KRCA: An Efficient Root Cause Analysis System in Hyper-Scale Microservice Systems via Agentic AI

Jiamin Jiang[1,*], Jingfei Feng[1], Yu Luo[1], Qingliang Zhang[1], Yongqian Sun[1,†], Wenwei Gu[1]
Shenglin Zhang[1], Tianyu Cui[2], Yao Wu[2], Jielong Huang[2], Nan Qi[2], Dan Pei[3]
[1] Nankai University, China [2] Kuaishou Technology, China [3] Tsinghua University, China
{jiangjiamin,fengjingfei,luoyu,zhangqingliang}@mail.nankai.edu.cn {sunyongqian,wwgu,zhangsl}@nankai.edu.cn
{cuitianyu, wuyao05,zhangsong08,qinan03}@kuaishou.com peidan@tsinghua.edu.cn

ABSTRACT

Hyper-scale microservice systems have become the standard infrastructure for large-scale Internet companies. These systems consist of numerous loosely coupled microservices that evolve independently through continuous development and deployment. Such complexity makes failures unavoidable, necessitating efficient Root Cause Analysis (RCA) to help Site Reliability Engineers (SREs) quickly localize root cause services and classify failure types. However, existing RCA methods often struggle to adapt to the extreme dynamism and massive scale of these systems. In this paper, we present *KRCA*, an end-to-end RCA system designed for hyper-scale microservice systems. To manage the vast search space, *KRCA* employs a multi-stage pipeline that begins with an API-level drilldown to isolate suspicious services. It then instantiates a skeleton-based causal graph from anomalous metrics to serve as a high-recall structural prior, before utilizing a memory-augmented multi-agent framework to verify causality and generate the final failure report. By combining structured causal constraints with multi-agent reasoning, *KRCA* balances diagnostic accuracy with the efficiency requirements of real-time production use. Experimental results show that *KRCA* achieves AC@1 scores of 0.88 and 0.79 for root cause service localization and failure type classification, outperforming the strongest baseline by at least 31% in absolute gains. *KRCA* has been deployed in Kuaishou's production environment for over six months, reducing the average diagnosis time by 77.3%.

CCS CONCEPTS

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

KEYWORDS

Root Cause Analysis, Causal Discovery, Large Language Model

1 INTRODUCTION

Microservice architectures have become the mainstream application framework for large-scale internet companies due to their inherent advantages, particularly their exceptional scalability [9, 30, 37, 48, 49, 56, 60]. In a microservice architecture, a complex application is decomposed into a set of independently deployable and developmental services. With continuous feature updates and rapid business growth, these architectures steadily burgeon into hyper-scale microservice systems. For example, Kuaishou's hyper-scale microservice system already comprises over 200,000 microservices, ensuring 24×7 availability to deliver entertainment for more than 400 million global users daily. However, due to frequent service updates and massive scale expansion, system failures are inevitable. RCA has thus become a critical technique in modern service operations [12, 19, 43, 45, 50, 57, 61, 63]. Its primary goal is to help SREs rapidly diagnose the origin of a failure, which involves two core tasks: 1) *root cause service localization*—pinpointing the specific service responsible for the failure; and 2) *failure type classification*—determining the type of the failure (*e.g.*, CPU saturation or MySQL slow query). Accurate and efficient RCA can significantly reduce the Mean Time to Recovery (MTTR) and mitigate the impact of failures on end users.

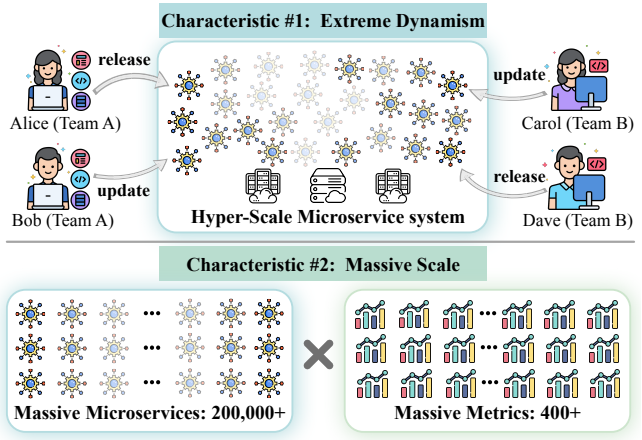


Figure 1: Two primary characteristics of hyper-scale microservice systems: extreme dynamism driven by decentralized, continuous updates across multiple teams, and massive scale involving hundreds of thousands of services monitored by numerous metrics.

Compared to open-source microservice benchmarks widely used in existing RCA literature—such as Google Online Boutique (10 services) [13] and even the largest industrial-scale study to date, which involves up to 30,000 services [22]—performing RCA in hyper-scale microservice systems presents unique challenges characterized by two primary dimensions: extreme dynamism and massive scale, as illustrated in Fig. 1. First, the system is highly dynamic. Microservices are developed and updated by different teams with completely independent release cycles. This decentralized governance means the topology and normal behavior patterns of the hyper-scale system are constantly evolving, demanding an RCA system that operates in real time without relying on pre-trained models. Second,

[*] Work done during an internship at Kuaishou Technology. † Corresponding author.

ASE '26, October 12–16, 2026, Munich, Germany
2026. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the scale of observability data is massive. Hyper-scale microservice systems encompass hundreds of thousands of microservices, each monitored by hundreds of metrics. When a failure occurs, it can quickly propagate through complex dependency chains and induce anomalies in a large number of downstream services and metrics. The combined growth in the number of services and the dimensionality of metrics therefore requires an RCA system with high efficiency.

Existing RCA approaches exhibit fundamental limitations when confronted with these two characteristics. On the one hand, deep learning-based methods [20, 25, 37, 49, 54, 55] can efficiently localize failure once trained. However, they require frequent model retraining to capture the constantly changing system features of a hyper-scale system. In practice, it is difficult to define exactly when retraining is needed, and more importantly, the training process is so time-consuming that a single training cycle often takes longer than the system's variation cycle, rendering these models unusable for real-time deployment. On the other hand, recent Large Language Model (LLM)-based RCA approaches [1, 6, 7, 15, 30, 32, 36, 41, 44, 62] attempt to leverage in-context learning to accurately acquire new system features on the fly. However, in a hyper-scale system, crucial evidence of failure propagation is often buried within prohibitively long contexts composed of massive services and metrics. Consequently, LLMs easily fall into the "Lost in the Middle" dilemma [24], failing to extract the true causal relationships and generating hallucinated diagnoses.

To bridge this gap, we propose *KRCA*[1], an end-to-end RCA system designed to efficiently localize root cause services and classify failure types in hyper-scale microservice systems. Inspired by the principle of hierarchical RCA [5], *KRCA* follows a progressive multi-stage diagnosis paradigm that systematically narrows down the search space. Specifically, the workflow is decomposed into three sequential steps: (1) Identifying the top-N most suspicious services from the massive dependency graph; (2) Constructing an initial causal graph for the anomalous metrics of each suspicious service; (3) Leveraging LLM-based reasoning to refine and complete the causal graph, and ultimately to localize the root cause service and classify the failure type.

To implement this concept into a practical system, three technical challenges must be addressed:

Challenge 1: A single failure case can involve a large number of services, and in extreme cases this number can exceed 10,000 in a hyper-scale system. This massive blast radius of downstream services makes it difficult to distinguish the true root cause service.

Challenge 2: A single service in a hyper-scale system contains hundreds of metrics. Traditional causal discovery algorithms infer causal relationships directly from raw time-series data, overlooking the rich semantic information carried by the metrics. Our empirical study shows that, as the number of metrics increases, the accuracy of these algorithms declines sharply, while the computational cost becomes prohibitive.

Challenge 3: Although LLMs can use the semantic cues encoded in metric names and gather external evidence through tool calls to infer causal relationships, an efficient RCA system still needs stronger reasoning tailored to RCA scenarios. In particular, the LLM must integrate cross-domain knowledge (*e.g.*, CPU and MySQL) to verify causality, while avoiding the high latency caused by multi-turn reasoning.

KRCA addresses these challenges through three core design components. First, to tackle Challenge 1, *KRCA* employs an API-level drilldown strategy. Starting from the alerting Application Programming Interfaces (APIs), it recursively traverses the dependency graph with a robust scoring function, thereby pruning irrelevant downstream services and narrowing the search space to a compact set of suspicious services. Second, to address Challenge 2, *KRCA* utilizes Skeleton-based Causal Graph Instantiation. Instead of relying on time-consuming statistical algorithms, it leverages the semantic information of metrics to map anomalous time series into a generic causal skeleton graph with high recall. Third, to overcome Challenge 3, *KRCA* introduces a multi-agent reasoning framework with Retrieval Augmented Generation (RAG) and tiered memory, enabling specialized agents to collaboratively verify causality and gather evidence, while also accelerating the overall reasoning process through parallel execution, thereby enabling efficient LLM-based reasoning for RCA.

In summary, this paper makes three main contributions: (1) To the best of our knowledge, *KRCA* is the first end-to-end root cause analysis system designed for hyper-scale microservice systems. It can localize the root cause service and classify the failure type within a practical time budget for real-world incident response; (2) We propose a progressive multi-stage RCA framework that addresses several key challenges in hyper-scale systems. *KRCA* first combines API-level drilldown over multiple API-related signals to reduce the search space, structured causal constraints to organize anomalous metrics, and multi-agent reasoning to verify causal relations, forming a complete end-to-end analysis pipeline; (3) We evaluate *KRCA* on 300 real-world failures and deploy it in Kuaishou's production environment for over six months. The experimental results show that *KRCA* achieves AC@1 scores of 0.88 and 0.79 for root cause service localization and failure type classification, respectively, outperforming the strongest baseline by absolute gains of 31% and 32%, respectively. In production, *KRCA* reduces the average time for root cause localization by 77.3%.

2 BACKGROUND AND MOTIVATION

2.1 Background

Hyper-scale microservice systems. Hyper-scale microservice systems consist of hundreds of thousands of loosely coupled services that are independently developed and deployed. This immense scale is driven by fine-grained functional decomposition, as well as the proliferation of service versions introduced by canary releases and multi-region deployments. Through interactions based on Remote Procedure Call (RPC) and HTTP, these services form highly complex dependency graphs. Middle-platform services play a central role in such systems by providing shared capabilities, *e.g.*, traffic scheduling and risk control, to hundreds of upstream and downstream services. As a result, any failure that originates in, or propagates through, middle-platform services can rapidly expand the affected subgraph from dozens of services to thousands, thereby creating a massive blast radius of cascading anomalies.

[1] K is an acronym for the first letters of the Chinese words Kuaisu and Kuaishou, where Kuaisu means fast in Chinese.

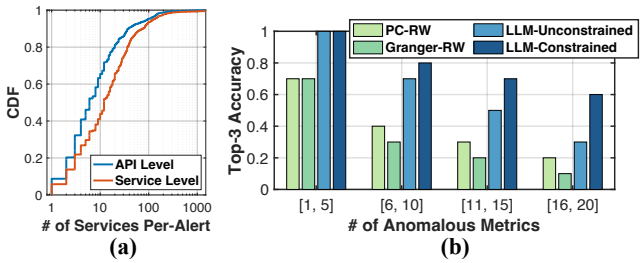


Figure 2: Empirical study on the limitations of existing RCA practices. (a) CDF of the number of services from the alerting service to the root cause service under API-level and service-level dependencies. (b) Top-3 root cause metric localization accuracy under different numbers of anomalous metrics.

2.2 Motivation

To motivate the design of our system, we conducted an empirical study based on production incidents in hyper-scale microservice systems.

Motivation 1: Service-level observability is too coarse for RCA in hyper-scale microservice systems.

In production environments, service-level metrics are often used as the default granularity for RCA [22, 51, 59]. However, this granularity is too coarse for accurate root cause service localization, because it aggregates multiple APIs within a service and therefore obscures the fine-grained failure signals associated with individual invocation paths. On the one hand, service-level diagnosis mixes multiple invocation paths from different APIs within the same service, which causes all downstream services related to that service to be indiscriminately included in the RCA search space. As shown in Fig. 2(a), service-level dependencies consistently involve more candidate services than API-level dependencies. Specifically, 36.3% of alerts involve more than 10 services under API-level dependencies, whereas this proportion increases to 56.7% under service-level dependencies. On the other hand, the service-level success rate is dominated by high-traffic APIs, which can mask severe degradation in low-traffic but faulty APIs [25]. For example, within the same service, API#1 handles 680K Queries Per Second (QPS), whereas API#2 handles only 1.6K QPS. Even when the success rate of API#2 drops sharply to 10%, the aggregated service-level success rate remains above 99% because the traffic volume of API#1 overwhelmingly dominates the overall service-level metric. As a result, the failure of the faulty API remains hidden at the service level.

Design insight 1: Root cause analysis in hyper-scale microservice systems should begin with API-level metrics rather than service-level metrics.

Motivation 2: Causality should not be inferred directly from the raw anomalous metrics set.

After locating root cause service, a common RCA process is to detect anomalous metrics, infer causality among them using causal discovery algorithms, and then apply graph-based ranking methods such as Random Walk (RW) [17] to locate root cause metric. Prior work [5, 21, 29, 58] typically relies on traditional causal discovery algorithms, such as Peter-Clark (PC) [35] and Granger causality [14], to construct causality among metrics in the anomalous metric set. However, production metric data are often high-dimensional (typically more than 10 metrics) and noisy due to sampling distortion and discontinuities, making these time-series-based causal discovery methods unreliable in production environments. As shown in Fig. 2(b), we collected 20 real-world incidents from Kuaishou with varying numbers of anomalous metrics, and applied the PC and Granger algorithms to each case. The results show that the performance of both PC and Granger degrades sharply as the number of metrics increases. When only 5 metrics are involved, both methods achieve an accuracy of around 50%. However, when the number of metrics reaches 20, which is a common scale in hyper-scale systems, the accuracy of both methods falls below 20%. Moreover, as the number of anomalous metrics and the length of the time series increase, the computational cost of traditional causal discovery becomes too high for real-time RCA in hyper-scale systems.

Recent studies have shown that LLMs possess certain causal reasoning capabilities: they can recognize variable semantics and assist causal discovery by incorporating prior knowledge [10, 11, 40]. As shown in Fig. 2(b), we further evaluate LLM-based causal discovery under both unconstrained and constrained settings. When the number of metrics is small, the unconstrained LLM achieves high accuracy, reaching 100%. However, as the number of anomalous metrics increases, its accuracy also drops rapidly to around 30%. This decline occurs because LLMs tend to be overconfident and hallucinate non-existent causal relations when they reason directly over a large set of anomalous metrics. A more robust approach is to impose structured causal constraints on the anomalous metric set using expert knowledge before causal reasoning. For example, directional constraints restrict candidate causal edges to those that are consistent with realistic failure propagation, whereas semantic constraints specify that certain availability-related metrics can only be treated as effects rather than causes. Under these constraints, the LLM remains substantially more robust, and its accuracy stays above 60% even as the number of anomalous metrics increases.

Design insight 2: Anomalous metrics should first be organized into a sparse, high-recall structural prior that constrains the causal search space, after which LLMs can perform targeted, evidence-based reasoning to refine the final causal explanation.

3 SYSTEM DESIGN

3.1 Overview

KRCA is mainly composed of three modules, as shown in Fig. 3. (1) **API-level drilldown module.** Starting from the alerting API, this module recursively traverses the service dependency graph and scores downstream APIs using failure rate and latency. It keeps only suspicious propagation paths, reducing the graph to a compact set of suspicious services; (2) **Causal graph G_o generation module.** For each suspicious service, this module detects anomalous metrics from monitoring data and maps them onto a generic causal skeleton with four meta-metric types. This process yields an initial service-level causal graph G_o, which provides structured prior knowledge for later diagnosis; (3) **Multi-agent collaboration module.** This module further refines G_o and improves diagnostic accuracy through a multi-agent framework in which a Main Agent coordinates multiple Sub Agents to verify causal relations and gather evidence. A memory-augmented retrieval mechanism

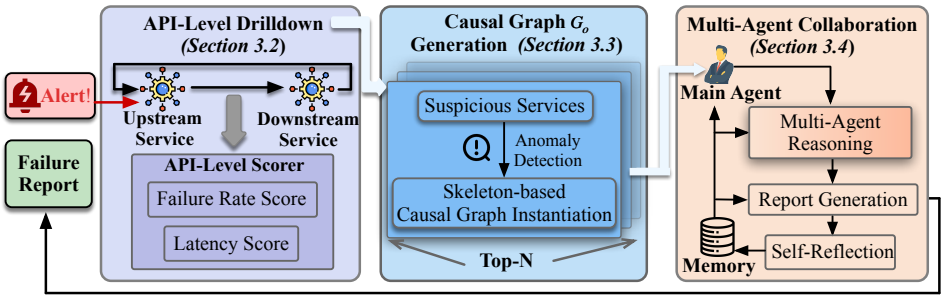


Figure 3: System architecture of *KRCA*. The workflow consists of three modules: (1) API-Level Drilldown to locate the top-N suspicious services from an alert; (2) Causal Graph G_o Generation to construct an initial skeleton-based causal graph for anomalous metrics; and (3) Multi-Agent Collaboration to refine the graph and generate the failure report.

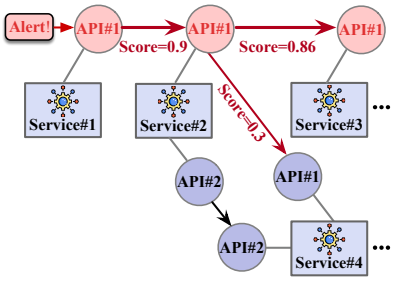


Figure 4: Illustration of the API-level drilldown process. Starting from the alerting API, *KRCA* recursively evaluates and prunes downstream APIs based on a scoring function.

supplies similar historical cases and diagnostic experience. After several rounds of refinement, the final causal graph is used to generate a failure report that identifies the root cause service and failure type.

3.2 API-level drilldown

In our hyper-scale microservices environment, the initial signal of a failure is often an availability alert on a specific service API, triggered when its success rate drops below a predefined threshold. To rapidly pinpoint the cause of this availability drop, *KRCA* must first drilldown from a vast number of downstream dependencies to a small set of suspicious services. The primary challenge lies in selecting accurate and efficient features to guide this drilldown process.

Following established SRE principles [3] and the observations in Section 2.2, we focus drilldown on failure rate and latency at the API level, as they provide the most direct signals of fault propagation: a fault in a downstream service typically manifests as increased errors or latency in its upstream caller. We quantify the likelihood that a failure observed at an upstream API P (Parent) originates from a downstream API C (Child) with the following scoring function:

$$\text{Score}(P, C) = \max(\text{Score}_f(P, C), \text{Score}_l(P, C)) \tag{1}$$

where $\text{Score}_f(P, C)$ is the failure rate score and $\text{Score}_l(P, C)$ is the latency score. If $\text{Score}(P, C)$ exceeds threshold, we regard the downstream service as suspicious and continue the drilldown from that point.

Failure rate score. Failures in a downstream API C often propagate to its upstream caller API P, leading to highly synchronized fluctuation patterns in their failure rate time series. To quantify this temporal dependency, we define the failure rate score as follows:

$$\text{Score}_f(P, C) = \max_{\tau \in [0, L]} \text{Corr}\left(\mathbf{p}[t_s, t_e], \mathbf{c}[t_s - \tau, t_e - \tau]\right) \tag{2}$$

where $\mathbf{p}$ and $\mathbf{c}$ denote the failure rate time series of the upstream API P and downstream API C, respectively. L represents the maximum time lag. The correlation is computed using the *Pearson Correlation Coefficient* [2] over a dynamic observation window $[t_s, t_e]$, where t_e denotes the timestamp at which the alert is triggered. To concentrate on the failure period, we determine the starting point t_s as the inflection point, defined as the latest moment before t_e at which the sign of the first-order difference in the failure rate changes, indicating the qualitative transition from a steady state to a failure state. Finally, we perform a p-value test with $\alpha = 0.05$ to ensure statistical significance; correlations that do not pass this test are set to zero to filter out spurious noise.

Latency score. Unlike the failure rate, latency is influenced by multiple factors (*e.g.*, network jitter and code execution), which leads to a lower signal-to-noise ratio. Consequently, relying solely on correlation often produces inaccurate results. To address this, we define a composite score that integrates anomaly degree, fluctuation contribution, and correlation degree:

$$\text{Score}_l(P, C) = w_1 \cdot \mathcal{A}(C) + w_2 \cdot \mathcal{F}(P, C) + w_3 \cdot \mathcal{C}(P, C) \tag{3}$$

where w_1, w_2, and w_3 are balancing weights. All components are computed over the same dynamic observation window $[t_s, t_e]$ defined in the failure rate. Let $\mathbf{p}$ and $\mathbf{c}$ represent the latency time series of the upstream API P and downstream API C, respectively, and let p_t and c_t denote their values at time t. Firstly, the anomaly degree $\mathcal{A}$ quantifies the average deviation of the downstream API C from its optimal performance baseline within the observation window:

$$\mathcal{A}(C) = \frac{1}{t_e - t_s + 1} \sum_{t=t_s}^{t_e} \frac{|c_t - c_{base,t}|}{c_t} \tag{4}$$

where $c_{base,t}$ denotes the historical minimum latency for the corresponding observation window, calculated across a multi-granularity look-back (1-hour, 1-day, and 1-week). Secondly, the fluctuation contribution $\mathcal{F}$ measures the extent to which the latency instability of the downstream API C explains the fluctuation of its caller P:

$$\mathcal{F}(P, C) = \frac{\text{QPS}_C}{\text{QPS}_P} \cdot \frac{\sum_{t=t_s}^{t_e} |c_t - c_{t-1}|}{\sum_{t=t_s}^{t_e} |p_t - p_{t-1}|} \tag{5}$$

where QPS denotes the Queries Per Second of the corresponding APIs. Thirdly, the correlation degree $\mathcal{C}$ is used to validate trend consistency. It adopts the same time-lagged Pearson Correlation Coefficient defined in Eq. 2.

The overall drilldown process is a recursive traversal of the service dependency graph guided by $\text{Score}(P, C)$. As depicted in Fig. 4, this process begins at the alerting API (Service#1::API#1). The drilldown process first evaluates the direct downstream call to Service#2::API#1. Since the score of 0.9 exceeds the threshold, the process continues down this path. From Service#2::API#1, the process then examines its own downstream dependencies. The

path leading to Service#3::API#1 yields a high score of 0.86, which prompts the process to continue along this branch. Conversely, the path to Service#4::API#1 is pruned because its score of 0.3 falls below the threshold. This indicates that Service#4 is not a major factor in the performance degradation. The drilldown process repeats throughout the dependency graph until no further paths meet the criteria. Through this process, *KRCA* effectively narrows the candidates to a compact set of suspicious APIs. Finally, all identified APIs are ranked based on their calculated scores. The top-N services associated with the highest-scoring APIs are recommended as the most suspicious root causes for the alert.

3.3 Skeleton-based Causal Graph Instantiation

In Section 3.2, *KRCA* has narrowed down the scope to a few suspicious services. However, in hyper-scale systems, a single service is typically monitored by a vast array of metrics—often exceeding 400. These metrics cover physical resources (*e.g.*, CPU and GPU), dependent middleware and database (*e.g.*, Kafka and MySQL) and service availability Key Performance Indicator (KPI). To effectively diagnose an alert, it is crucial to discover the causal relationships among these metrics. A straightforward approach is to apply standard causal discovery algorithms (*e.g.*, PC) to the anomalous metrics. However, as demonstrated in our empirical study (Section 2.2), the accuracy of these algorithms drops sharply as the number of metrics increases, while the computational overhead becomes prohibitive.

Therefore, instead of attempting to recover a complete causal graph directly from raw time-series, the objective of this section is to construct an initial causal graph, denoted as G_o, within each suspicious service. The rationale is to provide structured prior knowledge of the failure for the subsequent multi-agent refinement. Intuitively, this mirrors the conceptual design of Bayesian inference [4], offering a principled and systematic diagnostic pipeline.

Inspired by CIRCA [18] and cloud-native architectures, we propose a generic causal skeleton for individual services, denoted as G_s, which is composed of four types of meta-metrics: External (E), Internal (I), Dependency (D), and KPI (K), as illustrated in Fig. 5(a). The definitions of these meta-metrics and their relationships are described as follows: (1) **External (*E*)**: E represents external factors, such as traffic surges or upstream poison requests. Anomalies in E can directly impair K, induce anomalies in I by increasing processing load, or overload D through high-frequency access; (2) **Internal (*I*)**: I represents host-level or component-specific resources, such as CPU or GPU utilization. Although I directly affects K, anomalies in I may also degrade the performance of D through abnormal behaviors, such as aggressive retries or prolonged connection occupancy; (3) **Dependency (*D*)**: D represents the external dependencies of a service, such as Kafka, MySQL, and Redis. Anomalies in D can directly impair K; (4) **KPI (*K*)**: K represents the key performance indicators of a service, such as latency and success rate.

Next, we use the skeleton graph G_s to build a concrete intra-service initial causal graph G_o. The construction proceeds in two stages. We first detect anomalous metrics using a hybrid strategy to improve robustness across different metric types and services. We use Isolation Forest [23] as the default detector because it does not depend on strong distributional assumptions and is more reliable

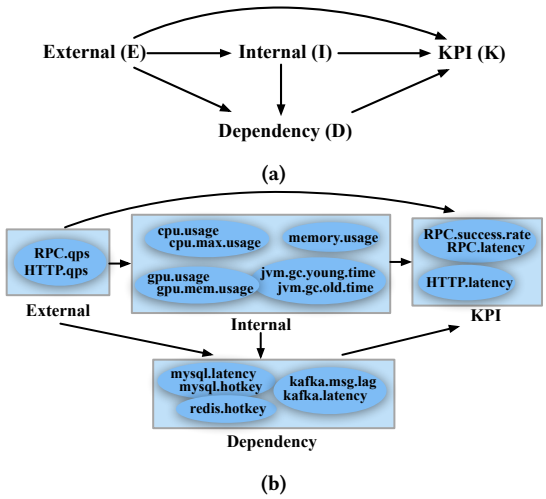


Figure 5: Skeleton-based causal graph instantiation. (a) The generic causal skeleton graph G_s defining the predefined causal relationships among four meta-metric types. (b) An instantiated causal graph example where anomalous metrics (*e.g.*, cpu.usage, mysql.latency) are mapped to their corresponding meta-metric types based on the skeleton structure.

for heterogeneous metrics collected from different services. In practice, however, we found that Isolation Forest can miss some shape-related anomalies, such as sustained shifts and plateau-like degradations. To address this issue, we combine it with PatternMatcher [47], which is better at capturing temporal anomaly patterns that are important in production. We then map each detected anomalous metric to its meta-metric type and instantiate causal edges among these meta-metrics according to the predefined structure of G_s. Fig. 5(b) shows an example. A sudden traffic surge (E) increases internal pressure (I), which appears in metrics such as cpu.usage and gpu.usage, and this change further affects RPC.latency and RPC.success.rate (K). Meanwhile, the increase in internal pressure also triggers anomalies in dependencies (D), such as mysql, redis, and kafka.

However, the causal graph G_o instantiated from the generic skeleton G_s remains insufficient to capture the full complexity of failure propagation within a service. For example, in Fig. 5(b), the anomalies of mysql.hotkey and redis.hotkey are caused by the traffic surge rather than by internal components of the service, such as GPU. Furthermore, causal relations may exist within a single meta-metric type, which cannot be represented by G_s. To address this, the next section introduces a multi-agent system to validate the causal relations in G_o and discover the finer-grained causal relations among metrics within the same meta-metric type, thereby completing the causal graph.

3.4 Multi-Agent Collaboration

Following the process described in the previous section, *KRCA* identifies a set of suspicious services and constructs an initial causal graph G_o for each service. However, these graphs often contain redundant causal edges and fail to capture finer-grained causal relations across meta-metrics. The final challenge is therefore to refine

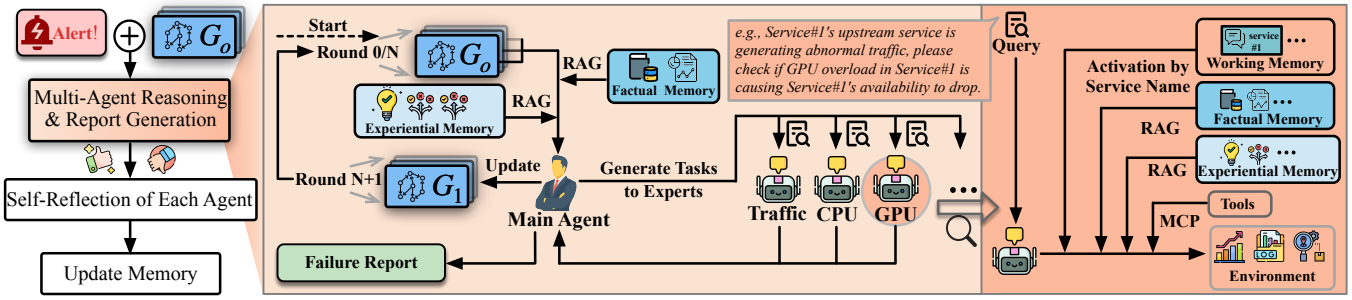


Figure 6: Overview of the Multi-Agent Collaboration framework. The Main Agent orchestrates domain-specific Sub Agents (*e.g.*, Traffic, CPU, GPU) to iteratively verify and refine the initial causal graph G_o. A three-tiered memory system (Working, Factual, and Experiential Memory) retrieves historical failure patterns via RAG to support evidence-based reasoning and continuous learning through self-reflection.

these graphs accurately by pruning spurious edges and uncovering implicit causal relations.

A naive solution is to present all causal edges from all suspicious services directly to a single LLM and ask it to assess their validity. However, this approach is prone to a critical form of hallucination. Because the skeleton graph is generally plausible, the LLM tends to accept the causal relations within each service as valid. Our key insight is that this ambiguity can be reduced by exploiting the rich semantic information encoded in metric names (*e.g.*, cpu.max.usage, mysql.slow.count) together with broader cross-service context. Based on this insight, we design a multi-agent architecture that separates high-level coordination from low-level causality verification, as illustrated in Fig. 6.

Agent definition. *KRCA* adopts a *master-slave* multi-agent architecture in which a single *Main Agent* serves as the central orchestrator and coordinates a pool of specialized *Sub Agents*. As described in Section 3.3, anomalous metrics are first mapped to broad meta-metric categories (*E*, *I*, *D*, *K*). Each meta-metric category covers multiple finer-grained failure scenarios. For example, an anomaly in the internal meta-metric may arise from a CPU bottleneck or a memory leak, and these cases require different diagnostic knowledge and tools. Accordingly, we define nine domain-specific Sub Agents aligned with classic failure scenarios (*e.g.*, Traffic Expert, CPU Expert, and GPU Expert). Each Sub Agent is equipped with a domain-specific prompt, common query tools, and specialized diagnostic tools (*e.g.*, SQL analyzer). The Main Agent maintains a global view of all G_o graphs and dispatches causality verification tasks to the appropriate Sub Agents. These Sub Agents collect concrete evidence to confirm or refute localized causal links, which enables the pruning of spurious edges and the discovery of implicit intra-meta-metric causality.

Memory system. To preserve reasoning continuity and exploit accumulated experience, *KRCA* incorporates a three-tier memory system. First, *Working Memory* records the short-term dialogue history for each service. It supports *cross-service activation*, which provides holistic context when a task involves multiple services. For long-term knowledge, we separate episodic cases from transferable insights to avoid retrieval interference. *Factual Memory* stores service-specific historical failures, enabling in-context learning from empirical evidence and thereby reducing hallucinations. *Experiential Memory* stores generalized reasoning insights (*e.g.*, "traffic saturation causes MySQL slow queries"), allowing the LLM to make informed judgments even for previously unseen services.

Retrieval-Augmented Generation. *KRCA* employs a RAG mechanism to incorporate historical knowledge dynamically. At each reasoning step, the module retrieves the top-3 relevant entries, prioritizing Factual Memory and then falling back to Experiential Memory. The Main Agent issues queries using the full alert context, whereas Sub Agents use task-specific metrics. The retrieval mechanism uses a hybrid scoring model that combines lexical similarity with temporal relevance. Given a query, the similarity score for a historical entry is computed as:

$$\text{Score}(Q, H) = \alpha \cdot S(Q_s, H_s) + \beta \cdot S(Q_m, H_m) + \gamma \cdot e^{-\lambda(T_Q - T_H)} \quad (6)$$

where $S(\cdot, \cdot)$ denotes BM25-based lexical similarity [31], computed over service names (subscript s) and anomalous metric names (subscript m), respectively, while the exponential term penalizes older entries. The weights α, β, and γ are tuned separately for each memory type to reflect their distinct retrieval objectives: Factual Memory prioritizes service names and recency to retrieve recent incidents from the same service, whereas Experiential Memory emphasizes metric names to retrieve transferable insights across services.

Putting it together. The collaboration begins with the Main Agent performing a global RAG query and decomposing graph refinement into parallel tasks. At the same time, Sub Agents perform focused RAG queries and use specialized tools to gather evidence. The Main Agent then synthesizes these findings to prune refuted edges and dispatches new tasks to identify missing intra-meta-metric links. Finally, the Main Agent generates a comprehensive failure report. The system then enters a self-reflection phase, during which case-specific details and generalizable insights are distilled into the memory system, enabling continuous evolution after each resolved alert.

4 EVALUATION

In this section, we evaluate the performance of *KRCA* with the aim of answering the following questions:

RQ1: How does *KRCA* compare with existing methods?
RQ2: Does each component contribute to *KRCA*?
RQ3: How sensitive is *KRCA* to key hyperparameters?
RQ4: How beneficial is *KRCA* in assisting SREs with RCA?

Table 1: Comparing *KRCA* with Existing Methods on the Sub-task of RCA.

Main Seed LLM	Sub Seed LLM	Method	Root Cause Service			Failure Type			Latency (s/case)
			AC@1	AC@3	Avg@3	AC@1	AC@3	Avg@3	
Claude-opus-4.6	**GPT-5.1**	*KRCA*	**0.88**	**0.96**	**0.92**	**0.79**	**0.98**	**0.89**	**146.59**
Claude-opus-4.6		CoT [46]	0.14	0.17	0.16	0.12	0.34	0.24	32.87
Claude-opus-4.6		ReAct [53]	0.42	0.47	0.45	0.29	0.44	0.37	211.48
Claude-opus-4.6		Reflexion [34]	0.48	0.52	0.50	0.37	0.57	0.48	411.80
Claude-opus-4.6		RCA-Agent [50]	0.57	0.62	0.60	0.47	0.72	0.61	342.80
GPT-5.2	**GPT-5.1**	*KRCA*	**0.77**	**0.89**	**0.84**	**0.66**	**0.88**	**0.78**	**129.36**
GPT-5.2		CoT	0.17	0.22	0.20	0.12	0.31	0.22	33.54
GPT-5.2		ReAct	0.41	0.49	0.45	0.29	0.45	0.38	166.82
GPT-5.2		Reflexion	0.47	0.55	0.51	0.35	0.53	0.45	298.47
GPT-5.2		RCA-Agent	0.55	0.64	0.60	0.44	0.67	0.57	255.91
MiniMax-M2.7	**DeepSeek-V3**	*KRCA*	**0.70**	**0.85**	**0.78**	**0.60**	**0.81**	**0.72**	**95.74**
MiniMax-M2.7		CoT	0.14	0.19	0.17	0.11	0.26	0.19	15.42
MiniMax-M2.7		ReAct	0.33	0.40	0.37	0.22	0.36	0.30	76.88
MiniMax-M2.7		Reflexion	0.38	0.46	0.42	0.27	0.43	0.36	136.91
MiniMax-M2.7		RCA-Agent	0.46	0.56	0.52	0.35	0.56	0.47	117.58

4.1 Experimental Setup

Dataset. To comprehensively evaluate the performance of *KRCA*, we constructed a real-world failure dataset from the monitoring platform of Kuaishou's hyper-scale microservice system. The dataset contains 300 failures that were sampled from October 2025 to March 2026, spanning multiple business lines (*e.g.*, Short Video, Recommendation, and E-commerce). In total, the dataset encompasses nine classical failure types commonly observed in hyper-scale microservice systems, including but not limited to abnormal traffic, CPU/GPU saturation and MySQL slow queries, thereby ensuring sufficient diversity for a robust evaluation. The ground truth for each failure, comprising both the root cause service and the failure type, was labeled by a dedicated annotation team consisting of five SREs with over five years of operational experience.

Evaluation metrics. We evaluate *KRCA* from three perspectives. For *accuracy*, we adopt AC@k and Avg@k, two metrics widely used in root cause analysis, computed separately for root cause service localization and failure type classification, where AC@k measures whether the correct answer appears in the top-k predictions, and $Avg@k = \frac{1}{k}\sum_{i=1}^{k} AC@i$. For *efficiency*, we measure end-to-end latency from alert triggering to the generation of the failure report. For *report usability*, we conduct a human-centered evaluation following [26, 61], inviting SREs to rate generated reports along two dimensions: *Readability* (clarity, logical organization, and technical fluency) and *Usefulness* (the extent to which the diagnosis supports timely failure resolution).

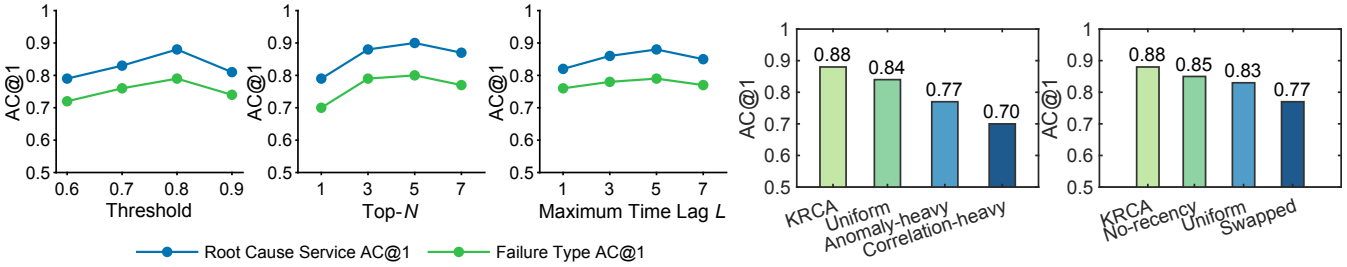
Baselines. We do not compare against deep learning methods [37, 55, 25], as training and maintaining such models in our hyper-scale microservice system would be prohibitively expensive. We therefore compare *KRCA* against four representative LLM-based baselines: three widely used agent reasoning frameworks and one RCA-focused agent framework. (1) CoT [46] improves diagnostic accuracy by breaking the task into intermediate reasoning steps; (2) ReAct [53] combines reasoning with action, allowing the model to interact with the environment while solving the task; (3) Reflexion [34] builds on ReAct by adding self-reflection, so the agent can critique and revise earlier reasoning traces; (4) RCA-Agent, introduced in OpenRCA [50], is designed for RCA and uses code-assisted data retrieval and analysis to reduce the burden of long-context diagnosis. All baselines are given the same diagnostic tools and the same historical knowledge base as *KRCA*, ensuring a fair comparison.

Implementation. The tool integration between agents and diagnostic tools is built upon the Model Context Protocol (MCP), using the open-source library[2]. For the API-level drilldown, we set the propagation threshold in Eq. 1 to 0.8, the maximum time lag $L = 5$. The latency score weights (w_1, w_2, w_3) in Eq. 3 are set to $(0.2, 0.5, 0.3)$. We retain the top-3 suspicious services before passing them to later modules, balancing diagnostic accuracy and end-to-end latency. For the memory retrieval weights (α, β, γ) in Eq. 6, we set $(0.5, 0.2, 0.3)$ for Factual Memory and $(0.4, 0.5, 0.1)$ for Experiential Memory. The 300 failure cases are used only for evaluation. All system parameters and the tiered memory are configured with routine daily alerts after removing failure-related alerts, ensuring that none of the evaluated incidents is involved in tuning.

4.2 RQ1: Overall Performance

Table 1 reports the diagnostic performance of *KRCA* and the baseline methods on real-world failures collected in Kuaishou's production environment. *KRCA* consistently outperforms all baselines in both root-cause service localization and failure-type classification, achieving AC@1 scores of 0.88 and 0.79, respectively. Compared with RCA-Agent [50], the strongest baseline, these results

[2]https://github.com/mark3labs/mcp-go



(a) Impact of the threshold, Top-N and the maximum time lag L in the API-level drilldown module on root cause service and failure type AC@1.

(b) Impact of balancing weights in the latency score (Eq. 3) and memory retrieve (Eq. 6) on root cause service AC@1.

Figure 7: Sensitivity analysis of key hyperparameters in *KRCA*.

correspond to absolute gains of 31% and 32%, respectively. We also evaluate different seed LLMs within *KRCA* by varying the underlying foundation model. The results show that *KRCA* preserves strong performance across different LLMs relative to the baselines, further demonstrating the robustness and generalizability of the framework.

Although CoT [46] benefits from the reasoning capability of LLMs, it performs well only on relatively simple failure cases, such as cases with no more than three anomalous services. ReAct [53] and Reflexion [34] can progressively narrow the search space through step-by-step reasoning, but their unconstrained reasoning process requires frequent tool calls and multiple interaction rounds, sharply increasing both latency and error rate. RCA-Agent [50], a purpose-built RCA framework, alleviates the long-context burden through code-assisted data retrieval and outperforms the general-purpose agent baselines. Nevertheless, it still lacks structural priors to manage the massive search space in hyper-scale systems, and its effectiveness is constrained by the model's error-handling capability during code execution. We further observe that in complex cases involving more than 100 anomalous services, all agent-based baselines frequently encounter context explosion or execution failures, causing the LLM to terminate without producing an answer. In contrast, *KRCA* follows a multi-stage diagnostic paradigm that systematically reduces the search space before LLM reasoning, enabling effective handling of complex failures while maintaining high accuracy in both root cause service localization and failure type classification.

4.3 RQ2: Ablation Study

To validate the necessity of the key design choices in *KRCA*, we conduct an ablation study in which each module is removed or simplified while the rest of the system remains intact. In *w/o API-level drilldown*, we replace API-level drilldown with service-level dependency traversal and pass all downstream services directly to the later modules. In *w/o Skeleton Graph*, we remove causal graph generation and provide the agents with raw anomalous metrics without explicit structure. In *w/o Multi-agent*, a single agent performs the entire reasoning process instead of multiple collaborating agents. In *w/o Tiered Memory*, the memory module is disabled, so agents rely only on the current incident and cannot retrieve similar past failures or their associated reasoning insights. In *w/ Naive RAG*, we replace the structured tiered retrieval scheme with flat similarity-based retrieval over a single memory pool that merges all memory types. We also set all weights in Eq. 6 to 1/3, so retrieval no longer distinguishes among different entry types. The results are reported in Table 2.

Table 2: Ablation study of *KRCA*.

Methods	Root Cause Service		Failure Type	
	AC@1	AC@3	AC@1	AC@3
KRCA	**0.88**	**0.96**	**0.79**	**0.98**
w/o API-level drilldown	0.64	0.82	0.76	0.95
w/o Skeleton Graph	0.72	0.91	0.61	0.92
w/o Multi-agent	0.75	0.88	0.67	0.89
w/o Tiered Memory	0.81	0.93	0.74	0.95
w/ Naive RAG	0.78	0.91	0.71	0.93

Without API-level drill-down, the system cannot filter services finely along invocation paths, so many irrelevant services remain in the candidate set. This mainly hurts root cause service localization, while failure type classification is less affected because the failure type is often still visible in upstream metrics. Without the skeleton graph, agents must reason over raw anomalous metrics without prior structure, which increases hallucination and reduces accuracy for both AC@1 root cause localization and failure type classification. Without multi-agent collaboration, all reasoning is handled by a single agent, which is less effective at verifying causality across heterogeneous metric categories. This leads to consistent degradation on both tasks. Disabling the memory module also reduces localization and classification performance, because the LLM can no longer draw on relevant historical cases and must reason from scratch. Replacing the proposed composite RAG with Naive RAG similarly causes clear performance drops in both tasks, because poorly matched retrieved cases mislead the LLM and divert its reasoning away from the true root cause.

4.4 RQ3: Hyperparameter Study

To assess the stability of *KRCA*, we perform a sensitivity analysis of its key hyperparameters. The analysis covers three parameters in the API-level drilldown module (Fig. 7(a)) and two sets of balancing weights (Fig. 7(b)). For the drilldown module, the threshold in Eq. 1 gives AC@1 values ranging from 0.79 to 0.88, with the best result at 0.8. Top-N produces AC@1 between 0.79 and 0.90, and all settings with $N \geq 3$ remain above 0.87. The maximum time lag L in Eq. 1 is

Table 3: Practical utility of *KRCA* rated by SREs.

SREs	Readability		Usefulness	
	Mean	HIP	Mean	HIP
SRE #1	4.30	86.67%	4.20	83.33%
SRE #2	4.27	86.67%	4.17	83.33%
SRE #3	4.20	83.33%	4.10	80.00%
SRE #4	4.17	80.00%	4.03	76.67%
SRE #5	4.11	80.00%	4.00	73.33%
Avg.	**4.21**	**83.33%**	**4.10**	**79.33%**

the most stable parameter, with AC@1 staying within 0.82 to 0.88 for $L \in 1, 3, 5, 7$. Although $N = 5$ achieves the highest AC@1 of 0.90, it also incurs much higher latency than $N = 3$ (191.2s vs. 146.6s). We therefore use Top-3 in production to balance diagnostic accuracy and real-time efficiency. For the latency score weights in Eq. 3, we compare *KRCA* $(0.2, 0.5, 0.3)$, Uniform $(1/3, 1/3, 1/3)$, Anomaly-heavy $(0.6, 0.2, 0.2)$, and Correlation-heavy $(0.2, 0.2, 0.6)$. The corresponding AC@1 values are 0.88, 0.84, 0.77, and 0.70, which suggests that the system remains stable unless correlation is given excessive weight. For the memory retrieval weights in Eq. 6, we compare *KRCA* (Factual: $(0.5, 0.2, 0.3)$, Experiential: $(0.4, 0.5, 0.1)$), Uniform (both $(1/3, 1/3, 1/3)$), Swapped (Factual: $(0.4, 0.5, 0.1)$, Experiential: $(0.5, 0.2, 0.3)$), and No-recency (Factual: $(0.7, 0.3, 0.0)$, Experiential: $(0.4, 0.6, 0.0)$). Their AC@1 values are 0.88, 0.83, 0.77, and 0.85, respectively. Excluding the structurally mismatched Swapped setting, performance stays within a narrow range of 0.83–0.88. Even under the least favorable configuration (*e.g.*, Correlation-heavy, AC@1 = 0.70), *KRCA* still outperforms all baselines reported in Section 4.2.

4.5 RQ4: Usability for SREs

To evaluate the practical utility of the failure reports generated by *KRCA* in real-world failure scenarios, we invited five SREs, each with at least five years of hands-on operational experience, to assess the reports. We randomly sampled 30 failure cases and adopted the same scoring criteria as [61], namely a 5-point Likert scale where 1 indicates the lowest quality and 5 indicates the highest. All SREs rated the reports independently to ensure the objectivity of the evaluation. As shown in Table 3, the average readability score reached 4.21, with a HIP (the percentage of samples rated 4 or above [26]) of 83.3%, indicating that the majority of reports are clearly structured and easy to comprehend. The average usefulness score reached 4.10, with a HIP of 79.3%, suggesting that most reports effectively assisted SREs in their troubleshooting process. These results, grounded in firsthand feedback from frontline SREs, confirm that *KRCA* provides tangible support during failure diagnosis in production environments.

5 DISCUSSION

5.1 Deployment

KRCA has continuously deployed in Kuaishou's internal monitoring platform, *Tianwen*, from October 2025 to March 2026. As shown in Fig. 8, it currently provides RCA support for more than 2,000 alerts per day across six major business lines, including Short Video,

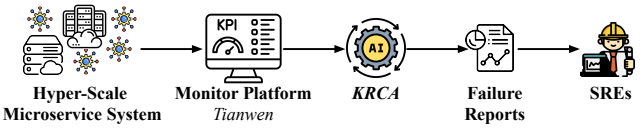


Figure 8: Deployment architecture of *KRCA*.

E-commerce, and Algorithms. During this period, we collected internal statistics on 483 emergency incidents[3]. For each incident, the ground-truth root cause service and failure type were established through postmortem analysis by the SREs responsible for the affected services. According to these records, *KRCA* correctly identified both the root cause service and the failure type in 82% of incidents, substantially reducing diagnosis effort in practice. We also analyzed the 18% of incidents for which *KRCA* failed to produce a correct diagnosis. These failures mainly fall into two categories. Firstly, some key downstream services lacked critical observability signals, such as availability-related metrics, which caused the drill-down process to terminate too early. Secondly, multiple cascading anomalies occurred simultaneously, causing several services to fail at once and making it difficult to separate the true propagation path from correlated noise.

Before the deployment of *KRCA*, incident diagnosis at Kuaishou mainly relied on manual, layer-by-layer inspection. Across all failure cases collected between 2024 and October 2025, the average time required to localize the root cause was about 52 minutes. After deployment, this average fell to 11.8 minutes for failures observed from October 2025 to March 2026, a 77.3% reduction. In our annual user survey, *KRCA* received a Net Promoter Score (NPS) of 74%, indicating strong satisfaction among SREs.

5.2 Case Study

To show how *KRCA* performs in practice, we present a real cascading failure from Kuaishou's production environment. The failure began with a coredump in a far-downstream storage service, storage_A. This failure triggered connection timeouts that propagated across three layers of dependencies and eventually appeared as a severe availability drop in the upstream recommendation service reco_B. Each layer contained dozens of candidate services, and several intermediate services showed different levels of availability degradation, making it difficult to separate the true propagation path from background noise. After the alert was triggered, *KRCA* automatically followed the anomaly along API invocation paths, quickly ruled out irrelevant branches, and identified storage_A as the primary suspect. It then organized the anomalous metrics into a structured causal graph and invoked domain-specific Sub Agents to collect supporting evidence. By combining the collected evidence, the agents found a sudden process termination in host-level logs and matched the case to similar historical incidents stored in the memory system, which confirmed a memory access violation as the root cause. The resulting diagnostic report was immediately sent to the responsible SREs, allowing rapid mitigation.

For this incident, before *KRCA* was deployed, diagnosing a similar cascading failure typically required about 30 minutes. With

[3]An emergency incident does not necessarily become a failure if it is mitigated quickly and causes only limited impact.

KRCA, the end-to-end diagnosis time for this case was reduced to about 5 minutes. Since the availability degradation of reco_B is estimated to cause a Gross Merchandise Volume (GMV) loss of $5,200 per minute, this improvement corresponds to an estimated reduction of more than $130,000 in potential business losses for this single incident.

5.3 Lessons Learned

Context engineering. We have invested significant effort in optimizing the LLM context within *KRCA* —including prompts and tool definitions—and have gained several key insights. First, managing context length is vital for balancing efficiency and accuracy. We have observed a noticeable decline in reasoning precision as prompt length increases, coupled with a sharp rise in inference latency. To mitigate this, we decompose complex logic into multiple sub-agents (Section 3.4) to keep individual prompts concise, while utilizing parallel execution to reduce end-to-end delay. Second, the robustness of tool invocation is critical for maintaining a stable reasoning trajectory. We found that erroneous or malformed tool returns can severely derail the multi-agent system, leading to outcomes that diverge significantly from expectations. Since different agents often interact with the same tool (*e.g.*, metric queries) using heterogeneous parameter schemas, providing generic descriptions is often insufficient. To ensure precise calls, we provide specialized tool versions and role-specific descriptions tailored to each agent's specific context. We have further enhanced this with techniques such as error-handling loops to safeguard the integrity of the diagnostic process.

Explainability. *KRCA* provides three key features to enhance result interpretability and assist SREs in troubleshooting: (1) link topology visualization, where a graph of suspicious downstream services is delivered immediately after API-level drilldown (Section 3.2) to provide a holistic view of the failure (latency < 5 s); (2) anomaly detection results (Section 3.3), which are performed concurrently across all suspicious services and surfaced in the *Tianwen* system to minimize the manual effort of inspecting metrics (latency < 10 s); (3) comprehensive failure reports, generated by the multi-agent system (Section 3.4) to offer a deep-dive failure analysis (latency < 3 mins). This design philosophy is rooted in progressive context disclosure. We have found that by providing diagnostic information in stages, SREs can obtain actionable insights at different time intervals, which significantly increases the adoption rate and operational trust in *KRCA* during real-world incidents.

6 RELATED WORK

Causal discovery. Causal discovery seeks to infer causal relations from observational data. Existing approaches generally fall into three categories: (1) statistical methods based on conditional independence tests (*e.g.*, PC algorithm and Granger causality) [5, 28, 29, 16, 21, 8]; (2) deep learning approaches, especially GNN-based methods, that learn causal structures from evolving topologies [42, 55]; and (3) recent LLM-based methods that extract causal intent from text [10, 39, 52]. However, these approaches exhibit fundamental limitations in hyper-scale microservice systems. Statistical methods are vulnerable to high-dimensional and coarse-grained monitoring data, and they often produce spurious relations when unrelated metrics surge simultaneously. Deep learning models struggle to remain effective as system scale increases, because both noise and dimensionality grow substantially. At the same time, the direct use of LLMs on massive time-series data is neither practical nor scalable. In contrast, *KRCA* addresses these limitations by instantiating a high-recall skeleton graph as a structural prior instead of inferring causality directly from raw time-series data, and then leveraging a multi-agent framework to verify fine-grained causal relations with rich metric semantics.

LLM-based RCA. The rapid progress of LLMs has inspired many automated RCA frameworks [27, 33, 38, 61], including tool-augmented autonomous agents (*e.g.*, RCAgent [44], Flow-of-Action [30]), in-context learning systems (*e.g.*, RCACopilot [6]), and domain-specific diagnostic bots (*e.g.*, D-Bot [62], OpenRCA [50], TrioXpert [36]). Although promising, these methods face severe scalability challenges in hyper-scale systems. They typically place LLMs directly in the RCA loop and expose them to a large and noisy search space from the beginning. As a result, the LLMs must depend on extensive multi-round reasoning and repeated tool use to narrow the candidate set, which significantly increases end-to-end latency and the risk of hallucination. This scalability bottleneck highlights the need for a more structured approach. *KRCA* addresses this issue by applying an efficient API-level drilldown strategy to sharply reduce the search space before introducing LLMs, thereby ensuring both real-time diagnostic efficiency and high accuracy in production environments.

7 CONCLUSION

In this paper, we present *KRCA*, an end-to-end root cause analysis system designed for hyper-scale microservice systems. To address the central challenges posed by extreme dynamism and massive scale, *KRCA* adopts a progressive multi-stage diagnostic paradigm. It first applies an API-level drilldown strategy to effectively reduce the search space and identify suspicious services. It then constructs a skeleton-based causal graph from anomalous metrics, which provides a high-recall structural prior for subsequent analysis. Finally, *KRCA* employs a memory-augmented multi-agent framework to collaboratively verify causal relationships and generate comprehensive failure reports. Extensive experiments on a dataset of 300 real-world failures collected from the production environment of Kuaishou show that *KRCA* significantly outperforms state-of-the-art baselines in both root cause service localization and failure type classification. In addition, *KRCA* has been deployed in a real-world production environment for more than six months, where it processes over 2,000 alerts per day and substantially reduces MTTR.

DATA AVAILABILITY STATEMENT

The datasets used in this study are not publicly available because they contain highly sensitive company-wide production data and proprietary infrastructure information. Due to confidentiality, security, and compliance requirements, these materials cannot be released. The source code is currently undergoing the company's internal open-source review process and will be released at https://anonymous.4open.science/r/KRCA-0FCC after desensitization and approval.

REFERENCES

[1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1737–1749.

[2] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 1–4.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site reliability engineering: how Google runs production systems*. " O'Reilly Media, Inc.".

[4] George EP Box and George C Tiao. 2011. *Bayesian inference in statistical analysis*. John Wiley & Sons.

[5] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. Causeinfer: automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1887–1895.

[6] Yinfang Chen et al. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, 674–688.

[7] Yinfang Chen et al. 2025. Stratus: a multi-agent system for autonomous reliability engineering of modern clouds. *arXiv preprint arXiv:2506.02009*.

[8] Yuxiao Cheng, Lianglong Li, Tingxiong Xiao, Zongren Li, Jinli Suo, Kunlun He, and Qionghai Dai. 2024. Cuts+: high-dimensional causal discovery from irregular time-series. In *Proceedings of the AAAI Conference on Artificial Intelligence* number 10. Vol. 38, 11525–11533.

[9] Tianyu Cui et al. 2025. Logeval: a comprehensive benchmark suite for llms in log analysis. *Empirical Software Engineering*, 30, 6, 173.

[10] Huaming Du, Yujia Zheng, Baoyu Jing, Yu Zhao, Gang Kou, Guisong Liu, Tao Gu, Weimin Li, and Carl Yang. 2025. Causal discovery through synergizing large language model and data-driven reasoning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 543–554.

[11] Tao Feng, Lizhen Qu, Niket Tandon, Zhuang Li, Xiaoxi Kang, and Gholamreza Haffari. 2025. On the reliability of large language models for causal discovery. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9565–9590.

[12] Ruowei Fu et al. 2025. Llm-powered multi-agent collaboration for intelligent industrial on-call automation. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2222–2234.

[13] Google Cloud Platform. 2021. Online boutique: a cloud-native microservices demo application. https://github.com/GoogleCloudPlatform/microservices-demo. Accessed: 2026-03-23. (2021).

[14] Clive WJ Granger. 1980. Testing for causality: a personal viewpoint. *Journal of Economic Dynamics and control*, 2, 329–352.

[15] Yongqi Han, Qingfeng Du, Ying Huang, Jiaqi Wu, Fulong Tian, and Cheng He. 2024. The potential of one-shot failure root cause analysis: collaboration of the large language model and small classifier. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 931–943.

[16] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems*, 35, 31158–31170.

[17] Gregory F Lawler and Vlada Limic. 2010. *Random walk: a modern introduction*. Vol. 123. Cambridge University Press.

[18] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal inference-based root cause analysis for online service systems with intervention recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3230–3240.

[19] Zeyan Li et al. 2023. Generic and robust root cause localization for multi-dimensional data in online service systems. *Journal of Systems and Software*, 203, 111748.

[20] Zeyan Li et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[21] Cheng-Ming Lin, Ching Chang, Wei-Yao Wang, Kuang-Da Wang, and Wen-Chih Peng. 2024. Root cause analysis in microservice using neural granger causal discovery. In *Proceedings of the AAAI Conference on Artificial Intelligence* number 1. Vol. 38, 206–213.

[22] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. Microhecl: high-efficient root cause localization in large-scale microservice systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 338–347.

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[24] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: how language models use long contexts. *Transactions of the association for computational linguistics*, 12, 157–173.

[25] Qihan Liu, Pengfei Chen, Guangba Yu, Yuanhao Lai, and Xiaoyun Li. 2025. Causelens: causality-based interpretable root cause analysis for microservice systems. In *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[26] Yilun Liu, Shimin Tao, Weibin Meng, Feiyu Yao, Xiaofeng Zhao, and Hao Yang. 2024. Logprompt: prompt engineering towards zero-shot and interpretable log analysis. In *Proceedings of the 2024 IEEE/ACM 46th international conference on software engineering: Companion proceedings*, 364–365.

[27] Yilun Liu et al. 2025. R-log: incentivizing log analysis capability in llms via reasoning-based reinforcement learning. *arXiv preprint arXiv:2509.25987*.

[28] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. 2020. Automap: diagnose your microservice-based web applications automatically. In *Proceedings of The Web Conference 2020*, 246–258.

[29] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. 2020. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[30] Changhua Pei et al. 2025. Flow-of-action: sop enhanced llm-based multi-agent system for root cause analysis. In *Companion Proceedings of the ACM on Web Conference 2025*, 422–431.

[31] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. 2004. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, 42–49.

[32] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring llm-based agents for root cause analysis. In *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*, 208–219.

[33] Binpeng Shi et al. 2025. Flowxpert: expertizing troubleshooting workflow orchestration with knowledge base and multi-agent coevolution. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 4839–4850.

[34] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36, 8634–8652.

[35] Peter Spirtes, Clark N Glymour, and Richard Scheines. 2000. *Causation, prediction, and search*. MIT press.

[36] Yongqian Sun, Yu Luo, Xidao Wen, Yuan Yuan, Xiaohui Nie, Shenglin Zhang, Tong Liu, and Xi Luo. 2025. Trioxpert: an automated incident management framework for microservice system. *arXiv preprint arXiv:2506.10043*.

[37] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. Art: a unified unsupervised framework for incident management in microservice systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 1183–1194.

[38] Yongqian Sun et al. 2025. Llm-augmented ticket aggregation for low-cost mobile os defect resolution. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 215–226.

[39] Yuni Susanti and Michael Färber. 2025. Paths to causality: finding informative subgraphs within knowledge graphs for knowledge-based causal discovery. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2778–2789.

[40] Guangya Wan, Yunsheng Lu, Yuqi Wu, Mengxuan Hu, and Sheng Li. 2024. Large language models for causal discovery: current landscape and future directions. *arXiv preprint arXiv:2402.11068*.

[41] Chenxu Wang et al. 2025. Towards llm-based failure localization in production-scale networks. In *Proceedings of the ACM SIGCOMM 2025 Conference*, 496–511.

[42] Dongjie Wang, Zhengzhang Chen, Yanjie Fu, Yanchi Liu, and Haifeng Chen. 2023. Incremental causal graph learning for online root cause analysis. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, 2269–2278.

[43] Yidan Wang, Zhouruixing Zhu, Qiuai Fu, Yuchi Ma, and Pinjia He. 2024. Mrca: metric-level root cause analysis for microservices via multi-modal data. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 1057–1068.

[44] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. Rcagent: cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 4966–4974.

[45] Zeying Wang et al. 2025. Kaiops: a platform solution of end-to-end multi-modal aiops for ai training at scale. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 3192–3203.

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35, 24824–24837.

[47] Canhua Wu et al. 2021. Identifying root-cause metrics for incident diagnosis in online service systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 91–102.

[48] Zhe Xie et al. 2026. Foundroot: towards foundation model for root cause analysis via structured deep thinking.

[49] Zhe Xie et al. 2024. Microservice root cause analysis with limited observability through intervention recognition in the latent space. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, 6049–6060.

[50] Junjielong Xu et al. 2025. Openrca: can large language models locate the root cause of software failures? In *The Thirteenth International Conference on Learning Representations*.

[51] Jian Yang, Zian Wang, Shuangwu Chen, Huasen He, Yunpeng Hou, and Xiaofeng Jiang. 2025. Hg-pad: heterogeneous graph structure learning aided performance anomaly diagnosis in microservice systems. *IEEE Transactions on Services Computing*.

[52] Xiaojie Yang, Hangli Ge, Jiawei Wang, Zipei Fan, Renhe Jiang, Ryosuke Shibasaki, and Noboru Koshizuka. 2025. Causalmob: causal human mobility prediction with llms-derived human intentions toward public events. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, 1773–1784.

[53] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

[54] Zhenhe Yao et al. 2024. Sparserca: unsupervised root cause analysis in sparse microservice testing traces. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 391–402.

[55] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 553–565.

[56] Shenglin Zhang, Xiaoyu Feng, Runzhou Wang, Minghua Ma, Wenwei Gu, Yongqian Sun, Zedong Jia, Jinrui Sun, and Dan Pei. 2025. Too many cooks: assessing the need for multi-source data in microservice failure diagnosis. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 1–12.

[57] Shenglin Zhang, Chenyu Zhao, Yicheng Sui, Ya Su, Yongqian Sun, Yuzhi Zhang, Dan Pei, and Yizhe Wang. 2021. Robust kpi anomaly detection for large-scale software services with partial labels. In *2021 IEEE 32nd international symposium on software reliability engineering (ISSRE)*. IEEE, 103–114.

[58] Shenglin Zhang et al. 2024. Illuminating the gray zone: non-intrusive gray failure localization in server operating systems. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 126–137.

[59] Shenglin Zhang et al. 2023. Robust failure diagnosis of microservice system through multimodal data. *IEEE Transactions on Services Computing*, 16, 6, 3851–3864.

[60] Chenyu Zhao et al. 2023. Robust multimodal failure detection for microservice systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 5639–5649.

[61] Yongxin Zhao et al. [n. d.] When llms listen to experts: accurate failure diagnosis in operating systems.

[62] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-bot: database diagnosis system using large language models. *arXiv preprint arXiv:2312.01454*.

[63] Zhouruixing Zhu, Cheryl Lee, Xiaoying Tang, and Pinjia He. 2024. Hemirca: fine-grained root cause analysis for microservices with heterogeneous data sources. *ACM Transactions on Software Engineering and Methodology*, 33, 8, 1–25.