input: Metric Data $\mathcal{M}$, Log Data $\mathcal{L}$, Trace Data $\mathcal{T}$
Output: Anomaly Fusion Graph $\mathcal{G} = (V, E, \mathcal{A}_V, \mathcal{A}_E)$
1: $\mathcal{F}, \mathcal{M}_{align} \leftarrow$ MultiModalAlignment$(\mathcal{M}, \mathcal{L}, \mathcal{T})$ ▷ Call Algorithm 1
2: $V \leftarrow \{s.\text{svc} \mid s \in \mathcal{T}\}$; $E \leftarrow \varnothing$
3: **for** each trace span $s \in \mathcal{T}$ **do** ▷ Topology Extraction
4: **if** s.parent $\neq$ null **then**
5: $u \leftarrow s$.parent.svc; $v \leftarrow s$.svc
6: $E \leftarrow E \cup \{(u, v)\}$
7: **for** each service node $v \in V$ **do** ▷ Node Attribute Synthesis
8: $G_v \leftarrow \{g \in \mathcal{F} \mid g.\text{svc} = v\}$
9: $D_v \leftarrow \bigcup_{g \in G_v} \bigcup_{a \in g} a.\text{desc}$

(Continued)

Algorithm 2 (continued)

$\mathcal{A}_V[v] \leftarrow \text{LLM}(\text{Summarize}, D_v)$ ▷ Generate health report R_v
for each dependency edge $(u, v) \in E$ **do** ▷ Edge Attribute Extraction
  $\mathcal{T}_{uv} \leftarrow \{s \in \mathcal{T} \mid s.\text{svc} = v \wedge s.\text{parent.svc} = u\}$
  $stats \leftarrow \{\text{Count}(\mathcal{T}_{uv}), \text{ErrRate}(\mathcal{T}_{uv}), \text{Latency}(\mathcal{T}_{uv})\}$
  $\mathcal{L}_{uv} \leftarrow \{l \in \mathcal{L} \mid \exists s \in \mathcal{T}_{uv}, (l, s) \in \mathcal{M}_{align}\}$
  $\mathcal{A}_E[(u, v)] \leftarrow stats \cup \mathcal{L}_{uv}$
return $\mathcal{G}$

4.2 Multi-Agent Collaborative Reasoning

In complex microservice environments, manual root cause analysis typically relies on the expertise of Site Reliability Engineers (SREs). These engineers iteratively traverse service dependencies, analyze local logs, and verify hypotheses. While effective, this human-centric process is labor-intensive and time-consuming, often leading to prolonged Mean Time To Repair (MTTR). To automate this cognitive workflow, we propose a Multi-Agent Collaborative Reasoning framework that simulates human expert decision-making. Building upon the constructed Anomaly Fusion Graph $\mathcal{G}$, we model the root cause analysis as a Sequential Decision-Making Process (SDMP). Unlike black-box models that directly predict root causes from raw data, our system employs an LLM-Guided Heuristic Search strategy. It explicitly navigates the service dependency graph and verifies fault propagation at each step using semantic and statistical evidence. This approach improves localization accuracy while providing transparent, step-by-step explanations of the fault propagation path.

We decompose the reasoning task into four specialized roles, effectively separating state evaluation, path selection, and hypothesis verification, as illustrated in Fig. 4. Detailed prompts are provided in the Appendix A.

Diagnoser Agent (State Evaluator). This agent functions as the termination condition of the search, emulating a clinical analyst who examines a patient's symptoms. Upon visiting a node v, the Diagnoser analyzes the *Node Attribute* R_v (the LLM-synthesized health report from Section 4.1) and correlates it with local error logs $\mathcal{L}_{local}$. Its objective is to determine the node's state: *Root Cause* (failure origin), *Propagator* (victim passing on errors), or *Healthy*. To achieve high precision, the Diagnoser employs a **Chain-of-Thought (CoT)** reasoning process to distinguish intrinsic failures from extrinsic anomalies. It parses the health report for distinct signatures: (1) **Intrinsic Errors:** stack traces, unhandled exceptions, or resource saturation indicating the node is the culprit; (2) **Extrinsic Symptoms:** timeouts, connection refusals, or circuit breaker events suggesting upstream failures. By weighing these cues against local error logs, the Diagnoser filters out noise. Formally, the Diagnoser computes a root cause confidence score $P(Root|v)$ via an LLM inference process Φ_{diag}:

$$P(Root|v) = \Phi_{diag}(R_v, \mathcal{L}_{local}) \tag{7}$$

Note that the score $P(Root|v)$ represents an LLM-estimated confidence value (0–1) based on semantic analysis of the health report and local error logs. If $P(Root|v)$ exceeds a high-confidence threshold τ_{conf}, the agent flags the node as the root cause and triggers the termination of the search.

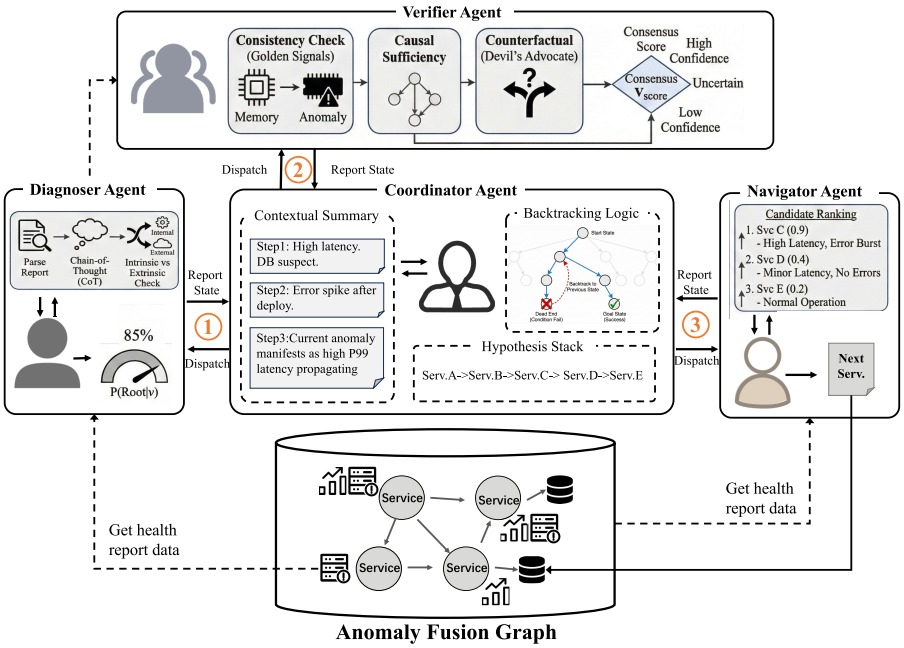

Figure 4: The multi-agent collaborative reasoning architecture. The coordinator serves as the central hub, orchestrating the diagnostic workflow through three key interactions: (1) dispatching the diagnoser for state evaluation, (2) invoking the verifier for hypothesis validation, and (3) directing the navigator for topology traversal. This collaborative loop simulates a human SRE team's decision-making process.

Navigator Agent (Transition Policy). This agent implements the transition policy $\pi(v_{next}|v_{curr})$ for graph traversal, acting as a topology scout that prioritizes suspicious dependencies. When the Diagnoser identifies a node as a "Propagator," the Navigator determines the most likely fault source. It functions as a **probabilistic router**, calculating a transition probability over outgoing edges. The ranking function Ψ_{nav} integrates multi-dimensional metric deviations from *Edge Attributes* $\mathcal{A}_E$: (1) **Latency Deviation:** response time degradation compared to baseline; (2) **Error Propagation Rate:** ratio of failed requests matching downstream error signatures; (3) **Traffic Volatility:** sudden spikes or drops in request volume. By synthesizing these signals, the Navigator constructs a local "fault gradient," guiding the search towards the direction of steepest anomaly intensity. The Navigator selects the next node v_{next} by maximizing a "suspicion score" derived from these edge attributes via a ranking function Ψ_{nav}:

$$v_{next} = \text{argmax}_{v_j \in \mathcal{N}_{out}(v_{curr})} \Psi_{nav}(\mathcal{A}_E(v_{curr}, v_j)) \tag{8}$$

This mechanism ensures that the search path strictly follows the causal chain of fault propagation, avoiding irrelevant branches.

Verifier Agent (Consensus-Based Validator). To mitigate LLM hallucination risks, the Verifier Agent operates under an **Adversarial Validation Protocol**. Unlike the Diagnoser which seeks to *confirm* a hypothesis, the Verifier actively attempts to *falsify* it. It employs a **Multi-Perspective Ensemble** strategy with three cognitive angles: (1) **Consistency Check:** verifying if the hypothesis contradicts any "Golden Signals" (e.g., claiming CPU overload when CPU usage is flat); (2) **Causal Sufficiency:** checking if the root cause sufficiently explains the propagation chain $\mathcal{P}$; (3) **Counterfactual Reasoning:** testing necessity via inverse scenarios ("If Service A were healthy, could Service B still fail?"). Crucially, if the validation fails ($V_{score} < \tau_{verify}$), the Verifier generates a structured **Critique** (e.g., "Mismatch in error logs"). This feedback is fed back

to the Coordinator to prune the search space, preventing the system from revisiting similar erroneous paths. Formally, the Verifier computes a composite validation score V_{score} by aggregating results from K distinct verification prompts $\{\phi_1, \ldots, \phi_K\}$:

$$V_{score} = \frac{1}{K} \sum_{k=1}^{K} w_k \cdot \Phi_{verify}^{(k)}(R_v, \mathcal{P}) \tag{9}$$

The diagnosis is finalized only if the consensus score V_{score} exceeds a strict threshold τ_{verify}. Note that V_{score} is an LLM-estimated confidence score (not a statistical probability) aggregated from multiple verification perspectives. This ensemble approach significantly reduces false positives by requiring agreement across multiple reasoning dimensions.

Coordinator Agent (Memory & Control). Acting as the central orchestrator, the Coordinator manages the *Shared Diagnostic Context* (SDC), a dynamic blackboard where agents exchange insights. This shared memory ensures all agents reason over a consistent global view. The Coordinator maintains a **Global State Board** tracking: a **Visited Set** to prevent cycles, a **Hypothesis Stack** (LIFO) recording decision points for backtracking, and a **Contextual Summary** providing a running narrative of findings (e.g., "Service A is slow, but Service B is healthy").

The Coordinator operates as a **State-Aware Controller**, executing the reasoning loop in Algorithm 3. It implements two key mechanisms. (1) **Dynamic Context Injection:** assembling prompts with the *Contextual Summary*, ensuring agents are aware of previous findings (e.g., preventing the Navigator from suggesting ruled-out paths). (2) **Stack-Based Backtracking:** when hitting a dead end, the Coordinator pops the last branching point from the *Hypothesis Stack* and redirects the search to the next-best candidate, pruning incorrect branches.

The collaboration follows a structured Investigate-and-Walk protocol, mimicking "tracing the error back to its source," as formalized in Algorithm 3. At each step, the Coordinator first invokes the Diagnoser (Step 1) for local diagnosis on v_{curr}. The Diagnoser analyzes the semantic report $R_{v_{curr}}$ for internal failure signs. If suspected as root cause ($P(Root|v_{curr}) \geq \tau_{conf}$), the Verifier Agent (Step 2) performs multi-perspective validation (consistency, sufficiency, counterfactuals). If $V_{score} \geq \tau_{verify}$, the search terminates. If identified as a propagator, the Navigator Agent (Step 3) examines outgoing edges $E_{out}(v_{curr})$, ranks neighbors by statistical anomalies, and transitions to the highest-suspicion neighbor. If no viable candidates exist but the node is not the root cause, the Coordinator initiates backtracking to explore alternative branches.

Algorithm 3: Multi-agent-based heuristic search algorithm

Input: Anomaly Fusion Graph $\mathcal{G}$, Entry Node v_{entry}
Output: Root Cause v^*, Causal Path $\mathcal{P}$
$frontier \leftarrow [v_{entry}]$; $\mathcal{P} \leftarrow [\,]$
while $frontier \neq \varnothing$ **do**
  $v_{curr} \leftarrow frontier$.Pop(); $\mathcal{P}$.Append(v_{curr})
  // Phase 1: State Evaluation
  $report \leftarrow \mathcal{G}.V[v_{curr}]$.Attribute
  $is\_root, confidence \leftarrow$ Diagnoser.Evaluate($report$) ▷ Eq. (7)
  if $is\_root \wedge confidence \geq \tau_{conf}$ **then**
    $is\_valid \leftarrow$ Verifier.ConsensusValidate($report, \mathcal{P}$) ▷ Eq. (9)
    if is_valid **then**
      return $v_{curr}, \mathcal{P}$

(Continued)

Algorithm 3 (continued)

// **Phase 2: Transition Selection**
$candidates \leftarrow$ GetOutgoingEdges$(\mathcal{G}, v_{curr})$
if $candidates = \varnothing$ **then**
  $\mathcal{P}$.Remove(v_{curr}) ▷ Backtrack from dead end
  continue
$ranked\_candidates \leftarrow$ Navigator.RankAll$(candidates)$ ▷ Eq. (8)
for each v_{next} in $ranked\_candidates$ **do**
  $frontier$.Push(v_{next})
return null, $\mathcal{P}$ ▷ No root cause found

Fig. 5 illustrates the agent collaboration through a real failure case from the AIOps dataset. When `order-service` reports timeouts, the workflow proceeds as follows: (1) The *Diagnoser* identifies extrinsic symptoms ("Connection timeout to db-pool") and labels it as a Propagator ($P(Root) = 0.3$). (2) The *Navigator* ranks outgoing edges—`db-proxy` shows 85% error rate and 5× latency spike, while `cache-service` is normal—and selects `db-proxy`. (3) At `db-proxy`, the Diagnoser finds intrinsic errors ("Max connections exceeded, pool exhausted") and flags it as root cause ($P(Root) = 0.92$). (4) The *Verifier* validates through three checks: Consistency (pool saturation matches timeout), Causal Sufficiency (db-proxy failure explains order-service timeouts), and Counterfactual reasoning (healthy db-proxy implies functional order-service). With $V_{score} = 0.94 > \tau_{verify}$, the diagnosis is confirmed. The Hypothesis Stack stores branching points (e.g., `cache-service` as alternative), enabling backtracking if verification fails.

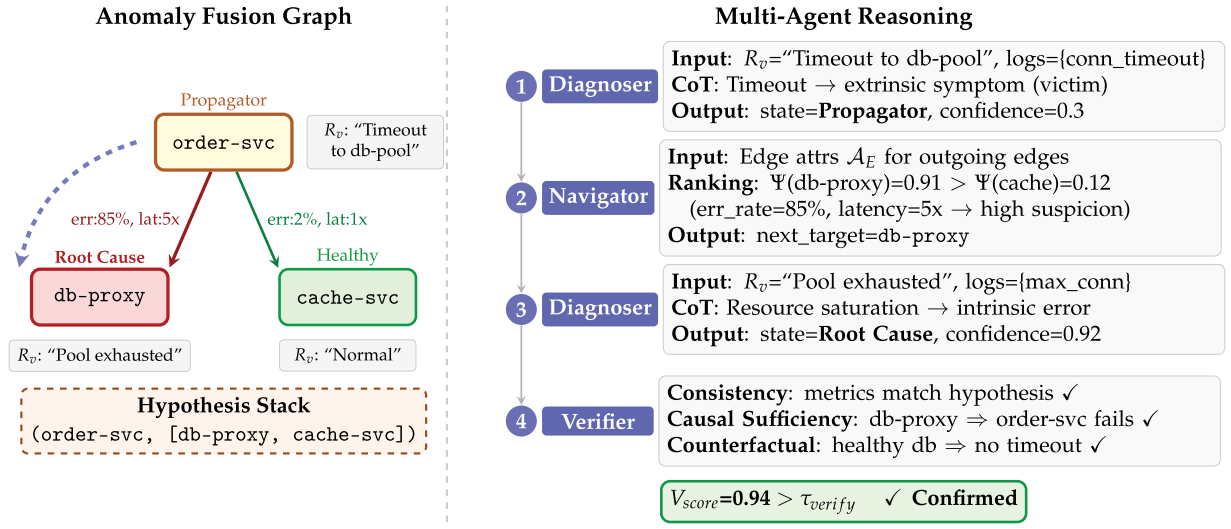

Figure 5: Running example of tracing a database connection failure.

5 Experimental Evaluation

This section presents a comprehensive evaluation of our Graph-Augmented Multi-Agent Framework for root cause localization. We assess the framework's ability to accurately pinpoint fault origins in complex microservice environments, validating the effectiveness of the Anomaly Fusion Graph and the collaborative reasoning mechanism. All experiments were conducted on a server equipped with an Intel Xeon Gold 6248R 3.0 GHz CPU, 512 GB of RAM, and four NVIDIA A100 GPUs.

5.1 Experimental Setup

Datasets. We evaluate our framework on five complementary datasets, shown in Table 1, covering different scales and scenarios of microservice and distributed systems. The first dataset is from the widely-used Train Ticket benchmark [1,7,22], utilizing its V0.2.0 release with 45 services to generate a dataset with 73 faults across 14 cases, resulting in over 132,000 traces and 7.7 million log messages. The second dataset utilizes the Online Boutique system [21], into which they inject 56 faults (42 resource issues and 14 code defects) to obtain traces, logs, and metrics. The third dataset is MSDS [23], a multi-modal dataset collected from an OpenStack distributed system with over 80 fault instances, containing metrics from 5 physical nodes (7 metrics per node including CPU and RAM usage), distributed logs with 23 features, and traces with complete span information. The fourth dataset is from the AIOps Challenge 2025 [24], which provides a comprehensive collection of observability data from a microservice system with 400 fault instances. It includes fine-grained infrastructure metrics (e.g., CPU, memory, disk I/O) from Prometheus, application performance monitoring (APM) metrics from DeepFlow, distributed traces from Jaeger, and container logs from Filebeat. The fifth dataset is self-collected from the OpenTelemetry Demo [25], which contains 45 fault scenarios (CPU contention, network delays, service crashes) injected over 7 days with 5-min durations each.

Table 1: Datasets.

Dataset	Fault Instances	Data Sources
TT	73	Logs, Traces
OB	56	Metrics, Logs, Traces
MSDS	80+	Metrics, Logs, Traces
AIOps	400	Metrics, Logs, Traces
OTel	45	Metrics, Logs, Traces

Baselines. To rigorously evaluate the contribution of our proposed framework, we compare it against two representative baseline categories. The first is DeepTraLog [7], a state-of-the-art graph-based root cause analysis framework. It constructs a service dependency graph and utilizes Graph Neural Networks (GNNs) to fuse traces and logs for anomaly detection and localization. This baseline represents the current advanced deep learning approach in the field. The second baseline is Direct LLM Diagnosis. This baseline serves as a naive upper bound for unstructured LLM usage, demonstrating the limitations of direct LLM application (context window constraints, noise sensitivity, hallucinations) and highlighting the necessity of our AFG construction and multi-agent framework. We note that while recent causal-graph methods (e.g., Nezha [21]) exist, they focus on statistical causal discovery from numerical metrics, whereas our framework emphasizes LLM-driven semantic reasoning over unstructured content. These represent fundamentally different technical approaches. Our baseline selection emphasizes the contrast between graph-based deep learning (DeepTraLog) and LLM-based semantic reasoning approaches.

Evaluation Metrics. We employ standard information retrieval metrics to evaluate the accuracy of root cause localization. For each fault instance, we compare the root cause service identified by the framework against the ground truth. Specifically, we calculate Precision as the proportion of correctly identified root causes out of all diagnoses produced by the system ($\frac{TP}{TP+FP}$), Recall as the proportion of ground truth root causes that were correctly identified by the system ($\frac{TP}{TP+FN}$), and F1-Score as the harmonic mean of Precision and Recall ($\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$).

Implementation Details. We implement our framework in Python 3.9. The Anomaly Fusion Graph is constructed using Drain3 for log parsing and OpenTelemetry for trace processing. For the agent backend,

we utilize the OpenAI GPT-4 API. The default thresholds (τ_{prox} = 0.6, τ_{conf} = 0.85, τ_{verify} = 0.90) were determined through grid search on validation sets. We split each dataset into 80% training and 20% validation sets, then evaluated candidate values for each threshold ($\tau_{prox} \in \{0.3, 0.5, 0.6, 0.7, 0.9\}$, $\tau_{conf} \in \{0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$, $\tau_{verify} \in \{0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$) and selected the combination that maximizes F1-score on the validation set. These values represent a balanced trade-off between precision and recall, validated across all five datasets. The maximum search depth for the Navigator is limited to 10 steps to prevent infinite loops.

5.2 Experimental Results

5.2.1 Root Cause Localization Performance

We compare the performance of our Graph-Augmented Multi-Agent Framework against the baselines across all five datasets. The results are summarized in Table 2.

Table 2: Root cause localization performance (Precision, Recall, F1-Score %). Values show mean ± std from 5 runs.

Method	Metric	TT	OB	MSDS	AIOps	OTel	Avg.
DeepTraLog	Precision	84.1 ± 1.3	86.2 ± 1.1	82.5 ± 1.5	85.1 ± 1.2	83.4 ± 1.4	84.3 ± 1.3
	Recall	83.2 ± 1.4	85.1 ± 1.2	81.8 ± 1.5	84.2 ± 1.3	82.9 ± 1.4	83.4 ± 1.4
	F1-Score	83.6 ± 1.3	85.6 ± 1.1	82.1 ± 1.5	84.6 ± 1.2	83.1 ± 1.4	83.8 ± 1.3
Direct LLM Diagnosis	Precision	70.2 ± 2.1	73.1 ± 1.8	67.5 ± 2.3	71.0 ± 2.0	69.4 ± 2.2	70.2 ± 2.1
	Recall	72.9 ± 2.3	75.4 ± 1.9	70.4 ± 2.4	73.3 ± 2.1	71.5 ± 2.3	72.7 ± 2.2
	F1-Score	71.5 ± 2.2	74.2 ± 1.8	68.9 ± 2.3	72.1 ± 2.0	70.4 ± 2.2	71.4 ± 2.1
Ours	Precision	**87.5 ± 0.8**	**89.6 ± 0.7**	**85.5 ± 1.0**	**88.1 ± 0.9**	**86.8 ± 0.9**	**87.5 ± 0.9**
	Recall	**88.9 ± 0.9**	**91.4 ± 0.8**	**87.3 ± 1.1**	**90.1 ± 1.0**	**88.4 ± 1.0**	**89.2 ± 1.0**
	F1-Score	**88.2 ± 0.8**	**90.5 ± 0.7**	**86.4 ± 1.0**	**89.1 ± 0.9**	**87.6 ± 0.9**	**88.4 ± 0.9**

Bold values indicate the best performance.

Results Analysis. As shown in Table 2, our framework significantly outperforms both baselines. DeepTraLog achieves an average F1-score of 83.8% (±1.3%), demonstrating effective graph-based deep learning. However, it lags behind our framework (88.4 ± 0.9%) by 4.6 percentage points. Paired t-tests confirm statistical significance ($p < 0.01$). The gap likely stems from DeepTraLog's embedding-based fusion lacking deep semantic reasoning for complex failure patterns. Direct LLM Diagnosis achieves 71.4% (±2.1%), showing the value of semantic understanding. However, it struggles with context window limitations and noise in raw data, often leading to hallucinations. Our framework achieves 88.4% (±0.9%), a 17.0 percentage point improvement. The *Anomaly Fusion Graph* structures data effectively, while *Multi-Agent Reasoning* ensures logical, verifiable diagnostic paths.

5.2.2 Anomaly Fusion Graph Construction Quality

The quality of the Anomaly Fusion Graph is a prerequisite for effective multi-agent reasoning. We evaluate the accuracy of associating unstructured logs with distributed traces, which is the core step in graph construction. We compare our method against two baselines: Temporal Window, which links logs to traces based solely on timestamps, and Semantic-Temporal, which combines time windows with keyword matching.

Analysis. Table 3 presents log-trace association results. Our method achieves 97.5% average F1-Score, significantly outperforming baselines. Temporal Window suffers from low precision (65.9%) due to high concurrency, where multiple requests overlap temporally. Semantic-Temporal improves but fails when logs lack explicit identifiers. Our approach leverages semantic embedding and structural correlation to resolve ambiguities, ensuring high-quality graph construction.

Table 3: Log-trace association quality comparison across different methods.

Method	TT	OB	MSDS	AIOps	OTel	Avg.
			F1-Score (%)			
Temporal Window	68.7	71.2	66.4	70.5	67.8	68.9
Semantic-Temporal	78.5	81.3	76.9	80.7	78.2	79.1
Ours	**97.3**	**98.1**	**96.8**	**97.9**	**97.5**	**97.5**
			Precision (%)			
Temporal Window	65.2	68.4	63.1	67.8	64.9	65.9
Semantic-Temporal	76.3	79.1	74.5	78.4	76.0	76.9
Ours	**96.8**	**97.6**	**96.2**	**97.4**	**97.1**	**97.0**
			Recall (%)			
Temporal Window	72.5	74.3	70.1	73.6	71.0	72.3
Semantic-Temporal	81.0	83.7	79.5	83.2	80.7	81.6
Ours	**97.8**	**98.6**	**97.4**	**98.4**	**97.9**	**98.0**

Bold values indicate the best performance.

5.2.3 Robustness Analysis: Efficacy of Consensus-Based Verification

To evaluate the impact of our *Consensus-based Validator* on system robustness, we compare three verification strategies across the AIOps dataset. This experiment specifically isolates the contribution of the multi-perspective ensemble mechanism against simpler alternatives.

- **No Verification:** The system relies solely on the Diagnoser's initial judgment.
- **Single-Pass Verification:** The Verifier employs a standard, single-prompt validation ("Is this the root cause?").
- **Consensus-Based Verification (Ours):** The Verifier aggregates results from Symptom Consistency, Causal Sufficiency, and Counterfactual Reasoning checks.

Analysis. As shown in Table 4, No Verification suffers from low Precision (79.5%), indicating high false positives where the Diagnoser is misled by local symptoms. Single-Pass Verification improves Precision to 84.2% by filtering obvious errors. Our Consensus-based Verification achieves the highest performance (88.1% Precision, 89.1% F1-Score). The multi-perspective ensemble resolves ambiguous cases by requiring counterfactual and causal consistency checks simultaneously. This demonstrates that the consensus mechanism significantly enhances robustness against hallucinations.

Table 4: Robustness comparison of verification strategies (AIOps Dataset).

Strategy	Precision (%)	Recall (%)	F1-Score (%)
No Verification	79.5	88.4	83.7
Single-Pass Verification	84.2	89.1	86.6
Consensus-Based (Ours)	**88.1**	**90.1**	**89.1**

Bold values indicate the best performance.

5.2.4 Ablation Study: Component Contribution

To quantify the contribution of each core component in our multi-agent framework, we conduct a comprehensive ablation study. We report the average performance across all five datasets to provide a holistic view. The variants are defined as follows:

- **Full Framework:** The complete system with all agents and mechanisms enabled.
- **w/o Verifier Agent:** The secondary validation step is removed; the Diagnoser's decision is considered final.
- **w/o Dynamic Backtracking:** The Coordinator disables the backtracking mechanism; the search terminates if a dead end is reached.
- **w/o Heuristic Navigation:** The Navigator selects the next service node randomly from the dependencies, ignoring the edge attribute-based suspicion ranking.

Analysis. Table 5 demonstrates that every component is essential. (1) **Verifier Agent:** Removal causes the most significant Precision drop (−8.5%), confirming its role in filtering false positives from local symptom noise. (2) **Dynamic Backtracking:** Disabling primarily hurts Recall (−13.6%), as the system cannot recover from wrong turns, missing root causes. (3) **Heuristic Navigation:** Replacing with random walking degrades F1 to 75.3%, highlighting the importance of edge attributes in guiding fault propagation path.

Table 5: Ablation study of component contributions (Average across all datasets).

Variant	Precision (%)	Recall (%)	F1-Score (%)
Full Framework	**87.3**	**89.0**	**88.2**
w/o Verifier Agent	78.8	87.4	82.9
w/o Dynamic Backtracking	83.3	75.4	79.2
w/o Heuristic Navigation	74.1	76.5	75.3

Bold values indicate the best performance.

5.2.5 Runtime and Cost Analysis

We report inference latency and token consumption across all datasets. Table 6 summarizes the results.

The multi-agent reasoning phase dominates the total latency (avg. 18.1 s), involving 3–5 agent calls per fault. The AFG construction phase (avg. 3.3 s) includes data alignment and LLM summarization. Token consumption averages 9.7K tokens per fault, with the AIOps dataset requiring the most (12.3K) due to its larger scale. LLM arbitration is triggered for 13.8%–20.3% of log-trace pairs across datasets, with the highest frequency (20.3%) in the high-concurrency AIOps dataset. While arbitration increases token consumption compared to pure statistical methods, it improves alignment precision by 18–22 percentage points (see Table 3). At GPT-4 Turbo pricing, the average cost per fault is approximately $0.15.

Table 6: Runtime and cost analysis per fault instance.

Dataset	AFG Build (s)	Reasoning (s)	Total (s)	Tokens (K)	LLM Arb. (%)
TT	3.2	18.5	21.7	9.8	15.2
OB	2.8	16.3	19.1	8.5	13.8
MSDS	3.1	17.8	20.9	9.2	16.5
AIOps	4.5	22.1	26.6	12.3	20.3
OTel	2.9	15.9	18.8	8.7	14.6
Avg.	3.3	18.1	21.4	9.7	16.1

5.2.6 Limitations and Failure Case Analysis

We analyze failure cases where our framework produced incorrect diagnoses to understand its limitations. Table 7 summarizes the failure distribution.

Table 7: Failure case analysis across datasets.

Failure Type	TT	OB	MSDS	AIOps	OTel
Ambiguous Propagation	4	3	5	24	2
Insufficient Log Semantics	1	1	3	12	1
Other	1	1	1	4	1
Total Failures	6	5	9	40	4
Failure Rate (%)	8.2	8.9	11.3	10.0	8.9

The primary failure mode is ambiguous propagation chains (59% of failures), where multiple services fail simultaneously with similar symptoms such as cascading timeouts, causing the Diagnoser to misidentify a propagator as the root cause. The second major cause is insufficient log semantics (28%), where generic log messages like "Error occurred" lack discriminative features, leading to inconclusive verification results. Regarding backtracking efficiency, in dense topologies with fan-out greater than 10, backtracking degrades to near-exhaustive search; on average it is triggered 1.8 times per fault, but in 12% of AIOps instances with complex dependencies, it exceeds 5 iterations and increases latency by approximately 40%. For LLM arbitration, which achieves 94.2% accuracy overall, errors mainly occur with domain-specific jargon absent from training data and extremely short time windows (<10 ms) where temporal features become unreliable.

6 Conclusion

In this paper, we proposed a Graph-Augmented Multi-Agent Framework for robust root cause analysis in AIOps. By constructing an Anomaly Fusion Graph through LLM-arbitrated log-trace alignment, we transform scattered telemetry signals into a unified semantic topology. Building upon this graph, a collaborative team of specialized agents—Diagnoser, Navigator, Verifier, and Coordinator—operates via a structured Diagnose-Propagate-Decide protocol, employing Chain-of-Thought reasoning, adversarial verification, and dynamic backtracking to ensure diagnostic robustness. Extensive experiments on multiple datasets demonstrate that our framework achieves superior accuracy compared to baselines.

Acknowledgement: This work is supported by the Science and Technology Project of State Grid Corporation of Jiangsu Electric Power—Research on "Key Technologies Research and Application Service for Large-Scale Heterogeneous Monitoring Data Processing in Information Systems" (No. SGJSXT00XTJS2500275).

Funding Statement: This work is supported by the Science and Technology Project of State Grid Corporation of Jiangsu Electric Power—Research on "Key Technologies Research and Application Service for Large-Scale Heterogeneous Monitoring Data Processing in Information Systems" (No. SGJSXT00XTJS2500275).

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Haodong Zou; methodology, Haodong Zou; software, Haodong Zou and Yichen Zhao; validation, Xin Chen, Ling Wang and Jinghang Yu; formal analysis, Haodong Zou and Long Yuan; investigation, Yichen Zhao and Xin Chen; resources, Jinghang Yu; data curation, Haodong Zou and Yichen Zhao; writing—original draft preparation, Haodong Zou; writing—review and editing, Haodong Zou, Ling Wang and Jinghang Yu; visualization, Yichen Zhao and Xin Chen; supervision, Jinghang Yu; project administration, Jinghang Yu; funding acquisition, Jinghang Yu. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Data are unavailable due to privacy or ethical restrictions.

Ethics Approval: Not applicable.

Conflicts of Interest: Authors Yichen Zhao, Xin Chen, Ling Wang and Jinghang Yu were employed by the company State Grid Jiangsu Electric Power Co., Ltd. The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

RCA	Root Cause Analysis
AIOps	Artificial Intelligence for IT Operations
LLM	Large Language Model
AFG	Anomaly Fusion Graph

Appendix A Prompt Templates

This appendix presents the prompt templates used by each agent in our framework.

LLM Arbitration Prompt Template

```
You are a semantic arbitrator for microservice log-trace
alignment. Given an error log and candidate trace spans, determine
which span most likely corresponds to the log.

Error Log: {log_content}
Timestamp: {log_timestamp}
Service: {log_service}

Candidate Spans:
{span_1}: {span_1_details} (timestamp: {span_1_time})
{span_2}: {span_2_details} (timestamp: {span_2_time})
...

Based on semantic similarity, spatio-temporal consistency (same
service and close time), and error/status consistency, assign a
```

```
suspicion score to each candidate span (0-1) and select the most
likely span.

Output format (JSON only):
{
  "span_id": "<best_span_id>",
  "score": 0.0-1.0,
  "rationale": ["brief reason 1", "brief reason 2"]
}
```

Node Health Summarization Prompt Template

```
Summarize the following anomaly events for service {service_name}
into a concise health report:

Metric Anomalies: {metric_alerts}
Error Logs: {error_logs}
Trace Anomalies: {trace_spans}

Generate a structured report that summarizes the observed
symptoms:
1. Primary symptoms (what is failing/degrading)
2. Resource/utilization anomalies (if any)
3. Error patterns (types, frequency, and representative messages)
4. Trace/latency signals (timeouts, retries, slow spans) if
present

Format: [Service: {service_name}] [Health: Normal/Degraded/Anomalous]
[Summary: {concise_description}]
```

Coordinator Agent Prompt Template

```
You are the Coordinator (Memory & Control). Maintain the
investigation state and decide the next action.

State:
- Current Service: {current_service}
- Current Path: {current_path}
- Visited Services: {visited_services}
- Hypothesis Stack: {stack_state}
- Contextual Summary: {current_summary}

Inputs (optional):
- Diagnoser Output: {diagnoser_output}
- Verifier Output: {verifier_output}
```

```
- Navigator Output: {navigator_output}

Task:
1. Update the contextual summary using the newest agent outputs.
2. Decide the next action:
  - DIAGNOSE: run Diagnoser on current service
  - VERIFY: run Verifier on current root-cause hypothesis
  - NAVIGATE: pick next service from Navigator ranking
  - BACKTRACK: pop hypothesis stack and try next candidate
  - STOP: finalize root cause

Output (JSON only):
{
  "action": "DIAGNOSE|VERIFY|NAVIGATE|BACKTRACK|STOP",
  "next_service": "<service_name_or_null>",
  "update_summary": "<one-sentence summary>",
  "update_stack": "<push/pop/none>",
  "final_root": "<service_name_or_null>"
}
```

Diagnoser Agent Prompt Template

```
You are a root cause diagnosis agent (State Evaluator). Analyze
the service health report and local error logs, then classify the
service state as Root Cause/Propagator/Healthy.

Service: {service_name}
Health Report: {health_report}
Local Error Logs: {local_logs}
Current Path: {current_path}
Contextual Summary: {contextual_summary}

Reasoning steps (Chain-of-Thought): (1) Examine the health report
for intrinsic errors (stack traces, unhandled exceptions, resource
saturation e.g. CPU/Memory) indicating this node is the culprit;
(2) Identify extrinsic symptoms (timeouts, connection refusals,
circuit breaker events) indicating this node is a victim of
upstream failures; (3) Correlate with local error logs and filter
noise; (4) Determine state: Root Cause (origin of failure),
Propagator (victim passing on the error), or Healthy.

Output (JSON only):
{
  "state": "Root Cause|Propagator|Healthy",
  "confidence": 0.0-1.0,
  "reasoning": "step-by-step analysis",
```

```
  "evidence": ["short indicator 1", "short indicator 2"]
}
```

Navigator Agent Prompt Template

```
You are a navigation agent (Transition Policy). You are invoked
when the current node is classified as a Propagator. Based on the
outgoing edges and their attributes, determine the most likely
source of the fault by ranking candidate services—construct a
fault gradient toward the direction of steepest anomaly intensity.

Current Service: {current_service}
Outgoing Edges (traffic statistics: request count, latency, error
rate;
interaction logs per edge):
- {service_1}: Latency = {latency_1}, ErrorRate = {err_rate_1},
Traffic = {traffic_1}, Logs = {relevant_logs_1}
- {service_2}: Latency = {latency_2}, ErrorRate = {err_rate_2},
Traffic = {traffic_2}, Logs = {relevant_logs_2}
...

Contextual Summary: {contextual_summary}
Visited Services: {visited_set}

Rank candidates by suspicion score using: (1) Latency Deviation
(response time degradation vs. baseline); (2) Error Propagation
Rate (ratio of failed requests matching downstream error
signature); (3) Traffic Volatility (spikes or drops in request
volume). Respond with the ranked list of service names (most
suspicious first).
```

Verifier Agent Prompt: Consistency Check

```
Act as an adversarial validator: try to falsify the hypothesis
by checking whether it contradicts the observed "Golden Signals"
(CPU, memory, latency, error rate). If you find any contradiction,
mark as inconsistent.

Hypothesis: Service {service_name} is the root cause.
Reasoning: {diagnoser_reasoning}
Golden Signals: {metric_data}

Example: If CPU usage is normal but the hypothesis claims CPU
overload, mark as inconsistent.

Output: {"consistent": true/false, "contradictions": [...]}
```

Verifier Agent Prompt: Causal Sufficiency

```
Evaluate if the identified root cause is theoretically sufficient
to explain the entire observed failure propagation chain (path P):

Root Cause: {service_name}
Propagation Chain: {path}
Downstream Failures: {downstream_services}

Determine if the root cause can theoretically cause all observed
downstream failures. Consider dependency relationships and error
types.

Output: {"sufficient": true/false, "explanation": "reasoning"}
```

Verifier Agent Prompt: Counterfactual Reasoning

```
Adopt a Devil's Advocate stance: test whether the root cause is
necessary for the observed failures. If the hypothesized root
cause service were healthy, would the downstream failures still
occur?

Hypothesis: Service {service_name} is the root cause.
Downstream Services: {downstream_list}
Error Patterns: {error_patterns}

Answer: If {service_name} were healthy, would {downstream_service_1}
still fail? Why or why not? Repeat for each downstream service.

Output: {"necessary": true/false, "counterfactual_analysis":
"..."}
```

References

1. Zhou X, Peng X, Xie T, Sun J, Ji C, Li W, et al. Fault analysis and debugging of microservice systems: industrial survey, benchmark system, and empirical study. IEEE Trans Softw Eng. 2018;47(2):243–60. doi:10.1109/tse.2018.2887384.
2. Wang T, Qi G. A comprehensive survey on root cause analysis in (micro) services: methodologies, challenges, and trends. arXiv:2408.00803. 2024.
3. Meng L, Shao Y, Yuan L, Lai L, Cheng P, Li X, et al. A survey of distributed graph algorithms on massive graphs. ACM Comput Surv. 2024;57(2):1–39. doi:10.1145/3694966.
4. Zhang S, Xia S, Fan W, Shi B, Xiong X, Zhong Z, et al. Failure diagnosis in microservice systems: a comprehensive survey and analysis. ACM Trans Softw Eng Methodol. 2025;35(1):1–55. doi:10.1145/3715005.
5. Han Y, Du Q, Huang Y, Li P, Shi X, Wu J, et al. Holistic root cause analysis for failures in cloud-native systems through observability data. IEEE Trans Serv Comput. 2024;17(6):3789–802. doi:10.1109/tsc.2024.3478759.
6. Chen S, Long X, Fan J, Jin G. A causal inference-based root cause analysis framework using multi-modal data in large-complex system. Reliab Eng Syst Saf. 2026;265(A):111520.

7. Zhang C, Peng X, Sha C, Zhang K, Fu Z, Wu X, et al. DeeptraLog: trace-log combined microservice anomaly detection through graph-based deep learning. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE); 2022 May 25–27; Pittsburgh, PA, USA. Piscataway, NJ, USA: IEEE; 2022. p. 623–34.
8. Yao Z, Pei C, Chen W, Wang H, Su L, Jiang H, et al. 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. New York, NY, USA: ACM; 2024. p. 50–61.
9. Zheng L, Chen Z, He J, Chen H. MULAN: multi-modal causal structure learning and root cause analysis for microservice systems. In: Proceedings of the ACM Web Conference 2024. New York, NY, USA: ACM; 2024. p. 4107–16.
10. Wang P, Zhang X, Cao Z. Anomaly detection for microservice system via augmented multimodal data and hybrid graph representations. Inf Fusion. 2025;118:103017. doi:10.1016/j.inffus.2025.103017.
11. Liu X, Liu Y, Wei M, Xu P. LMGD: log-metric combined microservice anomaly detection through graph-based deep learning. IEEE Access. 2024;12:186510–9.
12. Zhang C, Dong Z, Peng X, Zhang B, Chen M. Trace-based multi-dimensional root cause localization of performance issues in microservice systems. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. Piscataway, NJ, USA: IEEE; 2024. p. 1–12.
13. Gu S, Rong G, Ren T, Zhang H, Shen H, Yu Y, et al. TrinityRCL: multi-granular and code-level root cause localization using multiple types of telemetry data in microservice systems. IEEE Trans Softw Eng. 2023;49(5):3071–88.
14. Huang J, Yang Y, Yu H, Li J, Zheng X. Twin graph-based anomaly detection via attentive multi-modal learning for microservice system. In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway, NJ, USA: IEEE; 2023. p. 66–78.
15. Pazho AD, Noghre GA, Purkayastha AA, Vempati J, Martin O, Tabkhi H. A survey of graph-based deep learning for anomaly detection in distributed systems. IEEE Trans Knowl Data Eng. 2023;36(1):1–20. doi:10.1109/tkde.2023.3282898.
16. Xu H, Pang G, Wang Y, Wang Y. Deep isolation forest for anomaly detection. IEEE Trans Knowl Data Eng. 2023;35(12):12591–604. doi:10.1109/tkde.2023.3270293.
17. Xiang H, Zhang X, Hu H, Qi L, Dou W, Dras M, et al. OptIForest: optimal isolation forest for anomaly detection. arXiv:2306.12703. 2023.
18. Chalapathy R, Chawla S. Deep learning for anomaly detection: a survey. arXiv:1901.03407. 2019.
19. Kwon D, Kim H, Kim J, Suh SC, Kim I, Kim KJ. A survey of deep learning-based network anomaly detection. Cluster Comput. 2019;22(Suppl 1):949–61. doi:10.1007/s10586-017-1117-8.
20. Lee C, Yang T, Chen Z, Su Y, Lyu MR. Eadro: an end-to-end troubleshooting framework for microservices on multi-source data. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Piscataway, NJ, USA: IEEE; 2023. p. 1750–62.
21. Yu G, Chen P, Li Y, Chen H, Li X, Zheng Z. 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. New York, NY, USA: ACM; 2023. p. 553–65.
22. Zhou X, Peng X, Xie T, Sun J, Xu C, Ji C, et al. Benchmarking microservice systems for software engineering research. In: ICSE '18: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. New York, NY, USA: ACM; 2018. p. 323–4.
23. Soldani J, Brogi A. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: a survey. ACM Comput Surv (CSUR). 2022;55(3):1–39. doi:10.1145/3501297.
24. AIOps Challenge 2025. 2025 [cited 2026 Mar 10]. Available from: https://www.aiops.cn/gitlab/aiops-live-benchmark/aiopschallenge2025.
25. Blanco DG. Practical OpenTelemetry. New York, NY, USA: Apress; 2023.