A Goal-Driven Survey on Root Cause Analysis

AOYANG FANG, HAOWEN YANG, HAOZE DONG, QISHENG LU, JUNJIELONG XU, and PINJIA HE, The Chinese University of Hong Kong, Shenzhen, China

Root Cause Analysis (RCA) is a crucial aspect of incident management in large-scale cloud services. While the term *root cause analysis* or *RCA* has been widely used, different studies formulate the task differently. This is because the term "RCA" implicitly covers tasks with distinct underlying goals. For instance, the goal of localizing a faulty service for rapid triage is fundamentally different from identifying a specific functional bug for a definitive fix. However, previous surveys have largely overlooked these goal-based distinctions, conventionally categorizing papers by input data types (*e.g.*, metric-based vs. trace-based methods). This leads to the grouping of works with disparate objectives, thereby obscuring the true progress and gaps in the field. Meanwhile, the typical audience of an RCA survey is either laymen who want to know the goals and big picture of the task or RCA researchers who want to figure out past research under the same task formulation. Thus, an RCA survey that organizes the related papers according to their goals is in high demand. To this end, this paper presents a goal-driven framework that effectively categorizes and integrates 135 papers on RCA in the context of cloud incident management based on their diverse goals, spanning the period from 2014 to 2025. In addition to the goal-driven categorization, it discusses the ultimate goal of all RCA papers as an umbrella covering different RCA formulations. Moreover, the paper discusses open challenges and future directions in RCA.

CCS Concepts: • **Software and its engineering → Software post-development issues**.

Additional Key Words and Phrases: Root Cause Analysis, Incident Management

ACM Reference Format:
Aoyang Fang, Haowen Yang, Haoze Dong, Qisheng Lu, Junjielong Xu, and Pinjia He. 2025. A Goal-Driven Survey on Root Cause Analysis. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (October 2025), 55 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Microservices have emerged as the favored architecture for cloud-native development in the era of cloud computing. The adoption of microservice architecture can break down large, monolithic software applications into numerous smaller, more manageable components. This division facilitates parallel development across various software segments and enhances overall agility and efficiency. However, unlike monolithic applications, where components are tightly integrated and easier to trace, microservices operate as separate entities that interact through well-defined interfaces. This increases the complexity of the interactions between services, making it more challenging to pinpoint the origin of incidents. Incidents in large-scale cloud services can lead to significant financial losses and disruptions in critical services [51, 159, 163]. For example, the 2024 CrowdStrike-related IT outages [95] caused widespread service outages affecting Azure, Teams, and Xbox Live, resulting in prolonged downtime for many users, and the worldwide financial

Authors' address: Aoyang Fang, aoyangfang@link.cuhk.edu.cn; Haowen Yang, 222010523@link.cuhk.edu.cn; Haoze Dong, haozedong@link.cuhk.edu.cn; Qisheng Lu, qishenglu@link.cuhk.edu.cn; Junjielong Xu, junjielongxu@link.cuhk.edu.cn; Pinjia He, hepinjia@cuhk.edu.cn, The Chinese University of Hong Kong, Shenzhen, Shenzhen, Guangdong, China.

Manuscript submitted to ACM

damage has been estimated to be at least 10 billion US dollars. Root Cause Analysis (RCA) has emerged as a critical phase in identifying the underlying reasons for incidents. Traditional RCA requires substantial human effort to sift through vast quantities of telemetry data, code, and other resources [188]. Operators involved are expected to have extensive domain knowledge, such as a comprehensive understanding of the operational environment, familiarity with the codebase, and in cloud infrastructure scenarios, even insights into operating system-level components like the Linux kernel. Due to the complex inter-service dependency and intra-service business logic, it is challenging for operators to fully understand the full scope of service interactions [107], thus making root cause analysis difficult.

The practical application of RCA is not a monolithic task. As detailed in our formalization (Section 3), its objectives are fundamentally shaped by the incident management lifecycle (Section 2). This distinction is best understood by differentiating two key concepts, which we will formalize in Section 3. A Site Reliability Engineer (SRE) focused on rapid mitigation (minimizing Mean-Time-To-Recovery) seeks to identify the *trigger*: an event, such as a sudden traffic spike, that activates a latent flaw. In contrast, a developer aiming for a permanent fix must find the underlying *root cause*: the fundamental flaw itself, such as a buggy code commit. This difference in objectives, compounded by the varying data access permissions across roles, creates fundamentally different requirements for input data, analysis depth, and output granularity. This inherent complexity has led to significant fragmentation in RCA research. Most studies implicitly tailor their problem formulation to a specific goal, creating tight coupling between methods and narrow task definitions. For example, one line of research focuses on identifying faulty services from metrics [135, 170, 190] (typical SRE goals), while another aims to pinpoint buggy code changes from traces and logs [108, 221] (developer goals). This has resulted in a combinatorial explosion of task-specific solutions, making it exceedingly difficult to compare methods, generalize findings, or build a cohesive understanding of the field.

Consequently, the conventional survey's categorization based on input data types (*e.g.*, logs, traces, metrics) [178, 197, 228] fails to capture the underlying objectives that drive these formulations. This taxonomy is misleading because the relationship between input data and research objectives is not one-to-one; it arbitrarily groups studies with different goals that happen to use similar data, while separating those with shared goals that use different data types. For newcomers or practitioners seeking a high-level understanding, this view obscures the bigger picture of what RCA can achieve. For researchers, it makes comparing the true capabilities of different methods nearly impossible, as a technique's performance is inextricably tied to a narrow, implicit goal. This lack of a unified, goal-oriented perspective imposes a significant cognitive load, impeding not only the synthesis of academic knowledge but also the integration of disparate tools into a cohesive industrial workflow.

To this end, we anchor our perspective in the overarching goal of incident management: minimizing the **Mean Time to Recovery** (MTTR). As discussed in Section 2, this lifecycle involves a chain of activities where the output of one stage, with a specific granularity, becomes the input for the next. The effectiveness of RCA is central to this process, as its output directly dictates the speed and precision of resolution. A coarse-grained root cause may only permit service rollbacks, whereas a fine-grained cause, such as a specific faulty code commit, enables targeted fixes and prevents future recurrence. This lifecycle-centric view reveals that an ideal RCA solution must satisfy a set of fundamental requirements to be effective in practice.

Instead of categorizing research by input data types, we argue that a more insightful taxonomy should be based on the inherent goals an RCA system must achieve to excel at each stage of this lifecycle. We derive these goals by analyzing the key challenges from four perspectives: the data foundation (Input), the analytical core (Inference), the usability of results (Output), and practical deployment constraints (Efficiency). An ideal RCA system must be able to:

correlate **multi-dimensional data**, be **robust** to imperfect data, **adaptively learn** from system changes, provide **interpretable** and **multi-granularity** results that are **actionable**, all while maintaining **real-time performance**.

These seven goals, which include multi-dimensional data correlation, robustness, adaptive learning, real-time performance, interpretability, multi-granularity, and actionability, form the pillars of our goal-driven survey. They are not arbitrary but are directly derived from the practical needs of the incident management lifecycle, and will be formally defined in Section 3.

To systematically investigate the field through this goal-driven lens, we first establish a formal framework for the ideal RCA problem, defined as a function $\mathcal{F} : \mathcal{O} \rightarrow \mathcal{G}$ that maps rich observational data ($\mathcal{O}$) to a complete incident propagation graph ($\mathcal{G}$) (Section 3). This framework serves as a "north star," allowing us to deconstruct and unify the disparate problem formulations found in the literature. Following a rigorous survey methodology (Section 4), we collected and analyzed 135 papers from top-tier venues. Each paper is mapped onto our seven-goal taxonomy, enabling a structured analysis of the field's state of the art. Beyond this categorization, we provide a comprehensive overview of the research landscape, including publication trends, a summary of public benchmarks, datasets, and open-source tools (Section 12). Finally, we discuss the implications of our framework, identify critical gaps between current research and the ideal RCA, and outline promising future research frontiers (Section 13).

This work makes the following contributions:

- **A Formal Framework and Goal-Driven Taxonomy.** We propose a formal definition of the ideal RCA problem and introduce a novel, goal-driven taxonomy based on seven fundamental objectives. This framework moves beyond superficial data-based categorizations to provide a more insightful lens for understanding and comparing RCA research.
- **A Systematic and Comprehensive Survey.** We conduct a systematic review of 135 papers, analyzing each through our goal-driven taxonomy. This provides a structured overview of the state of the art, revealing the underlying design trade-offs and evolutionary trends in the field.
- **A Curated Overview of the Research Landscape.** We compile and analyze the distribution of research focus, public benchmarks, datasets, and open-source tools. This serves as a valuable guide for researchers and practitioners, while also highlighting critical limitations in existing resources, such as the lack of ground-truth propagation graphs.
- **Identification of Gaps and Future Frontiers.** We identify the significant gap between the current "point-finding" paradigm and the ideal "graph-building" RCA. Based on this analysis, we outline key future research directions, including the need for next-generation benchmarks and unified models for causal graph generation.

The remainder of this paper is structured as follows, also illustrated in Fig. 1. Section 2 and Section 3 introduce the background and our formal framework. Section 4 details our survey methodology. Section 5 to Section 11 present a systematic analysis of RCA research through the lens of our seven-goal taxonomy. Section 12 and Section 13 discuss research trends and future opportunities. Finally, Section 14 reviews related surveys, and Section 15 concludes the paper.

2 BACKGROUND

2.1 MICROSERVICE

Core Concepts. The microservice is an *implementation of decomposition philosophy* in software architecture, which advocates breaking down complex systems into smaller, independent components. Specifically, the microservice

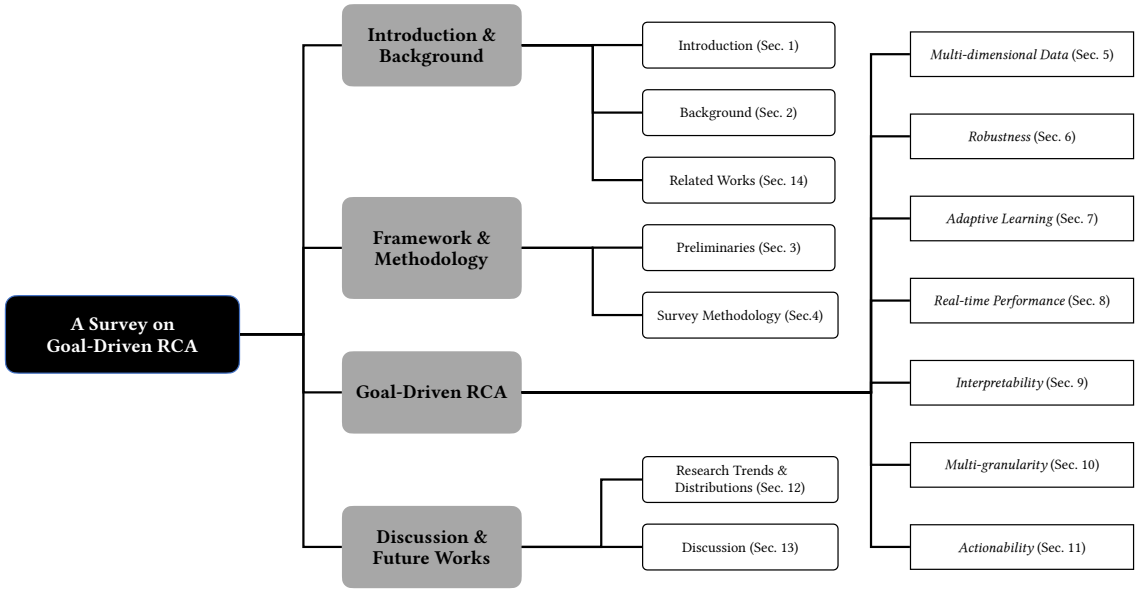

Fig. 1. The structure of this survey. We first introduce the background, our formal framework, and survey methodology (§2-§4). Then, we systematically analyze RCA research through the lens of a seven-goal taxonomy (§5-§11). Finally, we discuss research trends, future opportunities, and related work before concluding (§12-§15).

architecture divides a single application into a collection of small, lightweight services, each running in its own process and communicating via lightweight methods, often using an HTTP API. It emphasizes agile, DevOps practices, decentralized data management, and governance [96].

Technology Stack. Microservice architecture is supported by a series of infrastructure systems and techniques that work together seamlessly. Firstly, the development of microservices begins with development frameworks like Spring Boot [46] and Dubbo [6], which facilitate the creation of microservices by providing essential functionalities such as REST clients, database integration, externalized configuration, and caching. Subsequently, these microservices are deployed using containerization tools like Docker [5], which enhance portability, flexibility, efficiency, and speed. To manage these containers effectively, runtime infrastructure frameworks such as Spring Cloud [47], Mesos [38], Kubernetes [35], and Docker Swarm [5] are employed, offering capabilities like configuration management, service discovery, service registry, and load balancing. Finally, to ensure efficient and reliable development and deployment processes, continuous integration and delivery tools such as Jenkins [34] and GitLab CI/CD [31] are utilized to support ongoing integration and delivery efforts.

Benefits of Microservice Architecture. Microservice architecture offers several notable benefits, making it a popular choice for modern application development. By allowing each service to be updated independently, it ensures that changes or failures in one service do not impact the entire application. Additionally, it supports independent scaling, which enhances system flexibility and optimizes resource utilization. This architecture also facilitates parallel development, enabling multiple teams to work on different services simultaneously, thereby accelerating development speed [134].

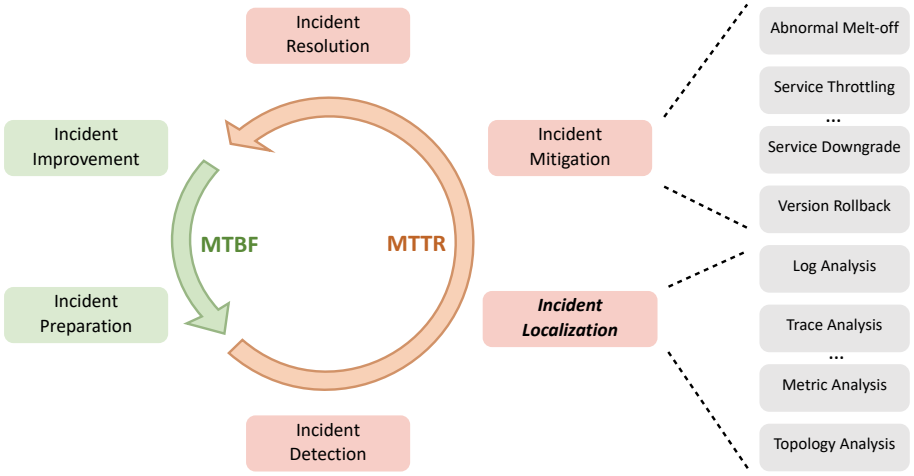

Fig. 2. Procedure of incident management. Starting from **incident preparation**, which involves techniques like software testing, canary releases, and disaster recovery simulations to closely mimic real-world conditions and prepare for potential issues. Once the product is deployed, it enters the **incident detection** stage, where monitoring systems, anomaly detection, and customer reports are used to identify any abnormalities in runtime services. If an issue arises, the process moves to the **incident localization** phase. Here, SREs work to quickly determine which service is the root cause and further pinpoint the specific underlying issues, including analyzing the log, metric, trace, etc. To halt the failure's propagation, the next step is **incident mitigation**, where common actions might include downgrading the service or rolling back to a previous code version, depending on the identified root cause. Following mitigation, the focus shifts to **incident resolution**. During this phase, SREs and developers collaborate to resolve the incident completely. Once the system is fully restored, the final phase is a **incident improvement** (postmortem review), which involves analyzing the incident to extract lessons learned and implementing measures to prevent future occurrences.

Challenges and Complexities. However, as systems transition from monolithic to microservice architectures, complexity shifts from internal code to interactions between services. This shift introduces new challenges, particularly in monitoring and troubleshooting, as the interactions between services become more intricate. Modern applications, typically involving hundreds of interconnected services, significantly complicate monitoring efforts [202]. For instance, distinguishing between individual service failures and cascading effects from other service failures becomes challenging. Moreover, many microservice failures originate from external environments, like their runtime environments, communication, or coordination issues.

2.2 INCIDENT MANAGEMENT

Drawing on Google's incident management practices [33], as illustrated in Fig.2, we can break down the process into several key stages. This lifecycle begins with proactive *preparation* and moves into reactive phases like *detection*, *localization*, *mitigation*, and *resolution* once an incident occurs, finally concluding with *improvement* through postmortems.

The Pivotal Role of RCA in Incident Response. Within this lifecycle, Root Cause Analysis (RCA) is not a single, isolated step but a continuous and multi-faceted process that bridges incident detection with effective mitigation and resolution. In a practical corporate setting, the nature of RCA dynamically adapts based on the incident's progression and the immediate goal. These practical demands directly motivate the fundamental goals that an ideal RCA system must pursue, which we will formalize in Section 3.

The RCA process typically begins immediately after an incident is detected. The initial phase is often a coarse-grained analysis, commonly known as *triage*. The primary goal here is not to find the precise bug, but to quickly identify the responsible service or component and route the alert to the correct on-call team [88, 104]. This demand for speed highlights the critical need for *real-time performance* in any practical RCA tool. For example, a high-level alert on transaction failures might trigger a triage process that, by examining system topology and top-level service metrics, determines the payment service is the most likely culprit, thus assigning the incident to the payment team.

Once the incident is assigned, the responsible team, typically Site Reliability Engineers (SREs), performs a more in-depth RCA with the immediate goal of *mitigation*. At this stage, the objective is to find a "good enough" root cause to stop the bleeding (i.e., minimize Mean Time to Recovery, MTTR). SREs must correlate heterogeneous data sources like metrics, logs, traces, and recent changes (*e.g.*, deployments, feature flag toggles) to localize the fault. This process underscores the challenge of achieving effective *multi-dimensional data correlation*. Furthermore, since this data is often incomplete or noisy, the underlying methods must exhibit *robustness*. The output of this RCA directly enables mitigation actions, such as service rollbacks, emphasizing the need for findings to be *actionable*. For instance, identifying a specific canary instance as faulty leads to its removal from the load balancer.

After the service is stabilized through mitigation, the focus shifts to *permanent resolution*. This involves an even deeper and more fine-grained RCA, often conducted by developers in collaboration with SREs. The goal now is to uncover the precise, underlying bug or misconfiguration, which requires the RCA results to offer *multi-granularity* views, from service-level down to code-level. This requires meticulous forensic analysis, potentially down to a specific line of code or a configuration value. To be useful for developers, the causal chain leading to the failure must be clear, demanding a high degree of *interpretability* from the RCA model. For example, while the SRE's RCA identified a faulty deployment for rollback, the developer's RCA must pinpoint the exact code commit that introduced a memory leak. This fine-grained output is essential for developing a permanent fix, ensuring the incident does not recur and allowing the system to *adaptively learn* from past failures.

This progression illustrates that RCA's required input and desired output granularity evolve throughout the incident lifecycle. It begins broadly (which team is responsible?), narrows for mitigation (which service or deployment should be rolled back?), and becomes highly specific for resolution (which line of code/config needs to be fixed?). The effectiveness of the entire incident management process hinges on the ability to perform RCA at these varying levels of depth and speed.

3 PRELIMINARIES

This section introduces the key terminology and definitions that form the foundation of this paper. First, we introduce the core challenges and fundamental trade-offs inherent in RCA, from which we derive the seven goals that guide our survey. We then present a formal definition of the RCA problem to provide a unified conceptual model for analyzing existing work. Understanding these concepts is essential for discussing the challenges and goals of root cause analysis.

3.1 The Core Challenge: The Effectiveness-Data-Cost Triangle

Achieving the ideal goals of RCA is profoundly difficult. The challenges stem from the inherent complexity of telemetry data and a systemic trade-off that governs all practical RCA deployments. This tension forces strategic compromises that shape how existing research approaches the RCA problem.

Data Complexity: Fragmented, Imperfect, and Overwhelming Telemetry. The observation space O, which we will formally define in Section 3.3, presents formidable obstacles to effective analysis. Telemetry data is inherently *fragmented*, with different data types (*e.g.*, logs, metrics, traces) offering complementary but siloed views of system operation. The data is also *imperfect*, suffering from inaccuracies due to measurement errors, incompleteness from collection failures, and the inherent *sparsity* of failure-related signals in predominantly healthy systems. Moreover, observational data often suffers from *coverage gaps*, where certain system components, network segments, or application layers remain unmonitored due to instrumentation limitations, cost constraints, or architectural blind spots. In practice, RCA systems frequently must operate with only *partial observability*, where critical telemetry types may be missing entirely for specific services or time windows, forcing practitioners to infer root causes from incomplete evidence. Furthermore, the sheer *volume* and *velocity* of telemetry generation pose significant challenges for real-time processing and storage. Finally, the highly *dynamic* nature of modern distributed systems means that both the system topology and telemetry patterns continuously evolve, complicating model stability and generalization. These properties collectively force researchers to narrow their analytical scope to manageable subsets of the input space.

The Inherent Trade-off. These data complexities give rise to a fundamental trade-off that governs all practical RCA systems. Achieving desired **RCA effectiveness** is inextricably linked to the quantity and quality of available **observational data**. However, increasing data granularity and collection scope to improve analytical power inevitably leads to higher **resource costs** for data ingestion, storage, transmission, and computation. This creates a three-way tension that we term the **Effectiveness-Data-Cost Triangle**. This tension forces researchers and practitioners to navigate competing priorities. Some prioritize effectiveness by **expanding observational data** (*e.g.*, using eBPF), accepting higher costs. Others focus on **enhancing data utilization** through advanced algorithms to extract more value from existing data. A third approach emphasizes **improving computational efficiency** to enable more powerful analysis within the same cost envelope. This fundamental tension explains why the idealized goal of RCA is often simplified into more tractable sub-problems.

3.2 Seven Fundamental Goals of RCA

The Effectiveness-Data-Cost triangle dictates that no single solution can simultaneously perfect all aspects of RCA. Instead, existing research makes strategic compromises, focusing on tackling specific facets of this complex problem space. This provides the central insight for our survey: the seemingly ad-hoc nature of RCA research is, in fact, a collection of rational, focused efforts to push the boundaries of this trade-off triangle along specific vectors. These vectors can be distilled into seven fundamental goals that an ideal RCA system must pursue. As introduced in Section 1, these goals directly map to the needs of different stages in the incident management lifecycle, where the ultimate objective is to reduce Mean Time to Recovery (MTTR) and extend Mean Time Between Failures (MTBF) [99]. To provide a clear foundation for our taxonomy, we now define the scope and boundaries of each goal.

- **Multi-dimensional Data Correlation.** This goal addresses the challenge of fusing heterogeneous telemetry data (*e.g.*, metrics, logs, traces) into a unified representation for analysis. Research in this area focuses on developing techniques for semantic alignment, such as creating shared embedding spaces or modeling cross-modal dependencies. It is distinct from *robustness*, as it assumes data availability and prioritizes semantic integration over handling data imperfections.
- **Robustness.** This goal concerns the ability of an RCA model to function effectively with imperfect data, including noise, sparsity, and incompleteness. Methods in this category aim to infer causality from sparse signals,

reconstruct incomplete system topologies, or denoise telemetry streams. It differs from *adaptive learning* by addressing static data deficiencies, whereas the latter focuses on dynamic model adaptation to system evolution.

- **Adaptive Learning.** This goal focuses on enabling RCA models to evolve continuously in response to changes in the system's architecture, workload, or failure modes. The primary concern is developing mechanisms for online or incremental learning that obviate the need for complete model retraining, thus ensuring sustained performance in dynamic environments.
- **Real-time Performance.** As a non-functional requirement, this goal prioritizes the computational efficiency of RCA to ensure timely analysis during live incidents. The focus is on latency reduction through algorithmic optimization, parallelization, or approximation techniques. A method's contribution is measured by its speedup, distinguishing it from goals like *interpretability*, which concerns the causal accuracy of the output rather than the speed of its generation.
- **Interpretability.** This goal aims to make RCA results understandable and trustworthy for human operators. The emphasis is on generating causally sound and logically coherent explanations, such as incident propagation graphs or natural language summaries. It seeks to answer the "why" and "how" of a failure, in contrast to *multi-granularity*, which focuses on pinpointing the "where" at different abstraction levels.
- **Multi-granularity.** This goal is to achieve precise fault localization across multiple levels of abstraction, from high-level service dependencies down to specific code lines or configuration parameters. Its defining characteristic is the hierarchical depth and precision of the output, enabling drill-down analysis. This distinguishes it from the data-fusion focus of *multi-dimensional data correlation* and the explanatory nature of *interpretability*.
- **Actionability.** This goal focuses on translating diagnostic findings into concrete remedial actions. It bridges the gap between identifying a root cause and recommending a solution, such as suggesting a code rollback, generating a configuration patch, or retrieving relevant mitigation procedures from historical data. Unlike other goals centered on diagnosis, actionability is uniquely concerned with automated remediation.

3.3 A Formal View of the RCA Problem

We formalize Root Cause Analysis (RCA) as a function $\mathcal{F} : \mathcal{O} \rightarrow \mathcal{G}$, where $\mathcal{O}$ denotes the hierarchical observation data and $\mathcal{G}$ represents the incident propagation graph. As shown in Fig. 3, the RCA process consists of three main components: observation (input), inference, and output. This section first defines the input space $\mathcal{O}$ and the output space $\mathcal{G}$, and then discusses the role of this formalization as a conceptual framework for our survey.

3.3.1 Input Space: Observation Data. The input space $\mathcal{O}$ of RCA consists of the comprehensive telemetry data collected from various sources within a system. We define the observation space as:

Definition 3.1 (Input Space).

$$\mathcal{O} = \{\mathcal{L}, \mathcal{M}, \mathcal{T}, \mathcal{E}, \mathcal{D}\}$$

Where the components are defined as follows:

- $\mathcal{L}$ (**Logs**): Timestamped records of discrete events, crucial for debugging specific errors and capturing system behavior.
- $\mathcal{M}$ (**Metrics**): Numerical measurements aggregated over time, offering high-level views of system health and performance indicators.

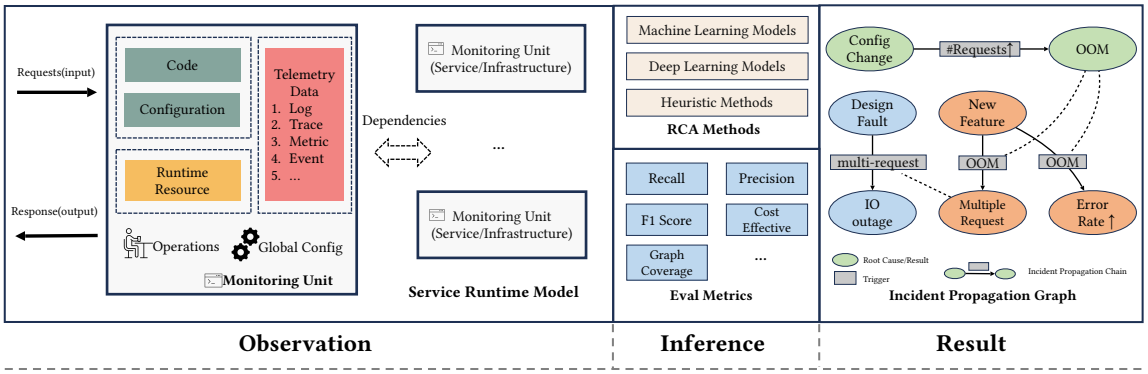

Fig. 3. Overview of Root Cause Analysis. The **observation** space includes monitoring units as its basic building blocks. Each unit contains static files (such as configuration and code) and the runtime resources (for example, CPU, memory, and network) needed to transform these static programs into processes that interact with other services. Additionally, there is telemetry data that describes the runtime behavior of these processes, as well as the operations and global configurations maintained during maintenance. The **inference** component comprises various implemented methods alongside their corresponding evaluation metrics; its practical application is critically constrained by **efficiency** requirements (*e.g.*, real-time performance). Finally, the **output** identifies the incident's root causes and illustrates how the root cause propagates to the observed symptoms. This includes an incident propagation graph composed of an incident propagation chain. Each chain contains the root cause, trigger (optional), and result nodes.

- $\mathcal{T}$ (**Traces**): Data representing the end-to-end journey of requests across multiple services, providing context for distributed transactions through spans.
- $\mathcal{E}$ (**Events**): Named occurrences at specific instants in time, representing discrete actions or state changes within a system [167]. Events are the fundamental building blocks of telemetry and can be aggregated into logs, traces, and metrics [164–166].
- $\mathcal{D}$ (**Supplementary Data**): Additional contextual information for RCA, including code, configuration files, and design documentation.

Together, these complementary data types enable operators to gain insights into system behavior, performance, and reliability, serving different aspects of the incident management lifecycle discussed in Section 2. Note that the observation space is often incomplete and noisy due to the inherent challenges in data collection and system complexity, as discussed in Section 3.1.

3.3.2 Output Space: Incident Propagation Graph. The output of RCA is an **Incident Propagation Graph**, which models the causal sequence of events that constitute an incident. It is a composite of single or multiple failures, illustrating the relationships among events. This graphical representation formalizes the mental model that Site Reliability Engineers (SREs) intuitively build when diagnosing cascading failures.

Definition 3.2 (Output Space). An Incident Propagation Graph is a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of event nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the causal dependencies between them. Each node $v \in \mathcal{V}$ can be categorized by its role in the incident. We identify three primary roles: *Root Cause, Trigger, and Symptom.* To formalize the causal dynamics, we draw an analogy to chemical reactions. The **root cause** (r) is a reactive component creating a latent failure condition (*e.g.*, a memory leak in code). The optional **trigger** (t) acts as a catalyst that activates this condition (*e.g.*, a sudden traffic spike). The **symptom** (s) is the observable manifestation of the failure (*e.g.*, an OOM error). This formulation captures the essential temporal and causal relationship: the root cause establishes the

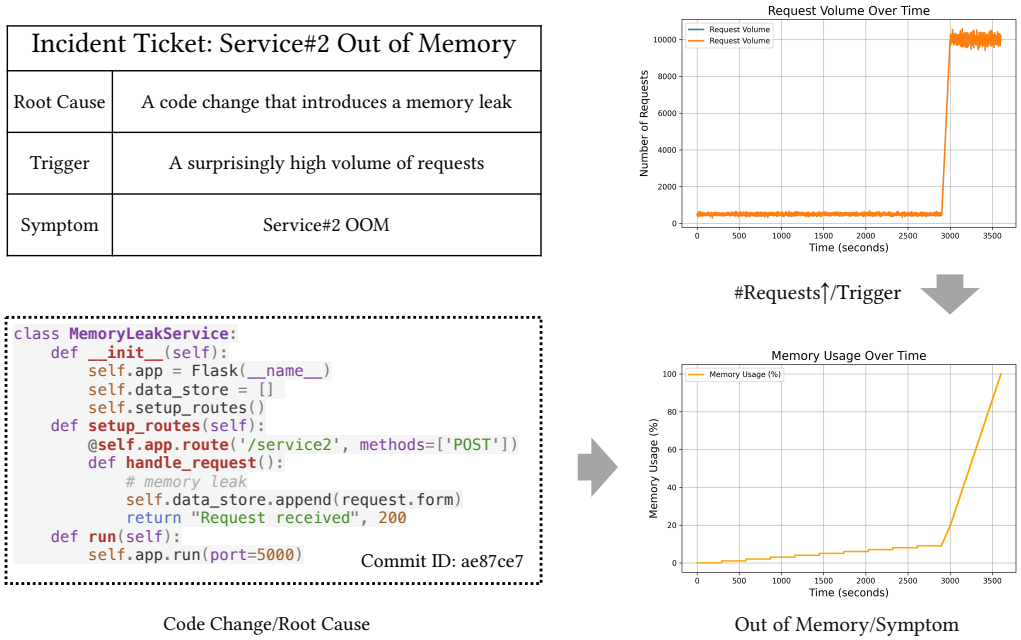

Fig. 4. An Out-of-Memory (OOM) incident propagation chain. The OOM event in Service#2 is the observed **symptom**. The **root cause** is the code change within Service#2. The **trigger** is the increased traffic volume, which activates the latent flaw. Mitigating the trigger (*e.g.*, traffic reduction) provides a temporary fix, while resolving the root cause (*e.g.*, fixing the code) offers a permanent solution.

precondition for failure, the trigger (if present) catalyzes the transition from latent to manifest failure, and the symptom is the observable outcome. Similar to how catalysts are not consumed in a reaction, triggers are not root causes but facilitate the manifestation of symptoms, explaining why mitigating a trigger offers a quick fix while only resolving the root cause provides a permanent solution. The optional nature of triggers parallels reactions that can proceed without a catalyst under specific conditions.

Definition 3.3 (Root Cause Event Node). An event node $r \in \mathcal{V}$ is a *root cause* if it represents an initial fault or change and has an in-degree of zero in the incident propagation graph $\mathcal{G}$. That is, $\forall v \in \mathcal{V}, (v, r) \notin \mathcal{E}$. An incident may have one or more root causes. As illustrated in Fig. 4, the root cause is the fundamental flaw, like a memory leak in a recent code change. In practice, RCA systems often output a ranked list of *root cause candidates (RCCs)*.

Definition 3.4 (Trigger Event Node). An event node $t \in \mathcal{V}$ is a *trigger* if it is an event that activates a latent failure condition introduced by a root cause. A trigger is not a root cause itself but acts as a catalyst. The presence of a root cause alone may not lead to a failure; the failure manifests only when the trigger occurs. For example, a sudden traffic spike (the trigger) can cause a service with a memory leak (the root cause) to finally crash. Suppressing the trigger can quickly restore service, but does not fix the underlying issue. In many scenarios, a trigger may not be present or identifiable, making this component optional.

Definition 3.5 (Symptom Event Node). An event node $s \in \mathcal{V}$ is a *symptom* if it represents an observed anomaly that directly initiates an incident response (*e.g.*, an OOM error or a service outage). It is the manifestation of a failure chain and, in many cases, corresponds to a node with an out-degree of zero in the graph $\mathcal{G}$.

Definition 3.6 (Intermediate Event Node). An event node $v \in \mathcal{V}$ is an *intermediate event* if it is neither a root cause nor a symptom that initiated the response. It acts as a symptom for its upstream events and a cause for its downstream events within a propagation chain.

3.3.3 Framework as a Unified Conceptual Model. The formalization $\mathcal{F} : \mathcal{O} \rightarrow \mathcal{G}$ represents an **idealized and comprehensive goal** for RCA that encompasses the full spectrum of input-output paradigms found across existing literature. For an SRE, identifying the root cause(s) answers the "what" question. The incident propagation graph $\mathcal{G}$, particularly the propagation paths, answers the critical "how" and "why" questions. The propagation path serves as the essential evidence chain to verify the correctness of the identified root cause, understand the incident's blast radius, and fundamentally prevent future recurrences.

However, reconstructing the complete propagation graph $\mathcal{G}$ faces enormous challenges, as detailed in Section 3.1. Consequently, the vast majority of existing research simplifies the problem. Most studies focus only on identifying a subset of event nodes from the graph, typically the root cause event node (r) and sometimes the symptom event node (s), rather than the full set of causal edges ($\mathcal{E}$) and trigger event nodes (t). Their outputs often present a root cause at varying granularities (*e.g.*, service level, metric level, component level, or pod level) without the explanatory power of a full propagation path. Notably, *no existing work fully conforms to our idealized definition by outputting complete incident propagation paths*, highlighting the forward-looking nature of our survey.

While this formalization represents an idealized goal, its primary utility in this survey is to serve as a unified conceptual framework. It allows us to deconstruct the diverse and seemingly disconnected problem formulations in existing literature. By mapping each study's inputs and outputs onto our defined space ($\mathcal{O}$ and $\mathcal{G}$), we can precisely articulate *how* they simplify the problem. For instance, many approaches focus solely on identifying the root-cause event node (r) rather than the entire graph. This approach not only enables a more insightful comparison of different methods but also systematically charts the path forward for future, more comprehensive RCA research that moves from "finding points" to "constructing graphs."

This distinction is critical and forms a central theme of our analysis. Mitigating the trigger, which is often the focus of SREs for immediate service restoration, offers a temporary fix to minimize MTTR. In contrast, resolving the root cause, which is the primary goal for developers, provides a permanent solution to improve MTBF. Recognizing this dichotomy is essential for understanding why different RCA methods produce outputs of varying nature and granularity, as they are implicitly optimized for different objectives within the incident management lifecycle.

4 SURVEY METHODOLOGY

4.1 SURVEY SCOPE

Root cause analysis is a broad topic, applicable in a wide range of scenarios where it is essential to determine the causes behind a particular situation and how that situation arises. Examples include questions such as "Why an individual may have a high income"[112], "Why intermittent slow queries occur in databases"[156], and "Why failures happen in microservices" [192, 240].

In this survey, we focus specifically on research that investigates the identification of root causes and how these causes contribute to observed behaviors in microservice systems, typically in the form of violations of expectations

Table 1. Venues Included in the Systematic Literature Review. We systematically searched 135 top-tier conferences and journals across four primary research domains, covering Software Engineering, Systems and Distributed Computing, Artificial Intelligence and Databases, and related disciplines.

Domain	Conferences	Journals
Software Engineering	ICSE, FSE/ESEC, ASE, ISSTA, ICST, ISSRE	TOSEM, TSE, JSS
Systems, Networking, & Distributed Computing	DSN, SIGCOMM, SIGMETRICS, EuroSys, ASPLOS, Middleware, ICDCS, INFOCOM, ICWS, CCGRID, GLOBECOM, ICC, IPCCC, IWQoS	TSC, TDSC
AI, Data Mining, & Databases	KDD, NeurIPS, ICLR, AAAI, SIGMOD, VLDB, CIKM, EMNLP, IJCNN	—
Other Related Disciplines	WWW, ICSOC, APNOMS, ICCBR, SMC, ISPA	—

such as Service Level Objectives (SLOs). These systems are often characterized by complex environments with intricate service interactions. We exclude papers focused on fault localization [201], as this represents a narrower scope within the broader context of root cause analysis for microservices, primarily concerned with identifying vulnerable code segments. Notably, relevant fault injection benchmarks include Defects4j [130].

4.2 PAPER COLLECTION

We conducted a systematic literature review of RCA research across four primary research domains. To ensure comprehensive coverage, we systematically searched leading venues in Software Engineering, Systems and Distributed Computing, Artificial Intelligence and Databases, and other related disciplines. The specific conferences and journals included in our search are presented in Table 1.

To identify relevant literature, we initially conducted a manual search of the DBLP database, focusing on key conferences and using specific keywords such as "Root Cause", "Fault Localization," "Micro-Service", "Detection", and "Localization". Recognizing the diverse nature of the RCA community, with its publications scattered across various venues and employing different terminologies, we aimed to ensure comprehensive coverage of the field. To achieve this, we adopted a snowballing approach as recommended in [124]. This involved both backward and forward snowballing techniques:

- **Backward Snowballing:** We reviewed the reference lists of each collected paper to identify additional relevant papers within our scope.
- **Forward Snowballing:** Using Google Scholar, we identified papers that cited our initially collected papers, thereby expanding our pool of relevant literature.

This iterative process was repeated until we reached a saturation point where no new relevant papers were identified. To maintain a high standard of quality, we ceased searching papers from non-top conferences and those with low citation counts from further reference list searches.

Furthermore, we limited our search to publications from the last ten years, as microservices have only gained significant traction after Google open-sourced Kubernetes [35]. Through this comprehensive search methodology, we identified and collected 135 top papers directly related to Root Cause Analysis.

4.3 PAPER ANALYSIS

To ensure a thorough and rigorous analysis of the collected papers, we adopted a systematic approach closely tied to the theoretical framework proposed in Section 3. This process aimed to deconstruct and categorize each research work within our unified conceptual model of RCA.

The first two authors undertook an extensive reading and examination of the full text of each paper. For each paper, we extracted key information according to our idealized RCA definition ($\mathcal{F} : \mathcal{O} \rightarrow \mathcal{G}$). Specifically, we identified:

- **Input Space ($\mathcal{O}$)**: What types of telemetry data does the research utilize (*e.g.*, logs, metrics, traces, events) and any supplementary data (*e.g.*, configurations, code changes).
- **Output Space (Simplified forms of $\mathcal{G}$)**: What does the research aim to identify? Root cause nodes (r), trigger nodes (t), symptom nodes (s), or partial paths between them? What is the granularity of the output (*e.g.*, service-level, instance-level, code-line-level)?
- **Core Methods and Challenges**: What methods or models does the research employ to address the problem? Which of our identified seven core goals does it primarily aim to tackle (*e.g.*, is it for improving **robustness** against noisy data, or achieving **real-time performance**)?
- **Evaluation and Resources**: What evaluation methods and datasets does the research use? Are its code and datasets open-source?

Through this structured information extraction, we formalized each paper's specific problem formulation and mapped it to our goal-driven taxonomy. For example, a research work focusing on quickly localizing faulty services from noisy metric data would be categorized as primarily addressing **robustness** and **real-time performance** goals. This approach enables us to move beyond surface-level categorization based on input data types, revealing the deeper design philosophies and trade-offs underlying different research works.

When the two primary authors had differing interpretations or findings about paper categorization or information extraction, they conducted discussion sessions with additional co-authors. These discussions were instrumental in resolving disagreements and ensuring a consensus on the categorization of papers and the extracted data. The involvement of co-authors, who possess extensive expertise in RCA and microservices, helped maintain the accuracy and integrity of the analysis.

All authors independently reviewed the content to ensure the reliability and consistency of the survey's findings. This review process was designed to identify and correct any potential errors, inconsistencies, or omissions. By employing this rigorous multi-step analysis and review process, we ensured the credibility and robustness of our survey.

5 GOAL 1: MULTI-DIMENSIONAL DATA CORRELATION

As defined in the preliminaries (Section 3), the input space $\mathcal{O}$ for RCA comprises a heterogeneous collection of logs, metrics, traces, events, and supplementary data. Each data type offers a unique yet siloed perspective on system behavior. The goal of multi-dimensional data correlation is to overcome this fragmentation by fusing these diverse data sources into a unified and coherent view. This fusion establishes a solid foundation for precise fault diagnosis. This section focuses on the data fusion techniques themselves, examining how existing works "translate" and "align" these heterogeneous data silos. We categorize these approaches into three mainstream paradigms: **fusion via unified representation learning**, **fusion via graph structures**, and the emerging **semantic fusion via large language models (LLMs)**.

5.1 Fusion via Unified Representation Learning

This paradigm aims to map data from different modalities into a shared, low-dimensional vector space (embedding space), thereby enabling the analysis of originally incomparable data (such as discrete log events and continuous metric sequences) within a unified mathematical framework.

Table 2. Overview of Data Fusion Paradigms and Representative Works for RCA.

Fusion Paradigm	Method Category	Papers
Unified Representation	Eventization & Embedding	DiagFusion [227], Nezha [221], UniDiag [230], DeepHunt [182], Chain-of-Event [216]
	Multi-modal Learning	**Contrastive:** MULAN [240], TVDiag [207]
		Attention/Gating: Eadro [133], FAMOS [98], Medicine [187]
		Multi-stage: ART [184]
		Voting: PDiagnose [117]
Graph Structures	Heterogeneous Graph	TrinityRCL [108], CHASE [239], FaaSRCA [119], SpanGraph [132], GIED [116]
	Cross-layer Fusion	MicroRCA [204], Microscope [145]
	Graph Ensemble	FRL-MFPG [92], Chen et al. [90], CloudRCA [237]
	Domain-Specific	**Trace-based:** TraceAnomaly [150]
		Data Structure: TLCluster [181], LogKG [180]
		Multi-metric: AutoMAP [155], MS-Rank [153], CMMD [212]
		Temporal-structural: GAMMA [179]
	Case-based	MicroCBR [148], MicroTR [215], SynthoDiag [232]
	Correlation-based	HeMiRCA [245], ICWS'20 [195], Log3C [115], MRCA [198]
LLM-based Semantic	NL Transform	RCACopilot [91], TrioXpert [183], SCELM [185], X-lifecycle [105]
	Knowledge-Data	**Documentation:** Atlas [209], RealTCD [136]
		Code: Raccoon [238], COCA [139]
		Multi-agent: SynergyRCA [206], KnowledgeMind [172], ThinkFL [226]

5.1.1 Eventization and Embedding. A core strategy involves abstracting all data into a unified "event" format and then leveraging this abstraction for further analysis. For instance, *DiagFusion* [227] and *Nezha* [221] uniformly transform metric fluctuations, log entries, and trace anomalies into event sequences, employing FastText or similar models to generate embeddings for these events, thus enabling semantic similarity computation in vector space. *Chain-of-Event* [216] also leverages this eventization framework to unify heterogeneous signals, but its primary goal is to use the resulting event stream to automatically learn a weighted, interpretable event-causal graph from historical data. *UniDiag* [230] and *DeepHunt* [182] follow a similar approach but utilize the fused event sequences to construct temporal knowledge graphs or directly feed them into graph autoencoders to further learn normal system behavior patterns.

5.1.2 Multi-modal Learning Architectures. An alternative technical route employs **multi-modal learning architectures** that end-to-end learn fused representations through carefully designed neural networks. These methods typically design specialized encoders for different data sources to preserve their unique characteristics, and then integrate information through a fusion module. *MULAN* [240] and *TVDiag* [207] introduce contrastive learning to not only learn modality-invariant features but also preserve modality-specific information, effectively preventing information loss during the fusion process. *Eadro* [133], *FAMOS* [98], and *Medicine* [187] leverage gating mechanisms or cross-attention to dynamically weight the contributions of different data sources, thereby generating more context-aware unified representations. *ART* [184] further employs a serialized multi-stage model (Transformer, GRU, GraphSAGE) to separately capture cross-channel, temporal, and topological dependencies, ultimately forming a "unified fault representation" that supports various downstream diagnostic tasks. *PDiagnose* [117] adopts a more lightweight approach, using a weighted voting mechanism to combine evidence from KPIs, traces, and logs.

5.2 Fusion via Graph Structures

Unlike approaches that compress all information into vectors, graph-based fusion methods explicitly model different data entities and their relationships by constructing heterogeneous graphs. In such graphs, nodes can represent services, instances, hosts, metric types, or even code entities, while edges denote call, hosting, causal, or correlation relationships among them.

5.2.1 Graph Construction Strategies. The key to these methods lies in graph construction strategies *TrinityRCL* [108] and *CHASE* [239] construct heterogeneous graphs containing multiple types of nodes, including services, hosts, metrics, and log anomalies, transforming the data fusion problem into a graph node feature learning problem. *FaaSRCA* [119] extends this concept to serverless environments by building a "Global Call Graph" that integrates multi-modal data from both the application and the platform layers. *SpanGraph* [132] focuses on a finer granularity, building a graph where nodes are spans and edges are invocations, with features enriched by metrics and configuration files. *GIED* [116] fuses numerical metrics and categorical service attributes into a unified graph representation to distinguish critical incidents from noise. The contribution of *MicroRCA* [204] and *Microscope* [145] lies in their fusion of application-layer metrics with infrastructure-layer metrics, and their computation of correlations between cross-domain metrics to weight graph edges, thereby establishing cross-layer causal associations. *FRL-MFPG* [92] fuses two distinct graphs (a call dependency graph extracted from traces and an association graph mined from historical faults) to construct a more comprehensive fault propagation model. *Chen et al.* [90] achieves fusion by creating an ensemble model where different base learners specialize in either metric data or expert knowledge, with a meta-learner combining their outputs. Similarly, *CloudRCA* [229] fuses KPIs, logs, and system topology into a Knowledge-informed Hierarchical Bayesian Network, using a probabilistic graphical model to perform inference.

5.2.2 Domain-Specific Graph Integration. Some works deeply integrate graph structures with specific data types. For example, *TraceAnomaly* [150] proposes Service Tracing Vectors (STV) that ingeniously fuse structured information (call paths) and numerical information (response times) into a single vector, which is then learned through Deep Bayesian Networks to capture patterns. Others focus on creating integrated data structures. *TLCluster* [181] creates a unified "trace log" data structure by combining traces with corresponding execution logs, enabling more precise instance-level fault localization through clustering. *LogKG* [180] focuses on integrating structured fields (such as component IDs) and unstructured content within logs by constructing a knowledge graph, thereby capturing a more comprehensive context than analyzing log messages alone. *AutoMAP* [155] and *MS-Rank* [153] focus on fusing multiple performance metrics by constructing individual impact graphs for different metrics and then merging them into a composite graph to capture more complex performance issues. *CMMD* [212] specifically addresses the fusion of different metrics by using a Graph Neural Network to automatically model the calculation relationships between fundamental metrics and derived KPIs. *GAMMA* [179] integrates temporal patterns from multi-variate metrics as node features into a structural dependency graph derived from traces.

5.2.3 Case-based and Similarity-based Fusion. Another distinct approach involves fusing heterogeneous data to enable case-based reasoning or diagnosis via similarity analysis. These methods typically construct a knowledge graph or a "fault fingerprint" from multiple data sources for a given incident. *MicroCBR* [148] combines anomalies from metrics, logs, and traces into such a fingerprint, which is then embedded into a spatio-temporal knowledge graph that also integrates system topology from a CMDB (Configuration Management Database). This enables the retrieval of similar historical cases using a weighted longest common subsequence algorithm. Similarly, *MicroTR* [215] reproduces the

execution state of transactions by building a multi-faceted knowledge graph that integrates temporal data, textual data from logs, call dependencies from traces, and performance metrics. Diagnosis is performed via similarity analysis against a historical database of labeled transactions. *SynthoDiag* [232] also employs a knowledge graph to fuse execution logs, trace logs, and test case metadata for diagnosing test alarms. It uses Knowledge Graph Embedding and Sentence-BERT to create unified vector representations, followed by a k-Nearest Neighbors classifier to determine the fault category.

5.2.4 Correlation-based Fusion. Beyond graph-based fusion, other works establish correlations between different data types to guide analysis. *HeMiRCA* [245] links a global anomaly score derived from traces to the behavior of individual metrics using Spearman correlation. *ICWS'20* [195] correlates log-derived anomaly scores with raw metric time series using Mutual Information. *Log3C* [115] explicitly correlates the frequency of log clusters with KPI degradation to identify impactful problems. *MRCA* [198] first fuses logs and traces for anomaly detection, then uses metrics from anomalous services for causal analysis.

5.3 Semantic Fusion via Large Language Models

The emergence of large language models (LLMs) has opened new pathways for multi-dimensional data correlation. Their core advantage lies in powerful natural language understanding and generation capabilities, enabling them to act as "universal translators" that bridge the vast semantic gaps among unstructured, semi-structured, and structured data.

5.3.1 Natural Language Transformation. A mainstream approach involves **transforming all heterogeneous data into natural language descriptions** and providing them as context to LLMs for unified reasoning. *RCACopilot* [91], *TrioXpert* [183], and *SCELM* [185] aggregate and summarize information from metrics, logs, traces, and change tickets into unified textual reports through automated workflows or dedicated data preprocessors, upon which LLMs perform root cause inference. *X-lifecycle* [105] further enriches this context by fusing incident data with historical knowledge bases and service dependency information.

5.3.2 Knowledge-Data Integration. A more sophisticated fusion approach leverages LLMs to **connect human knowledge with machine data**. *Atlas* [209] and *RealTCD* [136] utilize LLMs to parse unstructured texts such as system design documents and operation manuals, extracting a priori causal knowledge and transforming it into structured graphs or constraints to guide and optimize traditional causal discovery algorithms based on numerical metrics, thereby achieving effective integration of human experience and machine data. *Raccoon* [238] employs fault trees as an intermediate semantic layer, using LLMs to map user-reported natural language symptoms to fault tree nodes, which are then associated with specific code changes, successfully connecting the user perception layer with the engineering implementation layer. *SynergyRCA* [206] constructs runtime data into a knowledge graph, where LLMs actively explore and fuse information within the graph by generating query statements (Cypher), achieving dynamic, on-demand data fusion. *COCA* [139] fuses runtime issue reports with static source code by reconstructing execution paths, including across RPC boundaries, to create a code-aware context for the LLM. *KnowledgeMind* [172] uses a multi-agent architecture where a verifier agent fuses processed information from specialized log, metric, and trace agents. *ThinkFL* [226] achieves dynamic fusion through a reinforcement learning agent that adaptively decides whether to query trace or metric data at each step of its reasoning process.

Table 3. Overview of Robustness Strategies and Representative Works for RCA.

Strategy	Sub-strategy	Papers
Handling Observational Blind Spots	Structural Reconstruction	**Constraint-based:** CloudRanger [196], ServiceRank [154], MicroCause [158], DyCause [168], GrayScope [231], HRLHF [194], ICWS'17 [127]
		Learning-based: CMDiagnostor [223], MonitorRank [131], Murphy [114], CausalRCA [210], RUN [144]
		Multi-modal/LLM: MULAN [240], RealTCD [136]
	Inference of Unobserved Components	LatentScope [208], RCSF [193]
Ensuring Algorithmic Resilience	Tolerance to Sparsity & Imbalance	**Sparsity-specific:** MicroCU [128], SparseRCA [217], FaaSRCA [119]
		Imbalance (Learning): DeepHunt [182], OCRCL [121], TVdiag [207], LasRCA [113], SLIM [173], Raccoon [238]
		Imbalance (Data-level): DiagFusion [227], ICWS'20 [195], Medicine [187]
	Statistical Aggregation & Filtering	**Robust Statistics:** CauseRank [152], MicroRank [219], BARO [170], Squeeze [142], ShapleyIQ [138], ϵ-Diagnosis [175]
		Signal Enhancement: FChain [161], SwissLog [137], TraceRCA [141], WinG [214]
	Resilience to Novelty & Dynamics	CloudRCA [237], MicroIRC [244], UniDiag [230]
	Resilience via Agent Frameworks	**Knowledge-guided:** Flow-of-Action [169], KnowledgeMind [172]
		Consensus-based: mABC [233], RCAgent [200]

6 GOAL 2: ROBUSTNESS

As established in Section 3, a fundamental challenge for any practical Root Cause Analysis (RCA) system is the imperfect nature of its input data O. Real-world telemetry is invariably marred by noise, incompleteness, or sparsity, which can severely undermine diagnostic accuracy. The goal of **robustness**, therefore, is to ensure that an RCA system can maintain high accuracy and stability even when the quality of its input data is compromised. To address this challenge, we organize the surveyed techniques into two primary strategies that tackle distinct facets of data imperfection. The first, **handling observational blind spots**, focuses on reconstructing missing structural or behavioral information through inference, which is critical when dependency graphs are absent or key components lack direct monitoring. The second, **ensuring algorithmic resilience**, concerns methods that are intrinsically designed to tolerate noise, data sparsity, and other inconsistencies within the available telemetry. Together, these strategies enable RCA systems to deliver reliable insights under the adverse data conditions typical of production environments.

6.1 Handling Observational Blind Spots

Observational blind spots are a primary challenge to robust RCA, arising when telemetry data O is structurally or behaviorally incomplete This occurs when system topology information is missing, leaving a collection of isolated metrics, or when direct monitoring of a component is infeasible, providing only indirect signals Robust systems overcome these blind spots using causal discovery and inference to reconstruct the missing information.

6.1.1 Structural Reconstruction from Observational Data. In dynamic microservice environments, maintaining an accurate, up-to-date dependency graph is a significant challenge. Many robust RCA methods therefore operate without a predefined topology, instead inferring causal structure directly from observational data. These approaches can be broadly categorized into statistical and learning-based methods.

Constraint-based statistical methods, particularly those based on the Peter-Clark (PC) algorithm, are widely used to construct causal graphs by testing for conditional independence among time-series metrics *CloudRanger* [196] and *ServiceRank* [154] exemplify this approach by dynamically building an "impact graph" from performance metrics, which then guides a random walk to identify the root cause Recognizing that the standard PC algorithm's i.i.d assumption is violated by time-series data, *MicroCause* [158] introduces the PCTS algorithm to explicitly model propagation delays, yielding a more robust causal graph for intra-service analysis Similarly, Granger causality, which tests if one time series can forecast another, is another popular statistical tool *DyCause* [168] applies it in sliding windows to build dynamic causality maps, while *GrayScope* [231] and *HRLHF* [194] integrate it with expert knowledge to constrain the search space and improve accuracy in noisy OS-level environments. Other statistical approaches tackle imperfect data sources, such as mining control flows from interleaved logs that lack transaction identifiers [127].

Learning-based methods offer more flexibility in modeling complex, non-linear, and even cyclic dependencies that are common in enterprise systems but problematic for traditional statistical tests. For instance, *CMDiagnostor* [223] uses regression on traffic patterns to resolve ambiguities in call metric data, thereby reconstructing a more accurate call graph. Similarly, *MonitorRank* [131] enhances robustness against imperfect static call graphs by using a probabilistic random walk and correcting for un-modeled correlations discovered in historical data. *Murphy* [114] pioneers the use of Markov Random Fields (MRFs) to explicitly model and reason about cyclic dependencies, a critical capability for diagnosing issues like resource contention. *MULAN* [240] enhances robustness against noisy data by using a KPI-aware attention mechanism to dynamically assess the reliability of different data modalities (metrics and logs) before fusing them into a unified causal graph, effectively down-weighting imperfect data sources. *RealTCD* [136] improves robustness by leveraging large language models to incorporate domain knowledge as a high-quality prior, guiding the causal discovery process and making it more resilient to imperfect interventional data where the interventional targets are unknown. Other approaches leverage advanced neural architectures. *CausalRCA* [210] employs a gradient-based method (DAG-GNN) to learn non-linear causal relationships, while *RUN* [144] combines neural Granger causality with a novel contrastive learning scheme to robustly handle the complex periodicities in real-world metric data.

6.1.2 Inference of Unobserved Components and States. Beyond missing structure, RCA systems must often contend with unobserved components, where the true root cause candidate (RCC) lacks direct monitoring. Robustness in this context means inferring the state of these hidden variables from their observable effects. *LatentScope* [208] directly tackles this problem by modeling unobservable RCCs as latent variables in a dual-space graph. It then uses a regression-based algorithm to recognize interventions in this latent space, effectively identifying the unmonitored root cause from the behavior of related, observable metrics. Similarly, *RCSF* [193] infers the health of unmonitored functional components by analyzing their performance logs, enabling diagnosis in systems with incomplete monitoring coverage.

6.2 Algorithmic Resilience to Data Imperfections

Even when structural information is available, the data itself can be sparse, noisy, or imbalanced. Algorithmic resilience refers to the intrinsic ability of a method to function effectively despite these data quality issues.

6.2.1 Tolerance to Data Sparsity and Imbalance. Data sparsity is a common problem, especially in testing environments or systems with low sampling rates. Several methods are explicitly designed to operate under such constraints. *MicroCU* [128] addresses extreme data sparsity (*e.g.*, 60-80% missing) not by perfecting imputation, but by using "causal unimodalization" to calibrate and extract a reliable signal from the noisy causal curves derived from sparse data. *SparseRCA* [217] targets sparse testing traces by performing analysis at the individual span level, avoiding the need

for data aggregation altogether. This challenge is particularly acute in serverless environments, where telemetry is discontinuous; *FaaSRCA* [119] addresses this by analyzing system snapshots rather than continuous time-series, making it robust to such transient data.

Data imbalance, particularly the scarcity of labeled failure instances, poses another major challenge. *DeepHunt* [182] addresses the lack of labeled data through a self-supervised graph autoencoder, enabling a "zero-label cold start", and employs a data augmentation module that randomly masks input features to improve generalization from insufficient historical data. Contrastive learning has emerged as a powerful technique to address this. *OCRCL* [121] and *TVdiag* [207] both employ contrastive learning frameworks to learn discriminative representations from limited labeled data, effectively augmenting the sparse failure signals. *LasRCA* [113] takes this to an extreme in a one-shot learning scenario, using an LLM as a programmatic labeler to augment the training set for a small classifier. Other methods tackle imbalance at the algorithmic level; for instance, *SLIM* [173] uses a submodular optimization framework to directly optimize the F1-score, making it inherently robust to the minority fault class. Data augmentation is another common strategy, used by *DiagFusion* [227] and *ICWS'20* [195] to generate synthetic samples for rare failure types, preventing model bias. In cases with very few historical incidents, *Raccoon* [238] generalizes from sparse 'seed' causal knowledge using a Tree GNN, enabling it to identify root causes even when direct historical evidence is unavailable. *Medicine* [187] is explicitly designed to be robust against missing or low-quality data modalities through a parallel stream architecture and Multimodal Adaptive Optimization (MAO) module, allowing it to maintain high diagnostic accuracy even when one modality is compromised.

6.2.2 Robustness Through Statistical Aggregation and Filtering. Instead of relying on individual data points, which may be noisy, some methods achieve robustness by focusing on aggregate statistical patterns. *CauseRank* [152] combats noise from numerous, similar abnormal metrics by grouping them before performing causal discovery, preventing the analysis from being misled by spurious correlations. *MicroRank* [219] enhances spectrum-based fault localization by using Personalized PageRank to weight traces based on their rarity, ensuring that unique, informative traces are not drowned out by common, noisy ones. *BARO* [170] achieves robustness against inaccurate anomaly detection start times by using non-parametric statistics (median and IQR) instead of mean and standard deviation, making its analysis less sensitive to outliers. *Squeeze* [142] is designed to be robust to anomalies of both significant and insignificant magnitudes by using a Generalized Potential Score (GPS) that is less sensitive to cumulative forecast errors. *ShapleyIQ* [138] provides robustness in scenarios with multiple coexisting root causes, as its Shapley value framework inherently considers all combinations of factors, allowing it to accurately quantify the influence of concurrent faults Similarly, *ϵ-Diagnosis* [175] uses distribution-free e-statistics to compare time-series distributions, making it well-suited for the heavy-tailed and high-variance data characteristic of long-tail latency events.

Other methods use filtering techniques to improve the signal-to-noise ratio. *FChain* [161] dynamically adjusts anomaly detection thresholds based on a metric's predictability (via FFT), preventing normal workload fluctuations from being mistaken for faults. Others filter noise at the data source; *SwissLog* [137] achieves robustness to changing log formats through a dictionary-based parser and semantic analysis, while *TraceRCA* [141] employs adaptive feature selection to dynamically ignore irrelevant metrics during an incident. *WinG* [214] uses Dynamic Time Warping (DTW) to compare temporal sequences, providing a non-linear alignment that is resilient to normal variations in service invocation patterns.

6.2.3 Resilience to Novelty and System Dynamics. A critical aspect of robustness is the ability to diagnose novel faults not seen during training and to adapt to dynamic system topologies. Several methods build this resilience into their

Table 4. Overview of Adaptive Learning Paradigms and Representative Works for RCA.

Paradigm	Strategy	Papers
Incremental Model Evolution	Graph-based Incremental Updates	CORAL [190], Sage [101], DGERCL [93]
	Feedback-driven Refinement	DeepHunt [182], IPCCC'16 [162], ServerRCA [177], Chain-of-Event [216], TraceStream [241], UniDiag [230]
	Self-optimization Mechanisms	AutoMAP [155], MS-Rank [153], Medicine [187]
	Online Learning with Memory Mechanisms	OCRCL [121], CloudPD [176], MicroSketch [140]
Rapid Generalization	Few-shot & Zero-shot Learning	Sleuth [102], SpanGraph [132], SparseRCA [217]
Intelligent Policy & Knowledge Adaptation	Reinforcement Learning for Policy Adaptation	TraceDiag [97], ThinkFL [226], HRLHF [194]
	Knowledge Retrieval & Dynamic Reasoning (RAG)	ICLRCA [235], SCELM [185], Xpert [129]

model architecture. *CloudRCA* [237] uses a hierarchical Bayesian network that can generalize to identify the correct fault module even if the specific fault type is new. *MicroIRC* [244] combines a supervised GNN with an unsupervised random walk on a real-time graph, reducing dependency on a static set of failure signatures and making it resilient to both new anomaly types and dynamic instance scaling. Similarly, *UniDiag* [230] is designed to diagnose previously unseen failure classes and can maintain functionality even when certain data modalities are missing, demonstrating resilience to both novelty and incomplete data.

6.2.4 Resilience via Multi-Agent and LLM-based Frameworks. The recent adoption of Large Language Models (LLMs) in RCA has introduced a new source of imperfection: model hallucination and instability. Robust frameworks in this domain focus on constraining and validating the LLM's reasoning process. *Flow-of-Action* [169] and *KnowledgeMind* [172] use expert knowledge, encoded as Standard Operating Procedures (SOPs) or rules, to guide the agent's exploration and provide a reward mechanism, effectively grounding the model and preventing it from pursuing irrelevant diagnostic paths. A complementary approach is to use consensus. *mABC* [233] employs a blockchain-inspired voting mechanism where multiple specialized agents must reach a consensus, preventing a single agent's error from derailing the entire analysis. Similarly, *RCAgent* [200] uses Trajectory-level Self-Consistency (TSC) to aggregate multiple reasoning paths into a more reliable final answer, enhancing robustness even when using less powerful, locally-hosted LLMs.

7 GOAL 3: ADAPTIVE LEARNING

As established in Section 3, the highly dynamic nature of modern systems complicates model stability and generalization, demanding that RCA systems evolve in tandem. This section addresses this challenge through our third goal, **Adaptive Learning**, which enables RCA systems to **continuously adapt to dynamic conditions, novel failures, and evolving system topologies without costly retraining**. Unlike robustness (Goal 2), which handles static data imperfections, adaptive learning focuses on the model's dynamic response to temporal system changes, such as service deployments and workload shifts. We survey three primary adaptation paradigms: **Incremental Model Evolution**, for updating models with new data streams; **Rapid Generalization**, for adapting to new environments with minimal data; and **Intelligent Policy and Knowledge Adaptation**, for dynamically adjusting analytical strategies.

7.1 Incremental Model Evolution

This paradigm focuses on updating existing models with new data streams, ensuring they remain current without discarding previously learned knowledge, a challenge often referred to as catastrophic forgetting. These approaches are crucial for monitoring the gradual shift in system behavior and architecture.

7.1.1 Graph-based Incremental Updates. A prominent strategy involves incrementally updating graph-based models to reflect evolving system dependencies. *CORAL* [190] and *Sage* [101] both address architectural dynamism by updating graph representations. CORAL achieves this by disentangling the causal graph into state-invariant and state-dependent components, allowing for efficient updates on the dynamic part. Its LSTM component captures long-term temporal patterns, while the VGAE locally updates graph structures as services are added, removed, or dependencies modified, significantly reducing computational overhead while maintaining accuracy. Sage, in contrast, decomposes its GVAE model on a per-microservice basis following a Causal Bayesian Network structure, enabling selective partial retraining of only the components affected by a change. This targeted approach reduces retraining time while preserving causal integrity.

Similarly, *DGERCL* [93] employ online learning to process streams of multi-modal data, incrementally updating their graph structures and node embeddings to capture temporal dynamics. DGERCL processes continuous streams of invocation events through an LSTM, dynamically updating microservice node embeddings and employing a self-attention mechanism to weigh the importance of different metrics during the evolution of system behavior.

7.1.2 Feedback-driven Refinement. Another class of methods achieves adaptation through feedback-driven refinement, incorporating human-in-the-loop or automated validation steps to continuously fine-tune the model. *DeepHunt* [182], *IPCCC'16* [162], and *ServerRCA* [177] exemplify this by using operator feedback to correct or confirm diagnoses, which then serves as new labeled data to refine model parameters or knowledge graphs. DeepHunt employs a feedback mechanism to fine-tune its root cause scorer via a ranking-oriented loss function, allowing continuous adaptation to new failures. IPCCC'16 uses a closed-loop approach where operators label causal rules, which are then used to train a Random Forest classifier that updates rule weights in the causality graph. ServerRCA incorporates a human-in-the-loop mechanism specifically for handling unseen fault events, flagging them for expert review and adding newly labeled events to its knowledge repository without full model retraining. *Chain-of-Event* [216] also supports adaptation by enabling the model to be retrained as new labeled incident data accumulates over time, allowing its event-causal graph to evolve with the system. *TraceStream* [241] applies this concept at a cluster level, allowing operators to label groups of anomalous traces efficiently. By utilizing an online data stream clustering algorithm (DenStream), the model continuously updates to handle concept drift caused by system updates, making operator feedback more scalable than per-trace labeling. *UniDiag* [230] supports incremental learning by flagging new failure embeddings that are significantly distant from all known clusters, creating new failure type clusters that can be labeled by operators without requiring a full retraining cycle.

7.1.3 Self-optimization Mechanisms. A more automated variant is self-optimization, where models adjust their internal parameters based on diagnostic performance without explicit human intervention. *AutoMAP* [155] and the two variants of *MS-Rank* [153] dynamically update the weights assigned to different metrics based on their historical success in localizing faults. AutoMAP maintains a history of incidents and calculates similarity to past incidents, using confirmed outcomes to update metric weights for current diagnoses. MS-Rank employs a self-adaptive mechanism that evaluates the precision of each diagnosis result and updates the confidence weights of metrics accordingly, enabling the framework

to optimize itself over multiple incidents. *Medicine* [187] employs a sophisticated form of this through its Multimodal Adaptive Optimization (MAO) module, which dynamically balances the learning rates across different data modalities during training. By evaluating each modality's contribution, it suppresses gradients for high-performing modalities and enhances features for underperforming ones, preventing a dominant data source from suppressing others and ensuring all modalities contribute effectively.

7.1.4 Online Learning with Memory Mechanisms. Some approaches utilize online learning with memory mechanisms to balance stability and plasticity. *OCRCL* [121] uses a memory replay strategy to incrementally train its contrastive learning model on new business incidents without forgetting past knowledge, addressing the scarcity of historical data and enabling real-time model updates in evolving systems. *CloudPD* [176] and *MicroSketch* [140] maintain an adaptive model of normal behavior by continuously updating it with recent data streams. CloudPD employs a k-Nearest Neighbors model on an operating context defined by host metrics, while MicroSketch uses a Robust Random Cut Forest (RRCForest) that processes data as a stream and dynamically adapts to changes such as service auto-scaling or updates without offline retraining.

7.2 Rapid Generalization through Few-shot Learning

Rapid generalization addresses the challenge of quickly adapting to entirely new environments or failure modes with minimal training data. This capability is crucial for organizations deploying RCA systems across diverse microservice environments or when encountering novel failure patterns not seen in historical data.

Sleuth [102] represents a significant advancement over traditional incremental approaches by utilizing Graph Neural Networks (GNNs) to enable few-shot and zero-shot learning. Unlike Sage [101], which relies on GVAE and CBN for incremental updates, Sleuth's GNN architecture captures generalizable patterns across different microservices, reducing the need for extensive retraining. This approach makes Sleuth particularly effective in dynamic environments where rapid deployment and high accuracy are essential, as it can quickly adapt to new microservice applications with minimal data requirements.

SpanGraph [132] demonstrates exceptional performance in few-shot learning scenarios, achieving high F1 scores of 93% and 88.95% on SockShop [45] and Trainticket [49] datasets respectively with just 1% of the training data. The model's performance consistently improved across precision, recall, and F1-score as data proportion increased, highlighting its efficiency and reliability in generalizing with minimal data. This capability makes SpanGraph particularly valuable for fault localization in microservices systems where comprehensive training data is scarce.

SparseRCA [217] is specifically designed for sparse data environments, particularly in testing scenarios, and demonstrates strong adaptability to knowledge obsolescence. It addresses the challenge of frequent system upgrades that lead to constantly emerging new trace structures by estimating expected latency for entirely new patterns through parameter extrapolation from the most similar known structures. This enables the model to perform accurate RCA on previously unencountered trace structures without requiring retraining.

7.3 Intelligent Policy and Knowledge Adaptation

This paradigm moves beyond updating data representations to adapting the analytical *strategy* itself. This is achieved through Reinforcement Learning (RL) or by leveraging the dynamic reasoning and retrieval capabilities of Large Language Models (LLMs).

7.3.1 Reinforcement Learning for Policy Adaptation. RL-based policy adaptation enables systems to learn the optimal sequence of analytical actions. *TraceDiag* [97] exemplifies this approach by using RL to automatically learn adaptive pruning policies for service dependency graphs. The system employs Proximal Policy Optimization (PPO) to learn a policy represented as a filtering tree, which selectively eliminates redundant components based on latency, anomaly indicators, and correlation metrics. This learned policy adapts to changing system characteristics, ensuring that only the most relevant components are retained for subsequent causal analysis. *ThinkFL* [226] advances this paradigm further by using reinforcement fine-tuning to teach a lightweight LLM to autonomously discover optimal reasoning paths. Rather than following a rigid workflow, ThinkFL's "Recursion-of-Thought" actor dynamically decides which data tools to query through a progressive Group Relative Policy Optimization (GRPO) training process. A multi-factor reward function evaluates both the accuracy of the final root cause ranking and the quality of the reasoning path, guiding the LLM to learn interpretable and efficient localization strategies. *HRLHF* [194] integrates human feedback into the reinforcement learning process, drawing inspiration from RLHF techniques used to align large language models. By incorporating expert guidance, HRLHF constructs dependency graphs with high accuracy while minimizing human intervention requirements, effectively learning generalizable patterns for autonomous operation while adapting its analytical approach based on domain expertise.

7.3.2 Knowledge Retrieval and Dynamic Reasoning. A more recent and powerful approach for adaptation is Retrieval-Augmented Generation (RAG) with LLMs. This method avoids model retraining entirely by dynamically retrieving relevant, up-to-date information from an external knowledge base at inference time.

ICLRCA [235], *SCELM* [185], and *Xpert* [129] all leverage this technique by maintaining continuously updated vector databases of historical incidents, their root causes, and associated queries or solutions. When a new incident occurs, the most semantically similar past cases are retrieved and injected into the LLM's context, allowing the model to reason using the most current knowledge without any modification to its internal weights. ICLRCA uses this approach with GPT-4 for automated root cause analysis, outperforming fine-tuned models while avoiding computational costs and data staleness issues. SCELM extends this to unify erroneous change detection, failure triage, and root cause change analysis into a single automated pipeline, processing multimodal data and accessing operational knowledge dynamically. Xpert applies the same paradigm to generate customized domain-specific queries (*e.g.*, KQL) for incident investigation, with the vector database continuously updated as new incident-query pairs are resolved. This RAG-based approach makes the system inherently adaptive to new failure patterns as soon as they are documented in the knowledge base, representing a shift from model adaptation to knowledge adaptation.

8 GOAL 4: REAL-TIME PERFORMANCE

As defined in Section 3, real-time performance is a critical non-functional requirement for practical Root Cause Analysis (RCA) systems. This goal directly addresses the operational imperative to minimize Mean Time to Recovery (MTTR) by ensuring diagnostic completion within seconds or minutes. Achieving this objective requires navigating the fundamental trade-off between analytical depth and response time, a core tension encapsulated in the Effectiveness-Data-Cost Triangle (Section 3). This challenge is exacerbated by the massive data volumes and architectural scale of modern cloud environments. To address this, the literature has converged on three primary strategies, which we review in this section: **(1) Computational Optimization**, which reduces the problem's search space; **(2) Efficient Algorithmic Design**, which employs intrinsically fast algorithms; and **(3) Architectural Acceleration**, which uses system-level parallelism and incremental computation.

Table 5. Overview of Real-time Performance Paradigms and Representative Works for RCA.

Paradigm	Strategy	Papers
Computational Optimization	Heuristic & Statistical Pruning	MicroHECL [147], TraceDiag [97], GIED [116], TraceContrast [224], PatternMatcher [202], Onion [236], COMET [199]
	Dimensionality Reduction & Attribute Selection	RCOAS [94], TS-InvarNet [118], CMMD [212], Squeeze [142]
	Hierarchical & Localized Search	HALO [234], RCD [123], CauseRank [152], DyCause [168]
Efficient Algorithmic Design	Efficient Data Summarization & Representation	MicroSketch [140], KPIRoot [109], Log3C [115]
	Low-Complexity Analytical Algorithms	ShapleyIQ [138], Minesweeper [160], PDiagnose [117], ϵ-Diagnosis [175], FluxRank [149]
	Lightweight Model-Based Approaches	SLIM [173], MonitorRank [131]
Architectural Acceleration	Parallel & Distributed Processing	TraceContrast [224], FacGraph [146], Microscope [145], Sage [101], Murphy [114], CIRCA [135]
	Incremental & Online Learning	CORAL [190], MRCA [198]
	Multi-Stage & Hierarchical Architectures	ChangeRCA [220], CloudPD [176], FChain [161], Roots [125]
	Efficient Integration Frameworks	UniDiag [230], Groot [192], MicroDig [186]
	Specialized Approaches	ModelCoder [87], TraceRank [222], TraceStream [241], GLOBECOM'18 [122]
LLM-Enhanced Efficiency	Context Window & Token Optimization	KnowledgeMind [172], OpenRCA [211], XPERT [183]
	Model Optimization & Fine-Tuning	eARCO [106], ThinkFL [226]

8.1 Computational Optimization via Search Space Reduction

The most prevalent strategy for achieving real-time performance is to aggressively reduce the computational search space. Recognizing that telemetry data is often sparse in failure-related signals, these methods employ sophisticated pruning and filtering techniques to focus analytical resources on the most relevant data subsets.

8.1.1 Heuristic and Statistical Pruning. A primary technique is to eliminate irrelevant data early in the pipeline based on statistical significance or correlation thresholds. *MicroHECL* [147] prunes anomaly propagation chains by assessing the Pearson correlation of metrics between successive service calls, eliminating edges below certain thresholds. *TraceDiag* [97] employs a reinforcement learning agent to learn an optimal pruning policy based on latency, anomaly, and correlation criteria, dramatically reducing the size of the dependency graph before causal analysis. Similarly, *GIED* [116] leverages DBSCAN clustering and influence topology filtering to eliminate nodes with low structural significance, while *TraceContrast* [224] employs chi-square and minimum support pruning to filter out statistically insignificant patterns.

Several systems incorporate multi-stage filtering to progressively narrow the scope of the analysis. *PatternMatcher* [202] uses a lightweight KS-test for initial filtering before pattern classification, while *Onion* [236] implements downward-closure based pruning during clustering to avoid generating trivial log groups. *COMET* [199] employs an AutoExtractor to filter and select the most relevant logs from various sources, creating a concise information subset for downstream analysis.

8.1.2 Dimensionality Reduction and Attribute Selection. Another group of methods focuses on reducing the dimensionality of the search space itself. *RCOAS* [94] introduces an attribute selection pre-processing step that filters irrelevant

dimensions, enabling downstream multi-dimensional analysis algorithms to run orders of magnitude faster (*e.g.*, reducing *HALO*'s execution time from 199.6s to 5.9s). This approach combines rule-based filtering with an improved Logistic Iterative Relief (LIR) algorithm that is robust to data imbalance. *TS-InvarNet* [118] uses shape-based clustering (HDBSCAN) on KPIs to reduce redundancy, significantly decreasing the number of pairwise invariants that need to be mined and monitored.

For multi-dimensional analysis, *CMMD* [212] and *Squeeze* [142] tackle the combinatorial explosion by replacing exhaustive searches with more efficient alternatives. *CMMD* employs a genetic algorithm with an attention-based filtering mechanism to handle search spaces of up to 10^5 dimension value combinations, while *Squeeze* proposes a novel "bottom-up then top-down" search strategy that first clusters potential anomalies and then efficiently searches within clusters using the Generalized Potential Score (GPS) heuristic.

8.1.3 Hierarchical and Localized Search. A more structured approach to pruning involves leveraging hierarchical or causal structures. *HALO* [234] implements a sophisticated two-phase approach that leverages hierarchical attribute relationships, constructing an Attribute Hierarchy Graph (AHG) to organize attributes hierarchically. The system performs an efficient attribute-level search phase followed by a value-level search with adaptive early-stopping mechanisms, enabling it to scale to 1.2 million records while maintaining real-time performance.

RCD [123] achieves significant speedup by avoiding the construction of a complete causal graph. Its hierarchical, divide-and-conquer algorithm performs localized causal discovery on small subsets of metrics, applying the Ψ-PC algorithm to identify potential root causes within each subset and then recursively combining results. This method dramatically reduces the number of required conditional independence tests, making the approach orders of magnitude faster than baseline causal discovery methods and practical for large-scale systems with thousands of metrics. *CauseRank* [152] similarly reduces causal graph complexity by operating on metric groups rather than individual metrics, incorporating domain knowledge to improve both accuracy and efficiency. *DyCause* [168] also accelerates its dynamic causality discovery through optimized pruning strategies and by confining its analysis to specific, detected anomaly intervals, making it significantly faster than baseline methods.

8.2 Efficient Algorithmic Design

Beyond pruning, a second major line of work focuses on designing algorithms and data structures that are intrinsically lightweight and computationally efficient. This approach prioritizes low-overhead computation from the ground up, enabling RCA systems to handle large-scale data without sacrificing speed.

8.2.1 Efficient Data Summarization and Representation. One prominent technique is the use of compact data representations that preserve essential diagnostic information while drastically reducing computational cost. *MicroSketch* [140] employs an extended DDSketch data structure to summarize trace latency distributions with sublinear space and linear time complexity, enabling it to analyze 10,000 traces in approximately 1.1 seconds, which is at least 60 times faster than competing methods. Similarly, *KPIRoot* [109] leverages Symbolic Aggregate Approximation (SAX) to create compact string representations of KPI time-series, allowing for fast similarity and causality computations via Jaccard similarity and Granger causality tests. These symbolic approaches reduce computation time by 56.9% compared to baselines while maintaining diagnostic accuracy. *Log3C* [115] addresses the challenge of massive log volumes through a novel "Cascading Clustering" algorithm that avoids the quadratic complexity of traditional clustering by iteratively sampling, clustering, and matching log sequences, enabling the analysis of terabytes of daily logs in minutes.

8.2.2 Low-Complexity Analytical Algorithms. Several works introduce novel algorithms specifically designed for computational efficiency. *ShapleyIQ* [138] makes the typically exponential-time Shapley value computation feasible for real-time use by introducing a "splitting invariance" property that decomposes the problem, reducing complexity from exponential to $O(nlogn)$. This enables the system to analyze complex requests with hundreds of spans within milliseconds. *Minesweeper* [160] applies the efficient PrefixSpan algorithm for sequential pattern mining on app telemetry traces, completing analysis of tens of thousands of reports in under 3 minutes.

For root cause localization, several systems deliberately avoid computationally expensive operations. *PDiagnose* [117] eschews the costly construction of dependency graphs, instead employing lightweight unsupervised algorithms (KDE and WMA for anomaly detection) combined with a simple voting scheme, achieving polynomial time complexity suitable for real-time diagnosis. *ϵ-Diagnosis* [175] tackles small-window long-tail latency by framing RCA as a two-sample hypothesis test using e-statistics based on energy distance correlation, enabling rapid analysis within seconds even for extremely short time windows. *FluxRank* [149] uses Kernel Density Estimation to quantify KPI changes and employs DBSCAN clustering with Pearson correlation for digest distillation, reducing localization time by over 80% compared to manual approaches.

8.2.3 Lightweight Model-Based Approaches. Efficiency can also be achieved through careful model selection. *SLIM* [173] generates interpretable rule sets to handle imbalanced fault data, using an efficient minorize-maximization approach for rule selection that incurs only about 15% of the training overhead of state-of-the-art deep learning methods while maintaining superior accuracy. *MonitorRank* [131] splits computation into an intensive offline batch-mode engine for call graph generation and pseudo-anomaly clustering, and a lightweight real-time engine that performs a personalized PageRank-style random walk with time complexity O(Nc|V| + |E|), enabling rapid online diagnosis.

8.3 Architectural Acceleration

The third strategy involves architectural patterns that accelerate computation through system design, particularly for large-scale data processing and dynamic environments. These methods focus on how the system is organized to handle load, adapt to change, and leverage available computational resources.

8.3.1 Parallel and Distributed Processing. Parallel processing is essential for handling massive data volumes without sacrificing speed. *TraceContrast* [224] implements its core contrast sequential pattern mining algorithm on Apache Spark, distributing the computational load across a cluster to efficiently process large-scale trace data while maintaining real-time responsiveness. *FacGraph* [146] similarly develops a distributed version of its frequent subgraph mining algorithm using MapReduce, significantly improving performance and scalability. For causal analysis, *Microscope* [145] parallelizes the computationally intensive PC algorithm for causality graph construction, while *Sage* [101] incorporates parallel training capabilities for its Graph Variational Autoencoder (GVAE) and Causal Bayesian Network (CBN) components, enabling the system to handle large-scale microservice architectures while maintaining model accuracy. Both *Murphy* [114] and *CIRCA* [135] incorporate architecture-aware parallelization strategies that adapt to available computational resources and workload characteristics.

8.3.2 Incremental and Online Learning. A critical architectural pattern for maintaining real-time performance in dynamic environments is incremental learning, which avoids costly full-model retraining. *CORAL* [190] is designed for online, near-real-time performance through two main features: an automatic trigger point detection module that initiates analysis early, and an incremental disentangled causal graph learning approach that efficiently updates the

causal graph by decoupling state-invariant and state-dependent information. This enables the system to adapt to new faults without starting from scratch. Similarly, *MRCA* [198] employs reinforcement learning (Q-learning) to dynamically terminate the causal graph construction process, learning an optimal policy to stop expansion when the graph is sufficient for accurate diagnosis, thereby significantly reducing end-to-end analysis time.

8.3.3 Multi-Stage and Hierarchical Architectures. Several systems achieve speed through carefully designed multi-stage pipelines that separate expensive offline computation from lightweight online analysis. *ChangeRCA* [220] is explicitly designed for speed to minimize MTTR, with a multi-stage design starting with a fast check for common canary-related issues. Evaluation shows it can locate 90% of defective changes in under 3 minutes, a 90% reduction compared to baseline approaches. *CloudPD* [176] achieves real-time performance through a layered, two-phase methodology where a computationally inexpensive Event Generation Engine first filters out normal intervals, and a more expensive correlation-based analysis is invoked only for suspicious intervals, enabling diagnosis within tens of seconds. *FChain* [161] similarly completes fault localization within a few seconds through lightweight monitoring and efficient change-point selection, while *Roots* [125] processes data asynchronously and periodically, demonstrating detection and diagnosis within minutes.

8.3.4 Efficient Integration Frameworks. Modern systems increasingly integrate multiple data sources and analytical components through efficient architectural patterns. *UniDiag* [230] achieves an average online diagnosis time of under one second by separating computationally intensive offline training (graph construction, embedding, clustering) from a lightweight online diagnosis phase (embedding and distance comparison). *Groot* [192] demonstrates end-to-end RCA completion in less than 5 seconds by constructing fine-grained event causality graphs with customizable rules and applying a customized PageRank algorithm. *MicroDig* [186] reduces diagnosis time from tens of minutes to under a minute by first identifying a small sub-graph of "association calls" to significantly prune the search space before performing more complex analysis.

8.3.5 Specialized Approaches for Distinct Scenarios. Several approaches target specific diagnostic scenarios with tailored performance optimizations. For multi-dimensional analysis, *ModelCoder* [87] analyzes high-level inter-service call data instead of detailed intra-service events, achieving localization within 80 seconds on average. For trace-based analysis, *TraceRank* [222] combines lightweight spectrum analysis with PageRank-based random walk as a scalable alternative to deep learning models, while *TraceStream* [241] employs lightweight trace embedding (TDTV) and non-iterative centrality-based localization, executing in milliseconds. For change-related diagnosis, active RCA frameworks like those proposed in [122] use Greedy Entropy Minimization (GEM) and Reinforcement Learning to select and execute only the most informative actions in an optimal order, minimizing diagnostic time.

8.4 LLM-Enhanced Efficiency Strategies

The emergence of Large Language Models (LLMs) in RCA has introduced novel efficiency challenges and solutions. These systems must address the unique computational constraints of LLM-based reasoning while maintaining real-time performance.

8.4.1 Context Window and Token Optimization. A primary challenge in LLM-based RCA is managing the massive volume of telemetry data that often exceeds LLM context windows. *KnowledgeMind* [172] addresses token consumption through a service-by-service exploration strategy based on Monte Carlo Tree Search (MCTS), dramatically reducing the amount of information fed into the LLM in a single inference step and making the approach scalable to larger

Table 6. Overview of Interpretability Paradigms and Representative Works for RCA.

Paradigm	Strategy	Papers
Structural Interpretability	Causal Graph Learning from Telemetry	CloudRanger [196], LOUD [157], Sieve [189], CauseRank [152], RUN [144], AERCA [111], DyCause [168], CIRCA [135], FlowRCA [205], RCSF [193]
	Incorporating Domain Knowledge	GrayScope [231], TS-InvarNet [118], HRLHF [194], Atlas [209], RealTCD [136]
	Enriching Graph Semantics	MicroDig [186], REASON [191], ICWS'17 [127], Murphy [114], Chain-of-Event [216]
Semantic Interpretability	Natural Language Report Generation	RCACopilot [91], SCELM [185], COCA [139], SynergyRCA [206], LM-PACE [225]
	Interpretable Reasoning Process	Roy et al. [174], OpenRCA [211], Flow-of-Action [169], KnowledgeMind [172], TrioXpert [183], mABC [233]
	High-Level Abstractions	Minesweeper [160], COMET [199], DéjàVu [143]
Evidence-based & Rule-based Interpretability	Attribution to Evidence	Nezha [221], LoFI [120], FluxRank [149], ART [184], TraceAnomaly [150], PDiagnose [117]
	Quantitative Attribution	ShapleyIQ [138], CD-RCA [218], LADRA [151], DeepHunt [182], GAMMA [179]
	Explicit Rule Generation	SLIM [173], CMDiagnostor [223], PatternMatcher [202], KPIRoot [109], Graphbasedrca [86], Roots [125], DiagMLP [103]
Interactive Interpretability	Visual Exploration	Zhou et al. [242], Groot [192]
	Hypothesis-driven Investigation	EXPLAINIT! [126], TraceDiag [97], ThinkFL [226]

microservice systems. *OpenRCA* [211] introduces an RCA-agent architecture that uses code execution for data processing, avoiding costly token consumption by having the LLM generate and execute Python code to analyze telemetry data programmatically rather than processing raw data in the context window. *XPERT* [183] significantly reduces end-to-end latency by automating query authoring, generating domain-specific language queries in seconds through in-context learning, thereby eliminating the time-consuming manual query construction process.

8.4.2 Model Optimization and Fine-Tuning. Another line of work focuses on making LLM-based RCA cost-effective through model optimization. *eARCO* [106] focuses on improving efficiency and cost-effectiveness by automatically optimizing prompts using PromptWizard and demonstrating that fine-tuned Small Language Models (SLMs), when paired with optimized prompts, can serve as a computationally efficient alternative to large expensive LLMs, reducing inference costs while maintaining performance. *ThinkFL* [226] uses a lightweight LLM backbone (<10B parameters) with an efficient "Recursion-of-Thought" reasoning framework and progressive reinforcement fine-tuning, reducing end-to-end localization latency from minutes to seconds and making LLM-based RCA practical for production environments.

9 GOAL 5: INTERPRETABILITY

As established in Section 3, a core objective of RCA is not only to identify the root cause (the "what") but also to explain the failure propagation path (the "how" and "why"). The goal of interpretability is to make these diagnostic results understandable, trustworthy, and verifiable for human operators. This section organizes existing approaches by the *strategy* they employ to achieve this goal. We identify four primary strategies: 1) **Structural Interpretability**, which directly materializes the propagation graph; 2) **Semantic Interpretability**, which translates findings into human-readable narratives; 3) **Evidence-based and Rule-based Interpretability**, which exposes the underlying logic; and 4) **Interactive Interpretability**, which facilitates human-led exploration of the results.

9.1 Structural Interpretability through Causal Graph Construction

Structural interpretability aims to construct an explicit model of failure propagation, typically in the form of a graph. This approach directly addresses the challenge of explaining the "how" and "why" of an incident by visualizing the causal chain from the root cause to the observed symptoms. The graph itself becomes the explanation, providing a logical and verifiable narrative for operators.

A primary strategy in this area is to learn a causal graph from system telemetry. Many methods employ statistical techniques, such as Granger causality, on performance metrics to infer a directed graph representing influence or dependency. For instance, *CloudRanger* [196], *LOUD* [157], *Sieve* [189], and *CauseRank* [152] all construct causal graphs from metrics and then apply ranking algorithms to pinpoint the most central nodes in the failure propagation. *RUN* [144] and *AERCA* [111] specifically adapt neural Granger causality for this task, capturing more complex temporal dependencies. *DyCause* [168] further extends this to discover time-varying causalities, showing how relationships evolve during an incident. Other methods like *CIRCA* [135] and *FlowRCA* [205] ground their graph construction in formal causal inference theory, identifying root causes as "interventions" that break the learned normal causal relationships.

To improve the accuracy and plausibility of these graphs, some approaches incorporate domain knowledge. *GrayScope* [231] and *TS-InvarNet* [118] refine data-driven graphs by starting with an expert-defined "causality skeleton." *HRLHF* [194] uses reinforcement learning to actively query human experts, efficiently integrating their knowledge into the graph discovery process. *Atlas* [209] and *RealTCD* [136] leverage Large Language Models (LLMs) to parse documentation and generate a high-quality prior causal structure.

Other approaches focus on enriching the graph's semantics. *MicroDig* [186] constructs a heterogeneous graph distinguishing between services and calls, while *REASON* [191] models interdependent networks across system layers (*e.g.*, pods and servers). *ICWS'17* [127] builds a two-layer graph modeling both inter-service topology and intra-service control flow. *Murphy* [114] uses a Markov Random Field to handle cyclic dependencies, which are common in real systems but problematic for many causal discovery algorithms. *Chain-of-Event* [216] automatically learns a weighted event-causal graph where parameters have intuitive physical meanings, allowing engineers to inspect the model's reasoning. Finally, *RCSF* [193] focuses on generating fault propagation sequences even with incomplete monitoring coverage.

9.2 Semantic Interpretability through Natural Language and High-Level Concepts

Semantic interpretability focuses on translating technical findings into human-readable narratives or high-level concepts. This strategy has gained significant traction with the advent of LLMs, which excel at synthesizing complex information into coherent text and classifying issues into understandable categories.

A prominent application of LLMs is generating natural language reports. *RCACopilot* [91] and *SCELM* [185] analyze diagnostic data to produce structured reports that summarize the incident, identify the root cause, and suggest solutions. To ground the LLM's generation in factual data, many approaches adopt Retrieval-Augmented Generation (RAG). *COCA* [139] retrieves relevant source code, while *SynergyRCA* [206] queries a real-time "StateGraph" of the system. To enhance trust, *LM-PACE* [225] provides a calibrated confidence score for the LLM's output.

Another line of work focuses on making the LLM's reasoning process itself interpretable. Frameworks like *Roy et al.* [174] and *OpenRCA* [211] use a "Chain-of-Thought" or "ReAct" paradigm, where the LLM externalizes its diagnostic steps. Multi-agent systems like *Flow-of-Action* [169], *KnowledgeMind* [172], *TrioXpert* [183], and *mABC* [233] structure

this reasoning process further, assigning specific sub-tasks to specialized agents to make the overall analysis more robust and transparent.

Beyond LLMs, other methods provide semantic meaning through high-level abstractions. *Minesweeper* [160] discovers and ranks sequential event patterns (*e.g.*, "PlayVideo -> DeleteStory") that are distinctive to buggy sessions, providing a clear narrative of user actions leading to a failure. *COMET* [199] uses an LLM to extract keywords from logs, which helps engineers quickly understand the nature of an incident. *DéjàVu* [143] provides interpretability by classifying a failure into a known category and retrieving a similar historical incident, explaining the current problem by analogy.

9.3 Evidence-based and Rule-based Interpretability

This category of methods achieves interpretability by making the logic of the diagnosis explicit, either by exposing the underlying evidence or by presenting the conclusion as a set of human-readable rules. This approach builds trust by allowing operators to understand and verify the "why" behind a conclusion.

Several methods provide interpretability by attributing a finding to specific, understandable evidence. *Nezha* [221] explains a failure by showing the deviation between "expected" and "actual" execution patterns. *LoFI* [120] extracts specific fault-indicating phrases from logs, directly answering "what went wrong." *FluxRank* [149] distills thousands of alerts into a few "digests" (*e.g.*, "27 machines in module M1 experienced CPU overload"). *ART* [184] uses a "unified failure representation" where each feature dimension directly corresponds to an original data channel, showing which signals are deviating. *TraceAnomaly* [150] uses a handcrafted vector where each dimension represents a specific (service, callpath) tuple, making the source of deviation clear. *PDiagnose* [117] provides concrete evidence by outputting the specific KPI names and raw log entries that are anomalous.

Quantitative attribution is another powerful technique. *ShapleyIQ* [138] and *CD-RCA* [218] use Shapley values to quantify precisely how much each component contributed to the failure. *LADRA* [151] provides a probabilistic diagnosis of resource contention (CPU, memory, etc.) to explain why a task is slow. *DeepHunt* [182] calculates an interpretable score based on an instance's own anomaly and the anomalies of its neighbors in the propagation path. *GAMMA* [179] uses feature-omission studies to explain the type of bottleneck (*e.g.*, CPU-bound) by observing performance drops when certain metrics are excluded.

Finally, some methods generate explicit, human-readable rules. *SLIM* [173] produces a set of decision rules in Disjunctive Normal Form (*e.g.*, `IF cpu_usage > 80 THEN fault`). *CMDiagnostor* [223] uses rule-based pruning and ranking keys, providing a clear rationale for its choices. *PatternMatcher* [202] classifies anomalies into physically meaningful patterns before ranking, making the reasoning transparent. *KPIRoot* [109] uses a two-factor logic ("it looks similar and it happened first") that is intuitive for operators. *Graphbasedrca* [86] matches an anomaly to a library of pre-labeled, human-understandable patterns. *Roots* [125] employs a combination of statistical methods to identify the bottleneck, justifying its approach through a majority vote. The study *DiagMLP* [103] provides interpretability for an entire class of models by showing that their performance stems from data fusion rather than complex GNNs, clarifying the true drivers of success.

9.4 Interactive Interpretability through Visual and Exploratory Interfaces

Interactive interpretability empowers operators by providing tools to visually explore data, test hypotheses, and engage directly with the findings. This approach facilitates a dialogue between the operator and the system, where the operator's domain knowledge can guide the investigation.

Table 7. Overview of Multi-Granularity Paradigms and Representative Works for RCA.

Paradigm	Strategy	Papers
Hierarchical Drill-Down	From Service to Infrastructure/Instance	FAMOS [98], FaaSRCA [119], HALO [234], KnowledgeMind [172], MicroIRC [244], REASON [191], SwissLog [137]
	From Service to Component/Metric	AERCA [111], CausalRCA [210], CauseInfer [89], CloudRCA [237], CMDiagnostor [223], FlowRCA [205], HeMiRCA [245], ICSOC'20 [203], LatentScope [208], MRCA [198]
	From Service to Code/Change	ChangeRCA [220], COCA [139], LogFaultFlagger [85], MEPFL [243], Nezha [221], Raccoon [238], ServerRCA [177], TrinityRCL [108]
Multi-Dimensional & Fine-Grained Localization	Multi-Attribute Pattern Mining	TraceContrast [224], TVDiag [207]
	Operation & Span-Level Analysis	faultstudy [242], SpanGraph [132], TraceNet [213]

Visual exploration of system behavior is a cornerstone of this approach. The empirical study in *Zhou et al.* [242] validates this, showing that visual trace analysis significantly helps developers understand fault propagation. Systems like *Groot* [192] provide interactive interfaces that allow operators to click on graph nodes for details, filter the view, and trace failure paths, transforming an abstract graph into a concrete investigative tool.

Other systems facilitate hypothesis-driven investigation. *EXPLAINIT!* [126] offers a declarative, SQL-like interface for operators to formulate and test causal hypotheses against time-series data. This allows operators to leverage their domain expertise to guide the analysis. *TraceDiag* [97] learns an interpretable "filtering tree" (a form of decision tree) that explains its reasoning for pruning the search space, allowing engineers to understand the system's focus. *ThinkFL* [226] uses a "Recursion-of-Thought" mechanism that allows an LLM to dynamically query data tools, with the entire reasoning path being transparent and verifiable by SREs. By turning RCA into an interactive and exploratory process, these methods bridge the gap between automated analysis and human-led problem-solving.

10 GOAL 6: MULTI-GRANULARITY

As established in our formalization (Section 3), the primary objective of most RCA systems is to identify the root cause event node(s) r within the incident propagation graph $\mathcal{G}$. The goal of multi-granularity directly addresses the challenge of localizing this root cause at varying levels of abstraction. This capability is essential because different roles in the incident management lifecycle require diagnoses at different depths. Site Reliability Engineers (SREs) need coarse-grained localization for rapid mitigation (reducing MTTR), while developers require fine-grained analysis to implement permanent fixes (improving MTBF). The achievable precision of the output is fundamentally constrained by the granularity of the input observation space $\mathcal{O}$. This section, therefore, reviews approaches based on their ability to provide outputs at hierarchical levels of abstraction, from the service level down to the code.

10.1 Hierarchical Granularity Levels in RCA

The granularity of root cause identification is fundamentally constrained by two factors: the granularity of the input observational data and the effectiveness of the inference method in utilizing that data. As illustrated in our formalization (Section3), the observation space $\mathcal{O}$ contains telemetry data at various abstraction levels, and the analysis can only be as fine-grained as the most detailed available data permits.

The pursuit of multi-granularity in RCA is driven by the diverse needs of different roles within the incident management lifecycle. While SREs often require rapid, coarse-grained localization (*e.g.*, identifying a faulty service or instance) to facilitate immediate mitigation, developers need fine-grained, deep localization (*e.g.*, pinpointing a specific

metric, code change, or function) to implement permanent fixes. An effective RCA system must therefore provide outputs at multiple, hierarchical levels of abstraction. We review approaches based on the depth and precision of their localization capabilities, categorizing them by their ability to drill down from the service level to the infrastructure, component, and code levels.

10.1.1 From Service to Infrastructure and Component Granularity. A primary objective in multi-granularity RCA is to bridge the gap between service-level symptoms and their underlying causes within the infrastructure or application components. Several approaches achieve this by constructing hierarchical models that explicitly connect different system layers.

Two-Stage Top-Down Localization. A common strategy involves a two-stage, top-down localization process that first identifies a faulty service and then drills down to pinpoint a more specific root cause. *CauseInfer* [89] employs a two-layered hierarchical causality graph: a coarse-grained service dependency graph constructed by analyzing network traffic delays, and for each service, a fine-grained metric causality graph built using the PC-algorithm. When an SLO violation occurs, the system first traverses the service graph to localize the faulty service, then traverses the metric graph using depth-first search with CUSUM-based change detection to identify anomalous root cause metrics. Similarly, the method proposed in ICSOC'20 [203] constructs a service dependency graph and uses Personalized PageRank to identify potential culprit services, then applies an autoencoder-based model trained on normal data to analyze reconstruction errors of live metrics, pinpointing the root cause at the metric-level within each candidate service. *MRCA* [198] extends this paradigm by fusing features from logs and traces for more accurate anomaly detection and initial ranking of abnormal services, then performing fine-grained causal analysis using Granger causality on the metrics of top-ranked services, with a Q-learning agent dynamically terminating graph expansion to balance accuracy and speed. *KnowledgeMind* [172] employs a Monte Carlo Tree Search (MCTS) process guided by LLMs to identify the faulty service through service-by-service reasoning, followed by a dedicated Service-Pod Agent that drills down to the specific faulty pod.

Several methods refine the metric-level localization through sophisticated anomaly correlation techniques. *HeMiRCA* [245] leverages the monotonic correlation between heterogeneous monitoring data by constructing span vectors from traces to represent invocation latency, using a Variational Autoencoder (VAE) to compute anomaly scores, and calculating Spearman rank correlation with individual metric time series to rank suspicious metrics and their corresponding services. *CausalRCA* [210] applies gradient-based causal structure learning (DAG-GNN) to build a weighted directed acyclic graph representing causal dependencies between metrics, then applies PageRank to identify root causes at both service and metric levels without strict distributional assumptions. *FlowRCA* [205] constructs a metric-level causality graph using normalizing flows to infer causal direction and quantifies causal impacts using Conditional Average Treatment Effect (CATE), applying Personalized PageRank on the inverted anomalous subgraph to rank metrics. *AERCA* [111] uses an autoencoder-based framework integrating Granger causal discovery, defining anomalies as interventions on exogenous variables to identify not only the root-cause time series but also the specific time steps of the intervention.

Unified Multi-Layer Graph Models. Other approaches build unified, multi-layered graphs that inherently represent the system's hierarchical structure. *HALO* [234] automatically learns an Attribute Hierarchy Graph (such as Node → Cluster → Datacenter) by analyzing pairwise conditional entropy between attributes, then uses a failure-aware random walk to generate promising search paths and performs a self-adaptive top-down search with OTSU-based pruning to find the optimal fault-indicating attribute-value combination at the appropriate granularity. *REASON* [191] models the system as an interdependent network of high-level servers and low-level pods, employing a hierarchical

Graph Neural Network (GNN) to learn intra-level and inter-level causal relationships, combining topological causal discovery with individual causal discovery based on Extreme Value Theory. *FaaSRCA* [119] constructs a "Global Call Graph" that integrates multi-modal observability data from both application functions and platform components (such as Kubernetes pods), using an unsupervised Graph Attention Network (GAT) based autoencoder trained on normal operations to compare reconstruction errors and identify root causes across the full serverless lifecycle. *MicroIRC* [244] builds a Heterogeneous Weighted Topology (HWT) graph of services, instances, and hosts, running a personalized random walk to generate root cause candidates and feeding them with real-time metrics into a pre-trained GNN model (MetricSage) to produce a final ranked list at the instance level. *SwissLog* [137] provides multi-granularity localization by first detecting anomalies at the execution instance level from interleaved logs. It then uses a pre-constructed ID relation graph, which maps dependencies between different entity types (such as application, container, and block), to pinpoint the specific anomalous instance, allowing operators to drill down from a system-wide issue to a fine-grained root cause.

Specialized Domain Integration. Some methods achieve multi-granularity through domain-specific integration strategies. *CMDiagnostor* [223] operates across multiple granularities by ingesting fine-grained, method-level Call Metric Data and constructing an ambiguity-free call graph using a novel regression-based method (AmSitor) to resolve upstream-downstream call correspondences, ultimately outputting a ranked list of coarse-grained services. *CloudRCA* [237] integrates heterogeneous data sources (including KPIs, logs, and topology) into a Knowledge-informed Hierarchical Bayesian Network (KHBN) with a hierarchical root cause layer, enabling the model to pinpoint both the high-level faulty module and the specific, low-level fault type. *FAMOS* [98] collects metrics from both host and container levels and correlates them with service-level traces using a late-fusion paradigm with Gaussian-attention and cross-attention mechanisms, enabling it to identify fine-grained root cause types such as "Host CPU overload" or "Container stopped". *LatentScope* [208] models heterogeneous root cause candidates (including services, pods, hosts, databases, and software changes) as latent variables in a dual-space graph, using a Regression-based Latent-space Intervention Recognition (RLIR) algorithm to infer anomalous latent variables even with limited observability.

10.1.2 From Service to Code-Level Granularity. The ultimate goal for many RCA systems is to provide code-level localization, directly guiding developers to the source of a fault. This requires sophisticated techniques that can connect high-level system behavior to specific code artifacts.

Direct Code Integration. Several methods achieve code-level localization by integrating code-related information directly into their analysis. *TrinityRCL* [108] constructs a heterogeneous causal graph containing services, hosts, metrics, and faults (which represent code exceptions extracted from logs), assigning anomaly scores to edges using correlation algorithms (such as DTW and CORT) and using Random Walk with Restart (RWR) to simulate anomaly propagation and identify root causes across application, service, host, metric, and code levels. *Nezha* [221] transforms multi-modal data (including metrics, logs, and traces) into a unified stream of events structured into event graphs, comparing the frequency of event patterns (which are represented as subgraphs) between normal and faulty periods to identify deviating patterns that correspond to code regions or resource types, providing interpretable fine-grained root cause candidates. *COCA* [139] leverages LLMs to analyze issue reports, using static analysis-based backtracking to link log messages to source code locations, building a call graph patched with a novel RPC bridging method to reconstruct execution paths, and combining issue reports with retrieved code snippets for root cause inference at the class and method level.

Software Change Identification. Another category focuses on identifying specific software changes as the root cause. *ChangeRCA* [220] refines service-level RCA output through a three-stage framework: using cascaded Difference-in-Differences (DiD) to detect faulty canary releases, filtering non-change-related faults, and scoring recent changes by integrating KPIs, service dependency graphs, and change timing to pinpoint the specific defective software change. *Raccoon* [238] bridges the semantic gap between user-reported incidents and software changes by representing incidents at a user-perceived functional level using Fault Trees and Software Product Lines, mining causal knowledge with a Tree GNN to build a knowledge base linking incidents to changes, and recommending root-cause changes at multiple granularities (ranging from product line to specific change).

Test Log and Trace Analysis. Methods focusing on test environments and operational traces also achieve fine-grained localization. *LogFaultFlagger* [85] localizes faults from entire test log files down to specific log lines by calculating a score combining line-level Inverse Document Frequency (line-IDF) with historical fault association, using an Exclusive K-Nearest Neighbors (EKNN) algorithm to predict product faults and flag the most probable cause lines. *MEPFL* [243] learns from system trace logs to predict latent errors and locate faulty microservices, extracting comprehensive features at both trace-level and microservice-level to train machine learning models that identify the fault type and responsible service. *ServerRCA* [177] employs a hierarchical matching framework using contrastive learning on operating system logs, analyzing at three levels (namely Fault, Module, and Event) with BERT-based encoders to drill down from general symptoms to specific, actionable fault events, constructing a fault propagation chain via a knowledge graph.

10.1.3 Multi-Dimensional and Fine-Grained Localization. Beyond simple hierarchical localization, some advanced methods provide multi-dimensional analysis, identifying root causes as a combination of factors across different system dimensions.

Multi-Attribute Pattern Mining. *TraceContrast* [224] frames RCA as a contrast sequential pattern mining problem, representing traces as sequences of attribute sets (including service version, API route, and OS version) and applying a parallel contrast sequential pattern mining algorithm to find patterns frequent in anomalous paths but rare in normal ones, ranked using spectrum analysis (Ochiai) to provide precise, hierarchical diagnosis. *TVDiag* [207] builds an instance correlation graph and employs a GNN-based multimodal co-learning module with task-oriented supervised contrastive learning and cross-modal contrastive learning to simultaneously perform root cause localization (identifying the faulty instance) and failure type identification, serving both localization and classification needs.

Operation and Span-Level Analysis. Methods that refine the definition of system entities achieve even finer granularity. *TraceNet* [213] constructs a Service Dependency Graph at the operation level (which represents specific API endpoints) rather than the service level, quantifying microservice abnormality by distinguishing between inner-abnormality and outer-abnormality to handle propagation effects, enabling it to differentiate between business functions within a single microservice. *SpanGraph* [132] operates at the span level by constructing a directed graph where nodes represent unique microservice requests (characterized by NodeId, InstanceId, ServiceName, and ApiName) and edges represent invocations, using a Graph Convolutional Network (GCN) to classify edges as normal or anomalous and localizing faults to the starting node of anomalous edges. The empirical study presented in the fault analysis work [242] proposes two distinct trace visualization strategies: "Microservice as Node" (which provides service-level view) and "Microservice State as Node" (which provides state-level view), enabling developers to analyze failures at different abstraction levels.

Table 8. Overview of Actionability Paradigms and Representative Works for RCA.

Paradigm	Strategy	Papers
Direct Remediation Generation	Automated Actuation & Resource Management	CloudPD [176], Sage [101]
	LLM-Driven Remediation Planning	SynergyRCA [206], SCELM [185], mABC [233], Ahmed et al. [84]
	Failure Classification & Knowledge-Based Remediation	Déjàvu [143], MEPFL [243], AutoMAP [155], LogKG [180]
Automated Responsibility Assignment	Automated Triage & Escalation	RCACopilot [91], COMET [199], RCAgent [200]
Actionable Knowledge Provision	Precision Diagnostic Artifacts	DéjàVu [143], Xpert [183], Ikeuchi et al. [122]
	Contextual Knowledge Retrieval	RCAgent [200], Roy et al. [174], RCACopilot [91], ICLRCA [235], SynergyRCA [206]
	Human-in-the-Loop Validation & Trust Calibration	Groot [192], HRLHF [194], Chain-of-Event [216], TraceDiag [97], Nezha [221], LM-PACE [225], GMTA [110], Eadro [133]

11 GOAL 7: ACTIONABILITY

As established in Section 3, the core output of the RCA function is the incident propagation graph $\mathcal{G}$, which provides the diagnostic explanation of a failure. However, a diagnosis alone is insufficient for effective incident management; it must be translated into concrete remedial actions. Actionability addresses this need by focusing on translating diagnostic outputs into concrete operational directives to guide swift remediation. This section examines how research transforms RCA into a prescriptive tool, organizing the discussion around three dimensions: direct remediation generation (Section 11.1), automated responsibility assignment (Section 11.2), and actionable knowledge provision (Section 11.3).

11.1 Direct Remediation Generation

The most direct form of actionability is achieved by systems that automatically generate or trigger specific repair actions, thereby closing the loop between detection and resolution with minimal human intervention.

11.1.1 Automated Actuation and Resource Management. Early work in this area integrated diagnostic frameworks with actuation controllers to enable automated corrective actions. *CloudPD* [176] and *Sage* [101] exemplify this approach by implementing closed-loop systems that, upon identifying faults such as resource contention or performance bottlenecks, automatically trigger remedial operations. These operations include VM reconfiguration, CPU frequency scaling, resource re-partitioning, or microservice scaling to restore quality of service. By directly coupling diagnosis with actuation, these systems significantly reduce Mean Time to Recovery (MTTR) and minimize the need for manual operator intervention during critical incidents.

11.1.2 LLM-Driven Remediation Planning. The advent of Large Language Models has enabled a new paradigm of context-aware, nuanced remediation generation. Rather than relying on predefined action templates, LLM-based systems can generate tailored repair plans that account for incident-specific context and organizational practices.

SynergyRCA [206] demonstrates this capability in Kubernetes environments by producing not only diagnostic reports but also precise, executable commands (*e.g.*, `kubectl patch`) tailored to the specific incident. This provides operators with validated, one-step solutions. Similarly, *SCELM* [185] and *mABC* [233] incorporate dedicated solution generation modules or specialized agents (*e.g.*,"Solution Engineer" agents) that formulate explicit resolution steps as integral components of their analytical output. The foundational work by Ahmed et al. [84] provided large-scale empirical

evidence for this approach, demonstrating through evaluation on over 40,000 Microsoft incidents that fine-tuned LLMs can effectively recommend concrete mitigation steps alongside root causes, with their utility validated by human incident owners.

11.1.3 Failure Classification and Knowledge-Based Remediation. An alternative approach enhances actionability through failure type classification coupled with knowledge-based remediation databases. Systems such as *Déjàvu* [143] and *MEPFL* [243] maintain taxonomies of failure categories (*e.g.*, configuration-related, resource-related, bad requests, slow queries), with each category mapped to established remediation workflows. This categorical approach enables immediate guidance on appropriate response measures and is particularly valuable in large organizations where different failure types require expertise from distinct teams or follow different escalation procedures. However, the effectiveness of this approach depends critically on the comprehensiveness and adaptability of the underlying taxonomy to organization-specific contexts and evolving architectures.

A complementary technique leverages historical incident databases through case-based reasoning. Methods such as *Déjàvu* [143], *AutoMAP* [155], and *LogKG* [180] retrieve similar historical incidents based on symptom patterns and failure characteristics, surfacing not only technical details but also the remediation actions that were successfully applied, their effectiveness, and lessons learned. This approach provides battle-tested solutions validated in production environments and captures organizational knowledge that may not be formally documented. However, systems must account for architectural changes and software version updates that may invalidate past solutions.

11.2 Automated Responsibility Assignment

In large, complex organizations, knowing *what* to do is insufficient without determining *who* should do it. Misdirected incidents lead to significant delays as tickets are manually rerouted between teams. Automated responsibility assignment addresses this challenge by routing incidents to appropriate personnel based on diagnosed root causes and organizational context.

RCACopilot [91] and *COMET* [199] directly tackle this problem through automated triage systems. These frameworks analyze incident data to identify the appropriate on-call engineers or teams responsible for affected components. *RCACopilot* maintains comprehensive mappings between system components, failure types, and responsible teams, enabling nuanced routing where, for instance, a memory leak is directed to the development team while resource exhaustion in the same service escalates to the infrastructure team. *COMET* employs LLMs to extract keywords from incident data and matches them to historical incidents to determine team ownership, demonstrating a 30% improvement in triage accuracy and 35% reduction in Time to Mitigation at Microsoft.

Advanced systems implement dynamic escalation paths that adapt based on incident severity, resolution progress, and team availability. If no progress occurs within defined time windows, incidents automatically escalate to senior engineers or involve additional teams. *RCAgent* [200] integrates responsibility assignment as a core feature of its autonomous agent framework, making it practical for DevOps workflows. The primary challenge lies in maintaining accurate mappings between technical components and organizational responsibilities in rapidly evolving environments, requiring either continuous manual maintenance or automated inference from operational data and organizational patterns.

11.3 Actionable Knowledge Provision

Beyond fully automated repairs or assignments, another line of research focuses on empowering human operators by making the diagnostic process itself more action-oriented. This is achieved by providing actionable knowledge or intermediate artifacts that accelerate subsequent manual investigation and resolution steps.

11.3.1 Precision Diagnostic Artifacts. One approach refines the diagnostic output to have clear operational implications. Rather than identifying only a faulty component (too coarse) or individual metrics (too fine-grained), systems can provide actionable diagnostic units that directly suggest remediation classes. *DéjàVu* [143] exemplifies this by identifying a "failure unit"—a combination of a faulty component and an indicative metric group (*e.g.*, 'high memory usage' on a specific Docker container). This output immediately suggests categories of remedial actions (*e.g.*, investigate memory leaks, increase resource limits), bridging the gap between diagnosis and mitigation without requiring extensive interpretation.

Another powerful technique equips operators with precise investigation tools. *Xpert* [183] operationalizes this by using an LLM to translate natural language incident descriptions into precise, executable Kusto Query Language (KQL) queries. This transforms a passive ticket into an active investigation, allowing engineers to immediately retrieve the necessary telemetry data without the cognitive load and potential errors of manual query construction. Similarly, the active probing mechanism proposed by Ikeuchi et al. [122] generates more discriminative diagnostic data through targeted user action execution, enabling operators to perform precise, low-impact fixes (*e.g.*, restarting a single process) instead of resorting to broad, disruptive actions (*e.g.*, rebooting an entire device).

11.3.2 Contextual Knowledge Retrieval. Agent-based systems leverage tool-use and knowledge retrieval to ground their analysis in validated operational wisdom. Frameworks such as *RCAgent* [200] and the LLM-based agent evaluated by Roy et al. [174] interact with live diagnostic services, knowledge base articles (KBAs), and historical incident databases. By retrieving proven solutions from past incidents or internal documentation, these systems ground their recommendations in battle-tested practices rather than generic heuristics. *RCACopilot* [91] extends this by generating detailed explanatory narratives that bridge knowledge gaps for on-call engineers unfamiliar with specific components, providing not just technical findings but also business context and relationships.

Several recent approaches employ retrieval-augmented generation (RAG) to combine LLM capabilities with historical incident knowledge. *ICLRCA* [235] implements a standard RAG pipeline that retrieves similar historical cases and uses them to prompt LLMs for contextualized root cause analysis. *SynergyRCA* [206] constructs a graph-based RAG system using StateGraphs and MetaGraphs to capture runtime spatial-temporal relationships in Kubernetes, enabling more precise context retrieval. These approaches ensure that generated recommendations are grounded in actual operational experience rather than generic knowledge, though they face challenges related to data quality, privacy, and the representativeness of historical cases in rapidly evolving environments.

11.3.3 Human-in-the-Loop Validation and Trust Calibration. While automation accelerates incident response, human expertise remains essential for handling novel failures and validating high-stakes decisions. Several systems incorporate mechanisms for expert feedback integration and confidence assessment to enhance the trustworthiness of actionable outputs.

Systems like *Groot* [192], *HRLHF* [194], and *Chain-of-Event* [216] enable human experts to inject domain knowledge by refining event relationships, validating causal dependencies, or adjusting analysis parameters. *HRLHF* [194] and *Chain-of-Event* [216] specifically leverage human feedback to refine the causal graphs generated by automated systems, allowing experts to correct erroneous edges and confirm valid causal paths, thereby improving the accuracy of the final

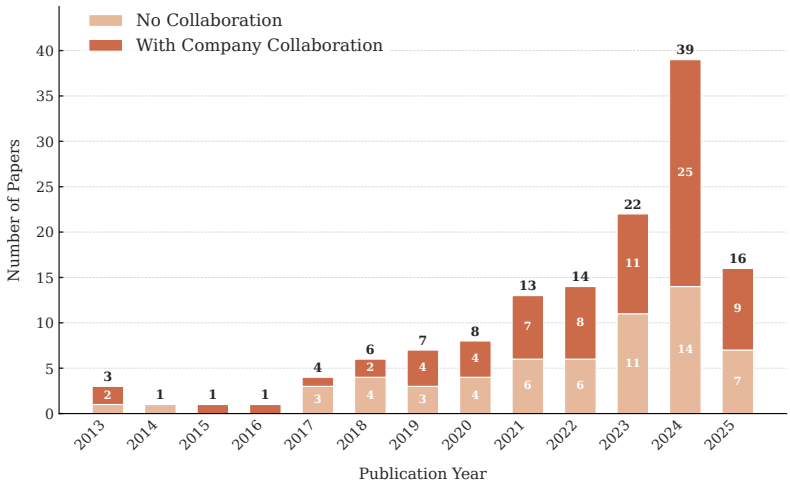

Fig. 5. The number of collected papers published per year in top-tier venues, distinguishing between those with and without company collaboration.

diagnosis. *TraceDiag* [97] learns from historical expert decisions through reinforcement learning, capturing pruning policies derived from experienced engineers without requiring real-time expert input. This balance between automation and expert oversight is critical, as purely manual configuration creates operational overhead while purely automated systems may lack organization-specific context.

A critical aspect of actionability is helping operators understand when to trust automated recommendations. *Nezha* [221] provides ranked suspicion lists with confidence measures, allowing experts to apply their judgment rather than blindly following system outputs. *LM-PACE* [225] implements a sophisticated two-stage prompting approach that quantifies evidence strength from historical incidents and provides calibrated confidence scores. These mechanisms enable more informed decision-making about when to follow automated guidance versus seeking additional expert input, though the challenge remains in developing confidence metrics that accurately reflect system reliability while remaining interpretable under operational pressure.

Interactive visualization interfaces further enhance actionability by presenting complex dependency relationships in comprehensible formats. Systems such as *GMTA* [110], *Eadro* [133], and *Groot* [192] provide user-friendly interfaces that help operators quickly validate findings and identify remediation actions. The design challenge lies in presenting sufficient detail for expert validation while maintaining clarity and usability during high-stress incident response.

12 RESEARCH TREND AND DISTRIBUTIONS

In this section, we analyze the research trend and distributions of root cause analysis through the lens of our seven-goal taxonomy (defined in Section 3). We examine how the field has evolved in terms of publication volume, research focus, and the diversity of problem settings addressed by the community.

12.1 Publication Trend and Research Focus Evolution

We first analyze the publication trend of RCA research in top-tier venues. As shown in Fig. 5, the data reveal a clear upward trend in the number of publications over the past decade. Particularly, the field has experienced significant

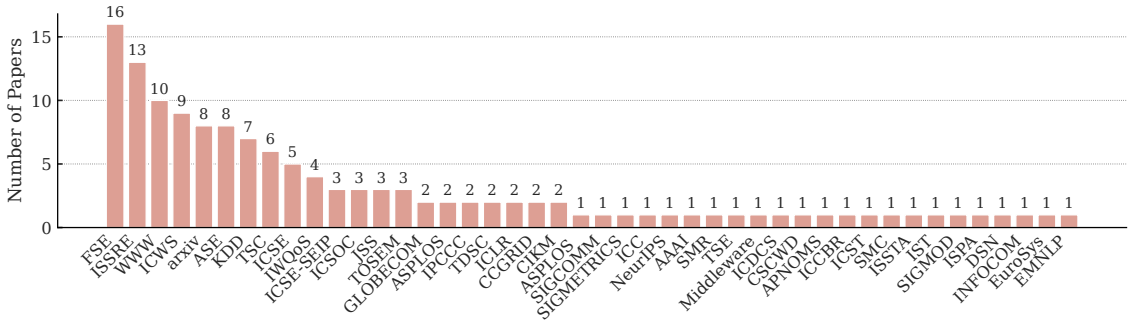

Fig. 6. Distribution of the collected papers across various research venues.

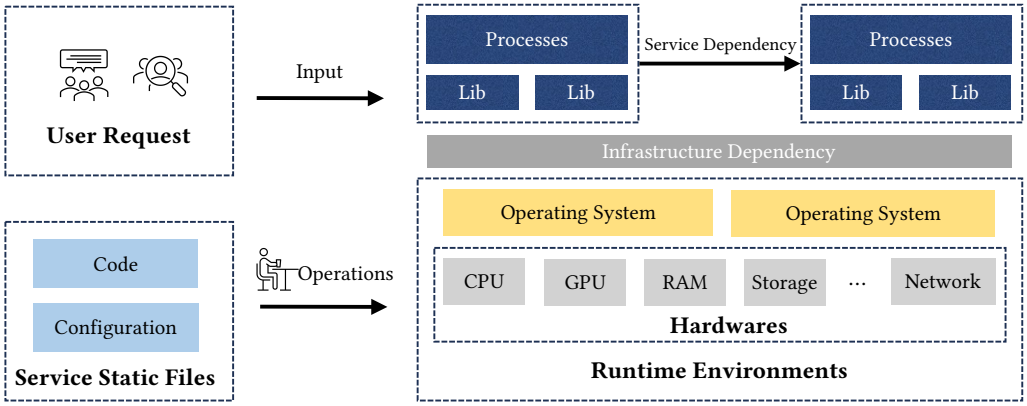

Fig. 7. Hierarchical Root Cause Model, showing all potential locations where a root cause can manifest in a microservice system.

growth since 2021, with the number of papers peaking at 40 in 2024. This steady increase underscores the growing importance and recognition of RCA research within the academic community. Furthermore, the figure highlights a substantial rise in papers involving collaboration with industry, especially in recent years. This trend suggests that RCA research is not only gaining academic interest but is also increasingly addressing challenges of significant practical relevance to the industry.

Fig. 6 illustrates the distribution of these papers across a diverse set of prestigious research venues. A significant portion of RCA papers are published in premier software engineering venues, such as FSE, ISSRE, ASE, and ICSE. At the same time, there are notable contributions in leading data mining (e.g., KDD), systems (e.g., ASPLOS), and security (e.g., TDSC) venues. This broad distribution highlights that RCA is a topic of interest that spans multiple research fields, demonstrating its wide-ranging applicability and relevance.

12.2 Settings of RCA

The diversity of ground truth root causes directly impacts the effectiveness of RCA models. In practice, failures can originate from any component in the root cause runtime model shown in Fig. 7. By analyzing 135 papers that explicitly

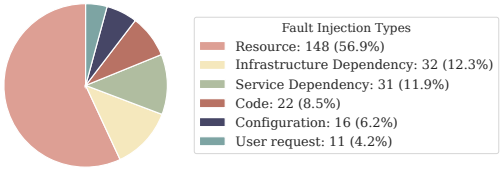

Fig. 8. Distribution of root cause types across 135 papers. The chart shows the proportion of studies focusing on each root cause type. Note that a single paper may address multiple root cause types and is thus counted in each applicable category.

discuss ground truth root causes used in their experiments, we categorize these failures into six groups based on their point of origin:

- *Resource*: Failures originating from the exhaustion or contention of fundamental system resources, where the system's provisioned capacity is insufficient for the given workload. This category includes issues like CPU saturation, memory exhaustion (OOM), insufficient disk space, I/O bottlenecks, and network bandwidth limitations, assuming no underlying software defects are the direct cause. For example, this applies when a correctly implemented service fails due to under-provisioning.
- *Code*: Failures originating from defects within the service's source code. This includes logic errors, concurrency bugs, memory leaks, inefficient algorithms leading to performance degradation, or faulty code changes. A failure is classified as a code issue even if its symptom is resource exhaustion (e.g., a memory leak causing an OOM error), because the initial trigger is the software defect itself.
- *Configuration*: Failures originating from incorrect values in externalized application or environment settings. This includes misconfigured parameters in deployment files (e.g., YAML, Helm charts), incorrect environment variables, or erroneous settings in external configuration services. This category is distinct from code-level defects as these issues can be remediated without changing the compiled source code.
- *User Request*: Failures originating from the characteristics of incoming user requests that the system cannot handle correctly. This includes unexpected request patterns (e.g., traffic surges, DDoS attacks), malformed or malicious inputs (e.g., SQL injection payloads), or "poison pill" requests that trigger latent bugs.
- *Infrastructure Dependency*: Failures originating from the functional correctness or availability of underlying, general-purpose infrastructure services. This includes database deadlocks, message queue service unavailability, or cache service failures. This category is distinct from *Resource* issues on the infrastructure's host; it concerns the failure of the service provided by the infrastructure itself.
- *Service Dependency*: Failures originating from an incorrect or unavailable response from another service within the application's architecture or from a third-party API. This includes cascading failures where an upstream service fails due to a fault in a downstream dependent service. The root cause is attributed to the dependent service that first breaks the expected contract.

These categories directly map to the hierarchical structure presented in Fig. 7. *Resource* and *Infrastructure Dependency* failures correspond to the foundational infrastructure layer. *Code* and *Configuration* issues are located within the service implementation layer. Finally, *User Request* and *Service Dependency* failures manifest at the service interaction layer, representing external and internal dependencies, respectively. This mapping provides a structured view of how different types of failures are situated within a microservice architecture.

Table 9. Root Cause Analysis Types and Detailed Entities

Benchmark[1]	Svc #	LoC/Programming Languages	Protocol	Last Update
TrainTicket [49]	45	37746/Java, 23/Go, 292/Python, 5335/JavaScript, 9733/HTML	HTTP	2022-11-01
Online Boutique [42]	11	5881/Go, 1043/Python, 740/HTML, 634/C#, 347/JavaScript, 255/Java	gRPC	2024-10-03
Sock Shop [45]	9	4010/JavaScript, 3577/Java, 3283/Go, 1640/Python	HTTP	2023-12-05
HotelReservation [4]	10	7298/Go	gRPC	2024-06-28
SocialNetwork [4]	12	5753/C++	Thrift	2024-06-28
Astronomy Shop [50]	14	10306/C++, 6714/TypeScript, 4452/Go, 1286/JavaScript, 1045/Elixir, 1035/Python, 696/C#, 401/Java, 313/Rust, 193/PHP, 67/Kotlin, 53/Ruby	gRPC/HTTP	2025-10-21
TeaStore [83]	7	12317/Java, 1693/JavaScript	HTTP	2025-01-08

As shown in Fig. 8, the high proportion of papers focusing on resource-related failures (56.9%) indicates that much of the RCA research prioritizes resource management issues. However, this heavy focus raises important questions about whether RCA research is adequately addressing other critical failure types. In real-world microservice systems, failures often arise from complex or less obvious causes, such as misconfigurations, code bugs, or intricate interactions between services. The relatively low percentages for code-related (8.5%) and configuration-related (6.2%) failures are particularly noteworthy. In practice, misconfigurations and software bugs are common sources of severe outages and performance degradation. This discrepancy suggests that current RCA research may not be adequately addressing these significant real-world challenges. Additionally, with only 4.2% of studies focusing on user request-related failures, there is a notable gap in research addressing how user behaviors, both intentional and unintentional, contribute to system failures.

This distribution pattern reflects an implicit simplification strategy in the field. From the perspective of our formalization in Section 3.3, resource-related root causes typically produce clear, observable signals in metrics (as part of the observation space O), making them easier to diagnose from a technical perspective. In contrast, detecting failures from code bugs, misconfigurations, or user requests often requires more sophisticated analysis of logs, traces, and their correlations, which substantially increases the complexity of inference within the problem space.

12.3 Benchmarks

This section lists the public benchmarks in the literature based on our collected papers. Publicly available benchmarks are important to RCA research since the companies usually cannot open-source their data due to privacy and security issues. Additionally, root cause analysis is very related to the industry practice, and the cloud environment is highly dynamic. As a result, researchers need publicly available benchmarks to inject corresponding failures, stimulate the environment, and show the effectiveness of the proposed method.

Table 9 provides a detailed summary of seven prominent public benchmarks used in RCA research. The table details each benchmark's scale (number of services and lines of code), technological stack (programming languages and protocols), and maintenance status (last update). The benchmarks exhibit significant diversity. TrainTicket [49] remains the largest in terms of service count, while TeaStore [83] and Astronomy Shop [50] feature substantial Java and polyglot codebases, respectively. Protocols range from the conventional HTTP to the more modern gRPC and Thrift, reflecting varied architectural choices in microservice systems.

[1]We are only counting the core business logic code and excluding any auto-generated code or infrastructure services like databases.

Table 10. Public dataset for root cause analysis (M for metrics, L for logs, and T for traces)

Dataset	Type	Format	Amount	Dataset	Type	Format	Amount
Dycause[7]	M	XLSX	4.1MB	DéjàVu-A1[8]	M	CSV	75.1MB
GrayScope[32]	M	CSV	8.4MB	DéjàVu-A2[8]	M	CSV	82.2MB
RCD-SS[43]	M	CSV	16MB	MicroCU[39]	M	npy	118MB
DéjàVu-C[8]	M	CSV	48.8MB	RCAEval1-OB[2]	M	CSV	166.84MB
ChangeRCA-OB[3]	M	CSV	60MB	RCAEval1-SS[2]	M	CSV	484.17MB
DéjàVu-B[8]	M	CSV	1.7GB	RCAEval1-TT[2]	M	CSV	1.75GB
LatentScope[36]	M	JSON	2.1GB	DéjàVu-D[8]	M	CSV	3.7GB
Squeeze[48]	M	CSV	18GB	MEPFL-SS[37]	T	CSV	59MB
MEPFL-TT[37]	T	CSV	2.3GB	AIOps Comp-2020[1]	M, T	CSV	16GB
Murphy-DSB[40]	M, T	JSON	99GB	GAMMA[11]	M, L	RAW format/CSV	39GB
RCAEval2-SS[71]	M, L	CSV	2.16GB	RCAEval3-SS[71]	M, L	CSV	872.18MB
Nezha-TT[41]	M, T, L	CSV	351MB	Eadro-TT[9]	M, T, L	CSV/JSON	841MB
FAMOS-TT[98]	M, T, L	CSV/Parquet	925MB	Eadro-SN[9]	M, T, L	CSV/JSON	1.3GB
RCAEval3-OB[71]	M, T, L	CSV	1.31GB	RCAEval3-TT[71]	M, T, L	CSV	2.07GB
Nezha-OB[41]	M, T, L	CSV	2.5GB	AIOps Comp-2025[81]	M, T, L	Parquet	5.8GB
RCAEval2-OB[71]	M, T, L	CSV	8.05GB	FAMOS-Mall[98]	M, T, L	CSV/Parquet	9.55GB
Fang et al.[100]	M, T, L	Parquet	12.70GB	RCAEval2-TT[71]	M, T, L	CSV	21.72GB
AIOps Comp-2021[1]	M, T, L	CSV	25GB	GAIA[10]	M, T, L	CSV	41GB

However, a critical analysis of their maintenance status reveals a significant challenge for the research community. Several foundational benchmarks show signs of abandonment. TrainTicket [49] has not been updated in over two years, and Sock Shop [45] is officially archived. DeathStarBench [4] also suffers from stalled development, with only a subset of its promised benchmarks released and recent activity confined to minor bug fixes. In contrast, Online Boutique [42], TeaStore [83], and the newly introduced Astronomy Shop [50] appear to be under active maintenance, receiving regular updates. This maintenance gap creates a bifurcation in the landscape, where newer research may gravitate towards actively maintained but potentially less complex systems.

A more profound limitation lies in the inherent design of these benchmarks. While they provide environments for evaluating fault localization, their static nature and limited complexity hinder research on higher-level RCA goals [100]. For instance, they are ill-suited for studying *Adaptive Learning*, as they do not model system evolution. Furthermore, their fault injection capabilities are often restricted to resource-level failures, offering little support for investigating complex, non-resource-related issues like logic bugs or misconfigurations. This deficiency makes it difficult to develop and validate methods aimed at achieving deep *Interpretability* or *Actionability*. This maintenance and complexity bottleneck suggests that the community needs either renewed investment in existing benchmarks or a shift toward automatically-generated, dynamically-maintained benchmark environments.

12.4 Datasets

This section presents an overview of the publicly available datasets relevant to root cause analysis (RCA), as identified from the literature. Table 10 summarizes the key attributes of these datasets, including their data types (metrics, traces, logs), data formats (*e.g.*, CSV, JSON), dataset size (calculated after decompression), and the research papers that utilized these datasets. Most datasets are collected from the public benchmarks, *e.g.*, Online Boutique [42] (OB), SockShop [45]

Table 11. Publicly Available Tools/Codes for Root Cause Analysis

Tool	URL	Year	Tool	URL	Year	Tool	URL	Year	Tool	URL	Year	Tool	URL	Year
Sieve	[44]	2017	Log3C	[22]	2018	Squeeze	[28]	2019	MEPFL	[68]	2019	TraceAnomaly	[69]	2020
MicroRank	[18]	2021	TraceRCA	[29]	2021	DyCause	[70]	2021	PDiagnose	[65]	2021	DéjàVu	[26]	2022
GIED	[17]	2022	CIRCA	[24]	2022	MicroCBR	[15]	2022	RCD	[13]	2022	SwissLog	[20]	2022
Diagfusion	[12]	2023	MicroCU	[21]	2023	CMDiagnostor	[25]	2023	CausalRCA	[58]	2023	Nezha	[19]	2023
Eadro	[14]	2023	ShapleyIQ	[67]	2023	TraceStream	[62]	2023	BARO	[30]	2024	LoFI	[66]	2024
ART	[59]	2024	LatentScope	[27]	2024	HeMiRCA	[78]	2024	Chain-of-Event	[23]	2024	DeepHunt	[60]	2024
ChangeRCA	[16]	2024	KPIRoot	[74]	2024	CHASE	[52]	2024	Medicine	[53]	2024	MicroIRC	[75]	2024
UniDiag	[55]	2024	MicroDig	[64]	2024	RUN	[79]	2024	SLIM	[72]	2024	mABC	[80]	2024
LasRCA	[57]	2024	FaaSRCA	[61]	2024	SCELM	[54]	2025	DiagMLP	[76]	2025	LEMMA-RCA	[82]	2025
RCAEval	[71]	2025	AERCA	[63]	2025	TVdiag	[77]	2025	TrioXpert	[73]	2025	FAMOS	[56]	2025

(SS), TrainTicket [49] (TT), DeathStarBench [4] (DSB), SocialNetwork [4] (SN). The table reveals a clear trend toward multi-modal and larger-scale datasets over time. Early datasets like Dycause [7] and GrayScope [32] were small and focused exclusively on metrics (M). In contrast, more recent contributions such as GAIA [10], FAMOS [98], and the RCAEval series [71] incorporate a combination of metrics (M), logs (L), and traces (T), with sizes scaling into tens of gigabytes. This evolution reflects the community's growing recognition that effective RCA requires a holistic view of the system, integrating diverse telemetry sources.

Despite this progress, a critical gap persists from the perspective of our formal framework (Section 3.3). While these datasets provide rich subsets of the observation space O, nearly all of them lack ground truth labels for the complete incident propagation graph $\mathcal{G}$. The provided labels are typically confined to identifying the root cause node, such as a faulty service or metric. This limitation reinforces a "point-finding" research paradigm, where the primary goal is to pinpoint a single fault origin. This scarcity of comprehensively labeled, graph-based ground truth is a major bottleneck for developing and evaluating methods aimed at achieving higher-level goals like *Interpretability*. Without the ground truth of the causal chain, researchers cannot validate the correctness of inferred causal paths, hindering the transition toward more sophisticated "graph-building" RCA models. Future dataset collection efforts must prioritize capturing not just the root cause, but the entire causal chain of events to foster this next generation of RCA research.

12.5 Public Available Tools

This section compiles a collection of publicly accessible toolkits and codebases that can facilitate further research in root cause analysis. Among the 135 papers we reviewed, 50 have made their implementations publicly available, as cataloged in Table 11. The chronological listing of these artifacts reveals a significant acceleration in open-source contributions, with the number of tools released in 2024 alone nearly matching the total from all preceding years combined. This trend toward openness, exemplified by recent and notable tools like BARO [170], LatentScope [208], and Chain-of-Event [216], is crucial for promoting transparency, reproducibility, and standardized evaluation within the community.

The evolution of these tools also mirrors the field's methodological shifts. Early contributions such as Sieve [189] and Log3C [22] laid the groundwork, while more recent systems like Eadro [133] and FAMOS [56] demonstrate increasing sophistication in handling multi-modal data and complex causal inference. The availability of this diverse array of open-source tools provides an invaluable resource for the community. It enables researchers to benchmark new techniques

against established baselines, adapt existing models for novel scenarios, and build upon prior work to push the frontiers of the field. This collaborative ecosystem is essential for systematically advancing RCA research.

13 DISCUSSION

13.1 Threats to Validity

Data source credibility. This survey covers only a subset of the available literature, with a focus on papers related to microservices root cause analysis published in top-tier conferences and journals over the past decade. Due to limitations in both time and resources, it was not feasible to collect all relevant works, which may result in some incompleteness. For instance, while the pipeline in RootCLAM [112] aligns with the general scope of this paper, which encompasses anomaly detection, root cause localization, and anomaly mitigation, the specific context of RootCLAM [112] is quite different from ours. RootCLAM [112] utilizes a loan approval scenario based on the German Credit dataset, which falls outside the domain of incident management. Consequently, works that do not pertain to incident management, such as RootCLAM, were excluded from our discussion. However, its inclusion highlights the broader applicability of root cause analysis, which extends beyond incident management scenarios.

Moreover, while we have strived to ensure the accuracy of our literature understanding and analysis, there is an inherent risk of subjective interpretation errors during the reading process. To mitigate these risks, we employed a cross-validation approach: the primary authors independently read and summarized the papers, followed by a cross-review of the results to enhance the accuracy and consistency of our findings.

13.2 Implications of the Goal-Driven Framework

Our seven-goal framework serves not merely as a taxonomy, but as an analytical lens to reveal the underlying dynamics of RCA research. The goals are not independent; they exist in a web of synergies and tensions that shape the field.

Synergies and Tensions. There are clear synergistic relationships. For instance, advances in *Interpretability* (Goal 5), particularly the construction of causal graphs, directly enhance *Actionability* (Goal 7) by providing a logical basis for automated remediation. Conversely, fundamental tensions force trade-offs. The pursuit of *Real-time Performance* (Goal 4) often necessitates algorithmic shortcuts that may compromise the depth of *Interpretability* (Goal 5). Similarly, highly *Adaptive* models (Goal 3) may struggle to maintain *Robustness* (Goal 2) against noisy data, highlighting a conflict between dynamic adaptation and static resilience. Recognizing these trade-offs is crucial for both researchers designing new methods and practitioners selecting tools for specific operational contexts.

Evolutionary Trends. The framework also illuminates an evolutionary trend in the field. Early research predominantly focused on foundational goals like achieving *Real-time Performance* and *Multi-granularity*. With the advent of Large Language Models (LLMs), the research frontier is rapidly expanding towards more semantic and human-centric goals, namely *Interpretability* and *Actionability*, signaling a shift from "what happened" to "why it happened and what to do about it."

13.3 Bridging the Gap to Ideal RCA

Our formalization of ideal RCA as a function $\mathcal{F} : \mathcal{O} \rightarrow \mathcal{G}$ that maps observations to a complete incident propagation graph, provides a "north star" for the field. However, our survey reveals a significant gap between this ideal and the current state of the art.

The Paradigm Shift: From Pinpointing Causes to Graphing Propagation. The vast majority of existing research still operates under a simplified paradigm: pinpointing a single root cause node ($r \in \mathcal{G}$). The true challenge, and the next frontier, lies in the paradigm shift towards constructing the full propagation graph $\mathcal{G}$. This shift is hindered by three fundamental gaps:

- **The Evaluation Gap:** There is a critical lack of standardized benchmarks and metrics to evaluate the correctness of a generated propagation graph. Without a target to aim for, progress is inherently limited.
- **The Data Gap:** Correspondingly, few, if any, public datasets provide ground-truth propagation graphs for incidents, making supervised learning of such structures nearly impossible.
- **The Methodology Gap:** Current methods are not explicitly optimized for graph generation. Causal discovery algorithms struggle with the scale and dynamics of real systems, while LLMs, despite their semantic power, lack the structural grounding to reliably produce verifiable causal chains.

Revisiting Triggers vs. Root Causes. Furthermore, the ideal graph $\mathcal{G}$ must distinguish between the underlying *root cause* (e.g., a buggy code commit) and the *trigger* (e.g., a specific user input that activates the bug). This distinction, vital for both immediate mitigation and long-term fixes, is largely overlooked in current research but is essential for achieving a truly comprehensive diagnosis.

13.4 Future Research Frontiers

Based on the identified gaps and implications, we outline three forward-looking research frontiers that aim to systematically advance the field towards the ideal of RCA.

The Next-Generation RCA Benchmark. The most pressing need is a community-driven effort to build a large-scale, multi-modal benchmark dataset where incidents are annotated not with single root causes, but with complete, ground-truth propagation graphs ($\mathcal{G}$). Such a benchmark would catalyze the development and rigorous evaluation of the next generation of RCA models.

Unified Models for Causal Graph Generation. Future research should move beyond single-cause localization and focus on novel architectures designed explicitly for end-to-end propagation graph generation. This may involve hybrid models that fuse the structural reasoning of GNNs, the semantic and common-sense understanding of LLMs, and the exploratory capabilities of reinforcement learning to navigate vast diagnostic search spaces.

Deep Integration with the Software Engineering Lifecycle. The output of RCA should not be the end of the story. A truly impactful direction is to create a feedback loop from RCA back into the software engineering lifecycle. The generated graph $\mathcal{G}$ could automatically inform and trigger actions such as pinpointing the exact faulty code commit, generating new regression tests that codify the failure scenario, and providing actionable insights for architectural redesign, thus transforming RCA from a reactive diagnostic tool into a proactive driver of system reliability.

14 RELATED WORK

Several surveys have provided valuable overviews of the Root Cause Analysis (RCA) landscape. However, their taxonomies are often based on surface-level features, such as input data modalities (*e.g.*, logs, metrics, traces). We argue that such classifications obscure the underlying strategic goals that drive RCA research. Our work distinguishes itself by introducing a goal-driven framework that offers a more insightful and functionally relevant perspective.

Surveys Based on Data Modality and Techniques. The most related surveys from Soldani et al. [178], Zhang et al. [228], and Wang et al. [197] organize the field primarily by data sources. While providing a useful catalog of methods, this

data-centric view has a key limitation: the mapping between data types and research objectives is not one-to-one. For example, methods using different data types might share the same goal (*e.g.*, *Real-time Performance*), while methods using the same data type might pursue different goals (*e.g.*, *Interpretability* vs. *Adaptive Learning*). This hinders a clear comparison of methodological trade-offs. Our survey moves beyond this by classifying works based on the seven fundamental goals defined in Section 3.2.

Surveys on Specific RCA Sub-domains. Other works focus on narrower aspects of RCA, such as causal inference-based methods [171] or fault localization in software engineering [201]. These specialized surveys are complementary to our work. Our goal-driven framework can contextualize their findings within the broader RCA landscape. For instance, challenges in causal graph construction [171] directly relate to the pursuit of *Interpretability* (Goal 5) and the gaps we identify in Section 13.

Our Unique Contribution. Unlike previous surveys that document "what has been done," our work provides a conceptual framework to understand "why it was done" and "where to go next." By focusing on seven orthogonal goals, we enable a more meaningful comparison of approaches and illuminate the inherent trade-offs in the field. Furthermore, our formalization of the ideal RCA output as an incident propagation graph ($\mathcal{G}$) provides a "north star" for the community, allowing us to systematically identify the gap between the current state of the art and the ultimate goal of explaining the full causal story of an incident. In summary, our survey offers a more profound, goal-oriented synthesis that provides a clearer roadmap for future research.

15 CONCLUSION

In this paper, we addressed the fragmentation that has long characterized Root Cause Analysis (RCA) research, a field where the prevalence of task-specific solutions has hindered a unified understanding. We argued that traditional surveys, which categorize methods by data types, fail to capture the underlying objectives driving the research. In response, we proposed a new conceptual framework built on two core contributions: a formal definition of ideal RCA as the generation of a complete incident propagation graph ($\mathcal{G}$), and a seven-goal taxonomy derived from the practical needs of the incident management lifecycle. Through the lens of this goal-driven framework, our analysis systematically deconstructed the field. This revealed the inherent synergies and trade-offs, such as the tension between real-time performance and interpretability, that shape all RCA systems. More critically, this perspective allowed us to identify a fundamental gap between the community's dominant focus on pinpointing single root causes and the ideal of constructing the full propagation graph. We contend that bridging this gap is the central challenge for the field, requiring a focused effort to overcome fundamental obstacles in evaluation, data availability, and methodology. Building on these insights, we outlined a future research agenda centered on three key areas: creating a next-generation benchmark with ground-truth graphs, developing unified models for causal graph generation, and deeply integrating RCA into the software engineering lifecycle. Ultimately, this survey aims not only to document the past but also to provide a structured roadmap to guide the community toward a more systematic and impactful future, advancing root cause analysis from a reactive practice into a proactive, data-driven science.

REFERENCES

[1] 2024. AIOps Competition Dataset. https://www.aiops.cn/åd'ŽæĺąæĂĄæ̧řæIJő/. Accessed: 2024-10-03.
[2] 2024. BARO Dataset. https://zenodo.org/records/11046533. Accessed: 2024-10-03.
[3] 2024. ChangeRCA Dataset. https://github.com/IntelligentDDS/ChangeRCA. Accessed: 2024-10-03.
[4] 2024. DeathStarBench. https://github.com/delimitrou/DeathStarBench/tree/master. Accessed: 2024-10-03.

[5] 2024. Docker. https://www.docker.com/. Accessed: 2024-08-19.
[6] 2024. Dubbo. https://dubbo.apache.org/en/index.html. Accessed: 2024-08-19.
[7] 2024. Dycause Dataset. https://github.com/PanYicheng/dycause_rca/tree/main. Accessed: 2024-10-03.
[8] 2024. Déjàvu Dataset. https://zenodo.org/records/6955909. Accessed: 2024-10-03.
[9] 2024. Eadro Dataset. https://zenodo.org/records/7615394. Accessed: 2024-10-03.
[10] 2024. GAIA Dataset. https://github.com/CloudWise-OpenSource/GAIA-DataSet. Accessed: 2024-10-03.
[11] 2024. GAMMA Dataset. https://www.kaggle.com/datasets/gagansomashekar/microservices-bottleneck-detection-dataset. Accessed: 2024-10-03.
[12] 2024. GitHub - AIOps-Lab-NKU/DiagFusion. https://github.com/AIOps-Lab-NKU/DiagFusion. Accessed: 2025-10-20.
[13] 2024. GitHub - azamikram/rcd: Root Cause Discovery: Root Cause Analysis of Failures in Microservices through Causal Discovery. https://github.com/azamikram/rcd. Accessed: 2025-10-20.
[14] 2024. GitHub - BEbillionaireUSD/Eadro. https://github.com/BEbillionaireUSD/Eadro. Accessed: 2025-10-20.
[15] 2024. GitHub - Fengrui-Liu/MicroCBR: Official repository for MicroCBR. https://github.com/Fengrui-Liu/MicroCBR. Accessed: 2025-10-20.
[16] 2024. GitHub - IntelligentDDS/ChangeRCA. https://github.com/IntelligentDDS/ChangeRCA. Accessed: 2025-10-20.
[17] 2024. GitHub - IntelligentDDS/GIED: Graph based Incident Extraction and Diagnosis in Large-Scale Online Systems (ASE'22). https://github.com/IntelligentDDS/GIED. Accessed: 2025-10-20.
[18] 2024. GitHub - IntelligentDDS/MicroRank: MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. https://github.com/IntelligentDDS/MicroRank. Accessed: 2025-10-20.
[19] 2024. GitHub - IntelligentDDS/Nezha: The implementation of multimodal observability data root cause analysis approach Nezha in FSE 2023. https://github.com/IntelligentDDS/Nezha. Accessed: 2025-10-20.
[20] 2024. GitHub - IntelligentDDS/SwissLog: The implementation of SwissLog in ISSRE'20 and TDSC'22. https://github.com/IntelligentDDS/SwissLog. Accessed: 2025-10-20.
[21] 2024. GitHub - jxrjxrjxr/MicroCU. https://github.com/jxrjxrjxr/MicroCU. Accessed: 2025-10-20.
[22] 2024. GitHub - logpai/Log3C: Log-based impactful problem identification using machine learning [FSE'18]. https://github.com/logpai/Log3C. Accessed: 2025-10-20.
[23] 2024. GitHub - NetManAIOps/Chain-of-Event. https://github.com/NetManAIOps/Chain-of-Event. Accessed: 2025-10-20.
[24] 2024. GitHub - NetManAIOps/CIRCA: Causal Inference-based Root Cause Analysis. https://github.com/NetManAIOps/CIRCA. Accessed: 2025-10-20.
[25] 2024. GitHub - NetManAIOps/CMDiagnostor. https://github.com/NetManAIOps/CMDiagnostor. Accessed: 2025-10-20.
[26] 2024. GitHub - NetManAIOps/DejaVu: Code and datasets for FSE'22 paper "Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems". https://github.com/NetManAIOps/DejaVu. Accessed: 2025-10-20.
[27] 2024. GitHub - NetManAIOps/LatentScope: Source Code and Dataset B for KDD 24 Paper "Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space". https://github.com/NetManAIOps/LatentScope. Accessed: 2025-10-20.
[28] 2024. GitHub - NetManAIOps/Squeeze: ISSRE 2019: Generic and Robust Localization of Multi-Dimensional Root Cause. https://github.com/lizeyan/Squeeze. Accessed: 2025-10-20.
[29] 2024. GitHub - NetManAIOps/TraceRCA: Practical Root Cause Localization for Microservice Systems via Trace Analysis. IWQoS 2021. https://github.com/NetManAIOps/TraceRCA. Accessed: 2025-10-20.
[30] 2024. GitHub - phamquiluan/baro: [FSE'24 - Best Artifact Award] BARO: Robust Root Cause Analysis for Microservice Systems. https://github.com/phamquiluan/baro. Accessed: 2025-10-20.
[31] 2024. Gitlab. https://about.gitlab.com/. Accessed: 2024-08-19.
[32] 2024. GrayScope Dataset. https://gitee.com/milohaha/grayscope/tree/master. Accessed: 2024-10-03.
[33] 2024. Incident Management Guide. https://sre.google/resources/practices-and-processes/incident-management-guide/. Accessed: 2024-08-19.
[34] 2024. Jenkins. https://www.jenkins.io/. Accessed: 2024-08-19.
[35] 2024. Kubernetes. https://kubernetes.io/. Accessed: 2024-08-19.
[36] 2024. LatentScope Dataset. https://github.com/NetManAIOps/LatentScope. Accessed: 2024-10-03.
[37] 2024. MEPFL Dataset. https://github.com/FudanSELab/Research-ESEC-FSE2019-AIOPS. Accessed: 2024-10-03.
[38] 2024. Mesos. https://mesos.apache.org/. Accessed: 2024-08-19.
[39] 2024. MicroCU Dataset. https://github.com/jxrjxrjxr/MicroCU/tree/main. Accessed: 2024-10-03.
[40] 2024. Murphy Dataset. https://github.com/netarch/Murphy-traces. Accessed: 2024-10-03.
[41] 2024. Nezha Dataset. https://github.com/IntelligentDDS/Nezha/tree/main. Accessed: 2024-10-03.
[42] 2024. Online Boutique. https://github.com/GoogleCloudPlatform/microservices-demo. Accessed: 2024-10-03.
[43] 2024. RCD Dataset. https://github.com/azamikram/rcd/tree/master. Accessed: 2024-10-03.
[44] 2024. Sieve by sieve-microservices. https://sieve-microservices.github.io/. Accessed: 2025-10-20.
[45] 2024. Sock Shop. https://github.com/microservices-demo/microservices-demo. Accessed: 2024-08-21.
[46] 2024. Spring Boot. https://spring.io/projects/spring-boot. Accessed: 2024-08-19.
[47] 2024. Spring Cloud. https://spring.io/projects/spring-cloud. Accessed: 2024-08-19.
[48] 2024. Squeeze Dataset. https://zenodo.org/records/8153367. Accessed: 2024-10-03.

[49] 2024. Train Ticket. https://github.com/FudanSELab/train-ticket. Accessed: 2024-10-03.
[50] 2025. Astronomy Shop. https://github.com/open-telemetry/opentelemetry-demo. Accessed: 2025-10-21.
[51] 2025. AWS - incident report. https://www.bbc.com/news/live/c5y8k7k6v1rt. Accessed: 2025-10-21.
[52] 2025. CHASE Dataset. https://drive.google.com/file/d/11erha3k8FeA67z-sfKGReqHz66PpO6o4/view. Accessed: 2025-10-20.
[53] 2025. GitHub - AIOps-Lab-NKU/Medicine. https://github.com/AIOps-Lab-NKU/Medicine. Accessed: 2025-10-20.
[54] 2025. GitHub - AIOps-Lab-NKU/SCELM-NKAIOps. https://github.com/AIOps-Lab-NKU/SCELM-NKAIOps. Accessed: 2025-10-20.
[55] 2025. GitHub - AIOps-Lab-NKU/UniDiag. https://github.com/AIOps-Lab-NKU/UniDiag. Accessed: 2025-10-20.
[56] 2025. GitHub - alibabacloud-observability/FAMOS. https://github.com/alibabacloud-observability/FAMOS. Accessed: 2025-10-20.
[57] 2025. GitHub - AmanecerTrio/LasRCA. https://github.com/AmanecerTrio/LasRCA. Accessed: 2025-10-20.
[58] 2025. GitHub - AXinx/CausalRCA_code. https://github.com/AXinx/CausalRCA_code. Accessed: 2025-10-20.
[59] 2025. GitHub - bbyldebb/ART. https://github.com/bbyldebb/ART. Accessed: 2025-10-20.
[60] 2025. GitHub - bbyldebb/DeepHunt. https://github.com/bbyldebb/DeepHunt. Accessed: 2025-10-20.
[61] 2025. GitHub - FaaSRCA. https://anonymous.4open.science/r/submission-C4C8. Accessed: 2025-10-20.
[62] 2025. GitHub - FudanSELab/TraceStream. https://github.com/FudanSELab/TraceStream. Accessed: 2025-10-20.
[63] 2025. GitHub - hanxiao0607/AERCA. https://github.com/hanxiao0607/AERCA. Accessed: 2025-10-20.
[64] 2025. GitHub - hburning/MicroDig. https://github.com/hburning/MicroDig. Accessed: 2025-10-20.
[65] 2025. GitHub - jia-tong-FINE/PDiagnosis. https://github.com/jia-tong-FINE/PDiagnosis. Accessed: 2025-10-20.
[66] 2025. GitHub - Jun-jie-Huang/LoFI. https://github.com/Jun-jie-Huang/LoFI. Accessed: 2025-10-20.
[67] 2025. GitHub - lonyle/ShapleyIQ. https://github.com/lonyle/ShapleyIQ. Accessed: 2025-10-20.
[68] 2025. GitHub - MEPFL. https://fudanselab.github.io/Research-ESEC-FSE2019-AIOPS/. Accessed: 2025-10-20.
[69] 2025. GitHub - NetManAIOps/anomalyDetection. https://github.com/NetManAIOps/anomalyDetection. Accessed: 2025-10-20.
[70] 2025. GitHub - PanYicheng/dycause_rca. https://github.com/PanYicheng/dycause_rca. Accessed: 2025-10-20.
[71] 2025. GitHub - phamquiluan/RCAEval. https://github.com/phamquiluan/RCAEval. Accessed: 2025-10-20.
[72] 2025. GitHub - renruirui1234/SLIM. https://github.com/renruirui1234/SLIM. Accessed: 2025-10-20.
[73] 2025. GitHub - TrioXpert. https://anonymous.4open.science/r/TrioXpert-F244/README.md. Accessed: 2025-10-20.
[74] 2025. GitHub - WenweiGu/KPIRoot. https://github.com/WenweiGu/KPIRoot. Accessed: 2025-10-20.
[75] 2025. GitHub - WHU-AISE/MicroIRC. https://github.com/WHU-AISE/MicroIRC. Accessed: 2025-10-20.
[76] 2025. GitHub - WHU-AISE/TVDiag. https://github.com/WHU-AISE/TVDiag. Accessed: 2025-10-20.
[77] 2025. GitHub - WHU-AISE/TVDiag. https://github.com/WHU-AISE/TVDiag. Accessed: 2025-10-20.
[78] 2025. GitHub - Zhuzrx/HeMiRCA. https://github.com/Zhuzrx/HeMiRCA/. Accessed: 2025-10-20.
[79] 2025. GitHub - zmlin1998/RUN. https://github.com/zmlin1998/RUN. Accessed: 2025-10-20.
[80] 2025. GitHub - zwpride/mABC. https://github.com/zwpride/mABC. Accessed: 2025-10-20.
[81] 2025. Gitlab - aiopschallenge2025. https://www.aiops.cn/gitlab/aiops-live-benchmark/aiopschallenge2025. Accessed: 2025-10-20.
[82] 2025. LEMMA-RCA. https://lemma-rca.github.io/. Accessed: 2025-10-20.
[83] 2025. TeaStore. https://github.com/DescartesResearch/TeaStore. Accessed: 2025-10-21.
[84] 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.
[85] Anunay Amar and Peter C Rigby. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 140–151.
[86] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. 2020. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* 159 (2020), 110432.
[87] Yang Cai, Biao Han, Jie Li, Na Zhao, and Jinshu Su. 2021. Modelcoder: A fault model based automatic root cause localization framework for microservice systems. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–6.
[88] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.
[89] 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.
[90] Ruibo Chen, Fang Peng, Xin Ji, Nan Xiang, Yihua Lou, Kui Zhang, Yanjun Pu, and Wenjun Wu. 2024. Graph-Based Ensemble Learning for Enhanced Fault Localization in Microservices. In *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 3356–3362.
[91] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, 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.
[92] Yuhua Chen, Dongqi Xu, Ningjiang Chen, and Xu Wu. 2023. FRL-MFPG: Propagation-aware fault root cause location for microservice intelligent operation and maintenance. *Information and Software Technology* 153 (2023), 107083.

[93] Han Cheng, Qian Li, Bingchen Liu, Shijun Liu, and Li Pan. 2024. DGERCL: A dynamic graph embedding approach for root cause localization in microservice systems. *IEEE Transactions on Services Computing* (2024).

[94] Yiran Cheng, Bo Cheng, Pengxiang Jin, Yongqian Sun, Xiaohui Nie, Nengwen Zhao, Shenglin Zhang, and Dan Pei. 2022. Effective attribute selection for multi-dimensional root cause analysis. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 321–331.

[95] Wikipedia contributors. 2024. 2024 CrowdStrike-related IT outages. https://en.wikipedia.org/wiki/2024_CrowdStrike-related_IT_outages Accessed: 2024-12-19.

[96] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. 2017. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International conference on software architecture (ICSA)*. IEEE, 21–30.

[97] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, et al. 2023. TraceDiag: Adaptive, Interpretable, and Efficient Root Cause Analysis on Large-Scale Microservice Systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1762–1773.

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

[99] Google Site Reliability Engineering. 2024. Incident Management at Google. https://sre.google/resources/practices-and-processes/incident-management-guide/ Accessed: 2024-11-19.

[100] Aoyang Fang, Songhan Zhang, Yifan Yang, Haotong Wu, Junjielong Xu, Xuyang Wang, Rui Wang, Manyi Wang, Qisheng Lu, and Pinjia He. 2025. An Empirical Study of SOTA RCA Models: From Oversimplified Benchmarks to Realistic Failures. *arXiv preprint arXiv:2510.04711* (2025).

[101] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ML-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 135–151.

[102] Yu Gan, Guiyang Liu, Xin Zhang, Qi Zhou, Jiesheng Wu, and Jiangwei Jiang. 2023. Sleuth: A Trace-Based Root Cause Analysis System for Large-Scale Microservices with Graph Neural Networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 324–337.

[103] Fei Gao, Ruyue Xin, Xiaocui Li, and Yaqiang Zhang. 2025. Are GNNs Actually Effective for Multimodal Fault Diagnosis in Microservice Systems? *arXiv preprint arXiv:2501.02766* (2025).

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

[105] Drishti Goel, Fiza Husain, Aditya Singh, Supriyo Ghosh, Anjaly Parayil, Chetan Bansal, Xuchao Zhang, and Saravan Rajmohan. 2024. X-lifecycle learning for cloud incident management using llms. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 417–428.

[106] Drishti Goel, Raghav Magazine, Supriyo Ghosh, Akshay Nambi, Prathamesh Deshpande, Xuchao Zhang, Chetan Bansal, and Saravan Rajmohan. 2025. eARCO: Efficient Automated Root Cause Analysis with Prompt Optimization. *arXiv preprint arXiv:2504.11505* (2025).

[107] Google. 2018. Site Reliability Engineering (SRE) Workbook: Incident Response. https://sre.google/workbook/incident-response/ Accessed: 2024-12-22.

[108] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. 2023. TrinityRCL: Multi-Granular and Code-Level Root Cause Localization Using Multiple Types of Telemetry Data in Microservice Systems. *IEEE Transactions on Software Engineering* 49, 5 (2023), 3071–3088.

[109] Wenwei Gu, Xinying Sun, Jinyang Liu, Yintong Huo, Zhuangbin Chen, Jianping Zhang, Jiazhen Gu, Yongqiang Yang, and Michael R Lyu. 2024. Kpiroot: Efficient monitoring metric-based root cause localization in large-scale cloud systems. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 403–414.

[110] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. 2020. Graph-based trace analysis for microservice architecture understanding and problem diagnosis. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1387–1397.

[111] Xiao Han, Saima Absar, Lu Zhang, and Shuhan Yuan. 2025. Root Cause Analysis of Anomalies in Multivariate Time Series through Granger Causal Discovery. In *The Thirteenth International Conference on Learning Representations*.

[112] Xiao Han, Lu Zhang, Yongkai Wu, and Shuhan Yuan. 2023. On root cause localization and anomaly mitigation through causal inference. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 699–708.

[113] 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.

[114] Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. 2023. Murphy: Performance diagnosis of distributed cloud applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 438–451.

[115] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 60–70.

[116] Zilong He, Pengfei Chen, Yu Luo, Qiuyu Yan, Hongyang Chen, Guangba Yu, and Fangyuan Li. 2022. Graph based incident extraction and diagnosis in large-scale online systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[117] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021. Diagnosing performance issues in microservices with heterogeneous data source. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 493–500.

[118] Zijun Hu, Pengfei Chen, Guangba Yu, Zilong He, and Xiaoyun Li. 2022. TS-InvarNet: Anomaly detection and localization based on tempo-spatial KPI invariants in distributed services. In *2022 IEEE International Conference on Web Services (ICWS)*. IEEE, 109–119.

[119] Jin Huang, Pengfei Chen, Guangba Yu, Yilun Wang, Haiyu Huang, and Zilong He. 2024. FaaSRCA: Full Lifecycle Root Cause Analysis for Serverless Applications. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 415–426.

[120] Junjie Huang, Zhihan Jiang, Jinyang Liu, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Hui Dong, Zengyin Yang, and Michael R Lyu. 2024. Demystifying and extracting fault-indicating information from logs for failure diagnosis. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 511–522.

[121] Xiaosong Huang, Hongyi Liu, Yifan Wu, Yujin Zhao, Changlong Wu, Songlin Zhang, Ling Jiang, Tong Jia, Ying Li, and Zhonghai Wu. 2024. OCRCL: Online Contrastive Learning for Root Cause Localization of Business Incidents. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 524–534.

[122] Hiroki Ikeuchi, Akio Watanabe, Takehiro Kawata, and Ryoichi Kawahara. 2018. Root-cause diagnosis using logs generated by user actions. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–7.

[123] 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 (2022), 31158–31170.

[124] Samireh Jalali and Claes Wohlin. 2012. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. 29–38.

[125] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. 2017. Performance monitoring and root cause analysis for cloud-hosted web applications. In *Proceedings of the 26th International Conference on World Wide Web*. 469–478.

[126] Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. 2019. ExplainIt!–A declarative root-cause analysis engine for time series data. In *Proceedings of the 2019 International Conference on Management of Data*. 333–348.

[127] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 25–32.

[128] Xinrui Jiang, Yicheng Pan, Meng Ma, and Ping Wang. 2023. Look Deep into the Microservice System Anomaly through Very Sparse Logs. In *Proceedings of the ACM Web Conference 2023*. 2970–2978.

[129] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. 2024. Xpert: Empowering incident management with query recommendations via large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[130] René Just, Darioush Jalali, and Michael D Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 international symposium on software testing and analysis*. 437–440.

[131] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. 2013. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review* 41, 1 (2013), 93–104.

[132] He Kong, Tong Li, Jingguo Ge, Lei Zhang, and Liangxiong Li. 2024. Enhancing fault localization in microservices systems through span-level using graph convolutional networks. *Automated Software Engineering* 31, 2 (2024), 46.

[133] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1750–1762.

[134] J. Lewis and M. Fowler. 2014. Microservices a definition of this new architectural term. Retrieved June 25, 2024 from https://martinfowler.com/articles/microservices.html

[135] 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.

[136] Peiwen Li, Xin Wang, Zeyang Zhang, Yuan Meng, Fang Shen, Yue Li, Jialong Wang, Yang Li, and Wenwu Zhu. 2024. RealTCD: Temporal Causal Discovery from Interventional Data with Large Language Model. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4669–4677.

[137] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2022. Swisslog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (2022), 2762–2780.

[138] Ye Li, Jian Tan, Bin Wu, Xiao He, and Feifei Li. 2023. Shapleyiq: Influence quantification by shapley values for performance debugging of microservices. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 287–323.

[139] Yichen Li, Yulun Wu, Jinyang Liu, Zhihan Jiang, Zhuangbin Chen, Guangba Yu, and Michael R Lyu. 2025. COCA: Generative Root Cause Analysis for Distributed Systems with Code Knowledge. *arXiv preprint arXiv:2503.23051* (2025).

[140] Yufeng Li, Guangba Yu, Pengfei Chen, Chuanfu Zhang, and Zibin Zheng. 2022. MicroSketch: Lightweight and adaptive sketch based performance issue detection and localization in microservice systems. In *International Conference on Service-Oriented Computing*. Springer, 219–236.

[141] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[142] Zeyan Li, Chengyang Luo, Yiwei Zhao, Yongqian Sun, Kaixin Sui, Xiping Wang, Dapeng Liu, Xing Jin, Qi Wang, and Dan Pei. 2019. Generic and robust localization of multi-dimensional root causes. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 47–57.

[143] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 996–1008.

[144] 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*, Vol. 38. 206–213.

[145] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*. Springer, 3–20.

[146] Weilan Lin, Meng Ma, Disheng Pan, and Ping Wang. 2018. FacGraph: Frequent anomaly correlation graph mining for root cause diagnose in micro-service architecture. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[147] 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.

[148] Fengrui Liu, Yang Wang, Zhenyu Li, Rui Ren, Hongtao Guan, Xian Yu, Xiaofan Chen, and Gaogang Xie. 2022. MicroCBR: Case-Based Reasoning on Spatio-temporal Fault Knowledge Graph for Microservices Troubleshooting. In *International Conference on Case-Based Reasoning*. Springer, 224–239.

[149] Ping Liu, Yu Chen, Xiaohui Nie, Jing Zhu, Shenglin Zhang, Kaixin Sui, Ming Zhang, and Dan Pei. 2019. Fluxrank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 35–46.

[150] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, et al. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 48–58.

[151] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. 2017. Log-based abnormal task detection and root cause analysis for spark. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 389–396.

[152] Xianglin Lu, Zhe Xie, Zeyan Li, Mingjie Li, Xiaohui Nie, Nengwen Zhao, Qingyang Yu, Shenglin Zhang, Kaixin Sui, Lin Zhu, et al. 2022. Generic and Robust Performance Diagnosis via Causal Inference for OLTP Database Systems. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 655–664.

[153] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2019. Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 60–67.

[154] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2021. Servicerank: Root cause identification of anomaly in large-scale microservice architectures. *IEEE Transactions on Dependable and Secure Computing* 19, 5 (2021), 3087–3100.

[155] 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.

[156] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.

[157] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. 2018. Localizing faults in cloud systems. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 262–273.

[158] 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.

[159] Microsoft. 2024. Azure Status History. https://azure.status.microsoft/en-us/status/history/ Accessed: 2024-12-20.

[160] Vijayaraghavan Murali, Edward Yao, Umang Mathur, and Satish Chandra. 2021. Scalable statistical root cause analysis on app telemetry. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 288–297.

[161] Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. 2013. Fchain: Toward black-box online fault localization for cloud systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 21–30.

[162] Xiaohui Nie, Youjian Zhao, Kaixin Sui, Dan Pei, Yu Chen, and Xianping Qu. 2016. Mining causality graph for automatic web-based service diagnosis. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[163] OpenAI. 2024. API, ChatGPT & Sora Facing Issues. https://status.openai.com/incidents/ctrsv3lwd797 Accessed: 2024-12-19.

[164] OpenTelemetry Community. 2025. Add Events - OpenTelemetry Trace API. https://opentelemetry.io/docs/specs/otel/trace/api/#add-events Accessed: 2024-12-22.

[165] OpenTelemetry Community. 2025. Event Model - OpenTelemetry Metrics Data Model. https://opentelemetry.io/docs/specs/otel/metrics/data-model/#event-model Accessed: 2024-12-22.

[166] OpenTelemetry Community. 2025. Events - OpenTelemetry Logs Data Model. https://opentelemetry.io/docs/specs/otel/logs/data-model/#events Accessed: 2024-12-22.

[167] OpenTelemetry Community. 2025. Events - OpenTelemetry Semantic Conventions. https://opentelemetry.io/docs/specs/semconv/general/events/ Accessed: 2024-12-22.

[168] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. 2021. Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 646–657.

[169] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, 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.

[170] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. BARO: Robust Root Cause Analysis for Microservices via Multivariate Bayesian Online Change Point Detection. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2214–2237.

[171] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. Root Cause Analysis for Microservices based on Causal Inference: How Far Are We?. In *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 706–718.

[172] Rui Ren. 2025. The Multi-Agent Fault Localization System Based on Monte Carlo Tree Search Approach. *arXiv preprint arXiv:2507.22800* (2025).

[173] R. Ren, J. Yang, L. Yang, X. Gu, and L. Sun. 2024. SLIM: A Scalable and Interpretable Light-weight Fault Localization Algorithm for Imbalanced Data in Microservice. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, 27–39.

[174] 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.

[175] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019. ϵ-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*. 3215–3222.

[176] Bikash Sharma, Praveen Jayachandran, Akshat Verma, and Chita R Das. 2013. CloudPD: Problem determination and diagnosis in shared dynamic clouds. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–12.

[177] Jiahao Shi, Sihang Jiang, Bo Xu, and Yanghua Xiao. 2023. Serverrca: Root cause analysis for server failure using operating system logs. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 486–496.

[178] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.

[179] Gagan Somashekar, Anurag Dutt, Mainak Adak, Tania Lorido Botran, and Anshul Gandhi. 2024. GAMMA: Graph Neural Network-Based Multi-Bottleneck Localization for Microservices Applications. In *Proceedings of the ACM on Web Conference 2024*. 3085–3095.

[180] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. 2023. LogKG: Log Failure Diagnosis through Knowledge Graph. *IEEE Transactions on Services Computing* (2023).

[181] Chang-Ai Sun, Tao Zeng, Wanqing Zuo, and Huai Liu. 2023. A trace-log-clusterings-based fault localization approach to microservice systems. In *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, 7–13.

[182] Yongqian Sun, Zihan Lin, Binpeng Shi, Shenglin Zhang, Shiyu Ma, Pengxiang Jin, Zhenyu Zhong, Lemeng Pan, Yicheng Guo, and Dan Pei. 2025. Interpretable failure localization for microservice systems based on graph autoencoder. *ACM Transactions on Software Engineering and Methodology* 34, 2 (2025), 1–28.

[183] 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* (2025).

[184] 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.

[185] Yongqian Sun, Tinghua Zheng, Xidao Wen, Weihua Kuang, Heng Liu, Shenglin Zhang, Chao Shen, Bo Wu, and Dan Pei. 2025. A Multimodal Intelligent Change Assessment Framework for Microservice Systems Based on Large Language Models. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 378–388.

[186] Lei Tao, Xianglin Lu, Shenglin Zhang, Jiaqi Luan, Yingke Li, Mingjie Li, Zeyan Li, Qingyang Yu, Hucheng Xie, Ruijie Xu, et al. 2024. Diagnosing performance issues for large-scale microservice systems with heterogeneous graph. *IEEE Transactions on Services Computing* (2024).

[187] Lei Tao, Shenglin Zhang, Zedong Jia, Jinrui Sun, Minghua Ma, Zhengdan Li, Yongqian Sun, Canqun Yang, Yuzhi Zhang, and Dan Pei. 2024. Giving every modality a voice in microservice failure diagnosis via multimodal adaptive optimization. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1107–1119.

[188] Google SRE Team. 2016. Effective Troubleshooting. In *Site Reliability Engineering*. O'Reilly Media. https://sre.google/sre-book/effective-troubleshooting/ Accessed: 2024-12-01.

[189] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve: Actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. 14–27.

[190] 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.

[191] Dongjie Wang, Zhengzhang Chen, Jingchao Ni, Liang Tong, Zheng Wang, Yanjie Fu, and Haifeng Chen. 2023. Interdependent causal networks for root cause localization. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5051–5060.

[192] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. 2021. Groot: An event-graph-based approach for root cause analysis in industrial settings. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 419–429.

[193] Kui Wang, Carol Fung, Chao Ding, Polo Pei, Shaohan Huang, Zhongzhi Luan, and Depei Qian. 2015. A methodology for root-cause analysis in component based systems. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, 243–248.

[194] Lu Wang, Chaoyun Zhang, Ruomeng Ding, Yong Xu, Qihang Chen, Wentao Zou, Qingjun Chen, Meng Zhang, Xuedong Gao, Hao Fan, et al. 2023. Root cause analysis for microservice systems via hierarchical reinforcement learning from human feedback. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5116–5125.

[195] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE international conference on web services (ICWS)*. IEEE, 142–150.

[196] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. 2018. Cloudranger: Root cause identification for cloud native systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 492–502.

[197] Tingting Wang and Guilin Qi. 2024. A comprehensive survey on root cause analysis in (Micro) services: methodologies, challenges, and trends. *arXiv preprint arXiv:2408.00803* (2024).

[198] 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.

[199] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, et al. 2024. Large language models can provide accurate and interpretable incident triage. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 523–534.

[200] 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.

[201] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[202] Canhua Wu, Nengwen Zhao, Lixin Wang, Xiaoqin Yang, Shining Li, Ming Zhang, Xing Jin, Xidao Wen, Xiaohui Nie, Wenchi Zhang, 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.

[203] Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, and Odej Kao. 2020. Performance diagnosis in cloud microservices using deep learning. In *International Conference on Service-Oriented Computing*. Springer, 85–96.

[204] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. Microrca: Root cause localization of performance issues in microservices. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–9.

[205] Zhikang Wu, Jingyu Wang, Qi Qi, Min-Gen Shu, Rui Chu, Ju-Biao Li, Jing Jin, and Danyang Chen. 2024. FlowRCA: Enhancing Microservice Reliability with Non-invasive Root Cause Analysis. In *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 1251–1258.

[206] Yong Xiang, Charley Peter Chen, Liyi Zeng, Wei Yin, Xin Liu, Hu Li, and Wei Xu. 2025. Simplifying Root Cause Analysis in Kubernetes with StateGraph and LLM. *arXiv preprint arXiv:2506.02490* (2025).

[207] Shuaiyu Xie, Jian Wang, Hanbin He, Zhihao Wang, Yuqi Zhao, Neng Zhang, and Bing Li. 2025. TVDiag: A Task-oriented and View-invariant Failure Diagnosis Framework for Microservice-based Systems with Multimodal Data. *ACM Transactions on Software Engineering and Methodology* (2025).

[208] Zhe Xie, Shenglin Zhang, Yitong Geng, Yao Zhang, Minghua Ma, Xiaohui Nie, Zhenhe Yao, Longlong Xu, Yongqian Sun, Wentao Li, et al. 2024. Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space. (2024).

[209] Zhiqiang Xie, Yujia Zheng, Lizi Ottens, Kun Zhang, Christos Kozyrakis, and Jonathan Mace. 2024. Cloud Atlas: Efficient Fault Localization for Cloud Systems using Language Models and Causal Insight. *arXiv preprint arXiv:2407.08694* (2024).

[210] Ruyue Xin, Peng Chen, and Zhiming Zhao. 2023. Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications. *Journal of Systems and Software* 203 (2023), 111724.

[211] Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. 2025. OpenRCA: Can large language models locate the root cause of software failures?. In *The Thirteenth International Conference on Learning Representations*.

[212] Shifu Yan, Caihua Shan, Wenyi Yang, Bixiong Xu, Dongsheng Li, Lili Qiu, Jie Tong, and Qi Zhang. 2022. Cmmd: Cross-metric multi-dimensional root cause analysis. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4310–4320.

[213] Jingjing Yang, Yuchun Guo, Yishuai Chen, and Yongxiang Zhao. 2023. TraceNet: Operation Aware Root Cause Localization of Microservice System Anomalies. In *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 758–763.

[214] Jingjing Yang, Yuchun Guo, Yishuai Chen, Yongxiang Zhao, Zhongda Lu, and Yuqiang Liang. 2022. Robust Anomaly Diagnosis in Heterogeneous Microservices Systems under Variable Invocations. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2722–2727.

[215] Xin-Wei Yao, Yu-Hao Ma, Qi-Chao Lu, Xing Fu, Qiang Li, Wei-Qiang Wang, and Kai-Gui Bian. 2025. MicroTR: Transaction Reproduction Fault Diagnosis Framework for Microservice on Multi-Source Data. In *2025 28th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 1676–1683.

[216] Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. 2024. Chain-of-Event: Interpretable Root Cause Analysis for Microservices through Automatically Learning Weighted Event Causal Graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 50–61.

[217] Zhenhe Yao, Haowei Ye, Changhua Pei, Guang Cheng, Guangpei Wang, Zhiwei Liu, Hongwei Chen, Hang Cui, Zeyan Li, Jianhui Li, 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.

[218] Hiroshi Yokoyama, Ryusei Shingaki, Kaneharu Nishino, Shohei Shimizu, and Thong Pham. 2024. Causal-discovery-based root-cause analysis and its application in time-series prediction error diagnosis. *arXiv preprint arXiv:2411.06990* (2024).

[219] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments. In *Proceedings of the Web Conference 2021*. 3087–3098.

[220] Guangba Yu, Pengfei Chen, Zilong He, Qiuyu Yan, Yu Luo, Fangyuan Li, and Zibin Zheng. 2024. ChangeRCA: Finding Root Causes from Software Changes in Large Online Systems. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 24–46.

[221] 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.

[222] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2023. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* 35, 10 (2023), e2413.

[223] Qingyang Yu, Changhua Pei, Bowen Hao, Mingjie Li, Zeyan Li, Shenglin Zhang, Xianglin Lu, Rui Wang, Jiaqi Li, Zhenyu Wu, et al. 2023. CMDiagnostor: An Ambiguity-Aware Root Cause Localization Approach Based on Call Metric Data. In *Proceedings of the ACM Web Conference 2023*. 2937–2947.

[224] Chenxi Zhang, Zhen Dong, Xin Peng, Bicheng Zhang, and Miao Chen. 2024. 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*. 1–12.

[225] Dylan Zhang, Xuchao Zhang, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. LM-PACE: Confidence Estimation by Large Language Models for Effective Root Causing of Cloud Incidents. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 388–398.

[226] Lingzhe Zhang, Yunpeng Zhai, Tong Jia, Chiming Duan, Siyu Yu, Jinyang Gao, Bolin Ding, Zhonghai Wu, and Ying Li. 2025. ThinkFL: Self-Refining Failure Localization for Microservice Systems via Reinforcement Fine-Tuning. *arXiv preprint arXiv:2504.18776* (2025).

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

[228] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. 2024. Failure Diagnosis in Microservice Systems: A Comprehensive Survey and Analysis. *arXiv preprint arXiv:2407.01710* (2024).

[229] Shize Zhang, Yunfeng Zhao, Jianyuan Lu, Biao Lyu, Shunmin Zhu, Zhiliang Wang, Jiahai Yang, Lin He, and Jianping Wu. 2021. CloudPin: A root cause localization framework of shared bandwidth package traffic anomalies in public cloud networks. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 367–377.

[230] Shenglin Zhang, Yongxin Zhao, Sibo Xia, Shirui Wei, Yongqian Sun, Chenyu Zhao, Shiyu Ma, Junhua Kuang, Bolin Zhu, Lemeng Pan, et al. 2024. No More Data Silos: Unified Microservice Failure Diagnosis with Temporal Knowledge Graph. *IEEE Transactions on Services Computing* (2024).

[231] Shenglin Zhang, Yongxin Zhao, Xiao Xiong, Yongqian Sun, Xiaohui Nie, Jiacheng Zhang, Fenglai Wang, Xian Zheng, Yuzhi Zhang, and Dan Pei. 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.

[232] Shenglin Zhang, Jun Zhu, Bowen Hao, Yongqian Sun, Xiaohui Nie, Jingwen Zhu, Xilin Liu, Xiaoqian Li, Yuchi Ma, and Dan Pei. 2024. Fault Diagnosis for Test Alarms in Microservices through Multi-source Data. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 115–125.

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

[234] Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, et al. 2021. Halo: Hierarchy-aware fault localization for cloud systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3948–3958.

[235] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 2024. Automated Root Causing of Cloud Incidents using In-Context Learning with GPT-4. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 266–277.

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

[237] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. 2021. CloudRCA: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4373–4382.

[238] Yujin Zhao, Ling Jiang, Ye Tao, Songlin Zhang, Changlong Wu, Tong Jia, Xiaosong Huang, Ying Li, and Zhonghai Wu. 2023. Identifying Root-Cause Changes for User-Reported Incidents in Online Service Systems. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 287–297.

[239] Ziming Zhao, Zhenwei Wang, Tiehua Zhang, Zhishu Shen, Hai Dong, Zhen Lei, Xingjun Ma, Gaowei Xu, Zhijun Ding, and Yun Yang. 2024. CHASE: A Causal Hypergraph based Framework for Root Cause Analysis in Multimodal Microservice Systems. *arXiv preprint arXiv:2406.19711* (2024).

[240] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. 2024. MULAN: Multi-modal Causal Structure Learning and Root Cause Analysis for Microservice Systems. In *Proceedings of the ACM on Web Conference 2024*. 4107–4116.

[241] Tong Zhou, Chenxi Zhang, Xin Peng, Zhenghui Yan, Pairui Li, Jianming Liang, Haibing Zheng, Wujie Zheng, and Yuetang Deng. 2023. Tracestream: Anomalous service localization based on trace stream clustering with online feedback. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 601–611.

[242] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* 47, 2 (2018), 243–260.

[243] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 683–694.

[244] Yuhan Zhu, Jian Wang, Bing Li, Yuqi Zhao, Zekun Zhang, Yiming Xiong, and Shiping Chen. 2024. Microirc: Instance-level root cause localization for microservice systems. *Journal of Systems and Software* 216 (2024), 112145.

[245] 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 (2024), 1–25.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009