Root Cause Analysis for Distributed Systems

Aoyang Fang

***Abstract*—Root Cause Analysis (RCA) is a crucial aspect of incident management in large-scale cloud services. While numerous studies have been proposed, existing surveys typically focus on specific methods and datasets without considering the difference in RCA task formulations across various works. Since the scenarios and constraints vary, the input-output format of these works is not unified, hindering the progress of this field. Our insight is that while the task formulation of each work may be unique, the final goal of their work is similar. To this end, we propose a goal-driven framework that effectively categorizes and integrates existing work based on their diverse "goals" of RCA, e.g., identifying the current failure type, localizing the ultimate root cause, etc. We also discuss open challenges and future directions in RCA. To our knowledge, this is the first survey that systematically defines the RCA task by summarizing current research. We aim for this survey to establish a standard framework for RCA that will benefit both academia and industry, ensuring compatibility with existing work and setting guidelines for future developments.**

I. Introduction

Microservices have emerged as the favored architectural style for cloud-native development in the era of cloud computing. Adopting a microservice architecture aims to 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 significantly more challenging to pinpoint the origin of issues when things go wrong. Root Cause Analysis (RCA) has emerged as a critical phase in identifying the underlying reasons for system failures. Traditional approaches to RCA, however, require substantial human effort to sift through vast quantities of telemetry data, code, and other resources. This task demands that operators possess extensive domain knowledge, a comprehensive understanding of the operational environment, familiarity with the codebase, and even insights into the Linux kernel.

In recent years, various techniques have been proposed to automate the localization of root causes for anomalies [1], [2], [3], [4], [5], [6], [7], [8], [9]. However, most of these studies have different formulations of the RCA task due to the various scenarios. For example, to identify the failure type and component, Déjàvu [6] localizes the failure unit and corresponding abnormal metric based on the observation of metrics and deployment architecture. In contrast, Onion [8] identifies incident-indicating logs by comparing normal and abnormal logs. In another instance, MicroRank[9] focuses on localizing the root cause service using trace data, and MULAN [2], Eadro [1] and TrinityRCL[7] utilize multi-modal data to pinpoint the root cause. Some works [1], [9], [2] produce service-level root causes, whereas others identify more fine-grained root causes, such as resource type [7], [10] or code region [7]. The fragmentation of current research efforts can be attributed to the varying targets of different teams involved in the software development process [11], particularly during incident management.

1) The SRE (Site Reliability Engineering) team focuses on detecting service anomalies and quickly taking action to mitigate the impact of failures, reduce financial losses, and identify the root cause(service) of the issue. Once the root cause is determined, they notify the relevant developer team to resolve it.
2) In contrast, the developer teams are not only concerned with identifying the problematic service but also with pinpointing the specific component within the service that is causing the issue. This involves narrowing down the potential sources, such as middleware, databases, configurations, or code logic.

These ad-hoc formulations in RCA often lead to inconsistencies, as various teams may define the 'root cause' differently, interpreting it as distinct elements within the system. Specifically, the root cause varies, it can be any component within the system's hierarchical structure, or the specific component within the service that is causing the issue (from which service, which attribute, to which lines of code).

RCA aims to streamline failure diagnosis and mitigation. However, the varying objectives of different teams have led to ad-hoc solutions that achieve localized rather than holistic optimizations. To understand why current research struggles to align with a unified goal for RCA (with unified input type and unified output type), we identified three key gaps by adhering to one guiding principle: RCA seeks to minimize the vast search space of an incident—both in identifying what caused the incident and understanding how it occurred.

Gap1: Observation Blind Spots. More granular data is essential for fine-grained root cause identification; however, real-world limitations, such as storage and computation constraints, often lead to data sampling, creating observation blind spots. [12]. The incompleteness of monitoring infrastructure can lead to missing telemetry data, resulting in metric-only based RCA methods [6], [13], [14], [15], trace-only based RCA [9], and log-only based RCA [8]. Because of this, the task formulations of previous papers are often ad-hoc, without considering the generalizability, thus making the RCA task not unified.

Gap2: Effectiveness/Performance Concern. Due to varied task formulations, different modeling approaches, even with identical telemetry data, produce inconsistent performance. This phenomenon is often influenced by how well the model-

ing methods incorporate domain-specific understanding, which decides the granularity and interpretability of output. For instance, Eadro [1] utilizes multi-modal data to enhance RCA precision, achieving over 90% accuracy in localizing service-level root causes, while Nezha [10] can localize more fine-grained root causes at the code region and telemetry resource type levels. The misalignment in modeling approaches—where inputs and outputs do not consistently align—further contributes to the fragmentation of the RCA task, preventing it from being addressed as a unified problem.

Gap3: Algorithm Efficiency, Scalability, and Robustness The efficiency and scalability of models are crucial, especially when handling large observational datasets in real-world contexts. Effective RCA applications require algorithms that can scale with data growth while remaining robust to ensure reliable outcomes, even with missing or incomplete data. Current practices are often scenario-specific, limiting the generalizability and the applicability of RCA models across diverse contexts.

Previous surveys in the field, such as the one by Soldani et al. [16] and Zhang et al. [17], have not approached root cause analysis (RCA) from this particular perspective. Rather, they tend to focus on categorizing the various models and algorithms applied under different input conditions. While useful, this method does not align with the inherently multi-objective nature of RCA, which may limit a comprehensive understanding of the field's overarching goals and future research directions. In contrast, our survey adopts a more goal-oriented framework for RCA. Specifically, we break down a primary objective—accelerating failure diagnosis and mitigation—into a series of more granular sub-goals: ensuring RCA results are *accurate*, *fine-grained*, and *interpretable*. These sub-goals are further linked to specific challenges in the field, and we examine how recent works have addressed these challenges. By organizing the discussion around specific objectives, we aim to provide readers with a clearer and more holistic understanding of the current state of RCA research and its future possibilities. This approach helps readers quickly grasp what the field is currently achieving and highlights what can be pursued in the future. To this end, this work makes the following contributions:

- *Uncovering problem nature and challenges*. We identify that the primary reason for the ad-hoc nature of these approaches lies in the inherent coupling between methods and task modeling. This coupling fundamentally stems from the fact that these papers are primarily focused on solving specific problems rather than defining the research topic from a broader perspective. We also outline the fundamental challenges in RCA research.
- *Categorization and formulation*. We categorize existing approaches based on their objectives and formulate the problem in a structured manner.
- *Comprehensive literature review of RCA*. We provide an exhaustive review of the literature on RCA, highlighting key advancements and gaps.
- *Summary of public datasets and open-source tools for Root Cause Analysis*. We provide a navigation for researchers and practitioners interested in the field.
- *Future Opportunities*. We identify potential future directions and opportunities for further research in the field.

The structure of the paper is illustrated in Fig. 1, and the detailed survey methodology is presented in Section IV.

II. BACKGROUND

A. MICROSERVICE

The microservice architecture divides a single application into a collection of small, lightweight services, each running in its process and communicating via lightweight methods, often using an HTTP API. It emphasizes agile, DevOps practices, decentralized data management, and governance [18].

Microservice architecture is supported by a series of infrastructure systems and techniques that work together seamlessly. It begins with microservice development frameworks like Spring Boot [19] and Dubbo [20], which facilitate the creation of microservices by providing essential functionalities such as REST clients, database integration, externalized configuration, and caching. Once developed, these microservices are deployed using containerization tools like Docker [21], which enhance portability, flexibility, efficiency, and speed. To manage these containers effectively, runtime infrastructure frameworks such as Spring Cloud [22], Mesos [23], Kubernetes [24], and Docker Swarm [21] 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 like Jenkins [25] and GitLab CI/CD [26] are used to support ongoing integration and delivery efforts.

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 [27].

However, as systems transition from monolithic to microservice architectures, complexity shifts from internal code to interactions between services. Modern applications, typically involving hundreds of interconnected services, significantly complicate monitoring efforts. Distinguishing between individual service failures and cascading effects from other service failures becomes challenging. Additionally, many microservice failures originate from external environments, like their runtime environments, communication, or coordination issues. These factors often obscure the root causes of failures, further complicating the detection and diagnosis of issues.

Various solutions have been proposed to automate the detection of failures and determine their root causes [28], [8], [10]. However, current definitions of Root Cause Analysis (RCA) are often case-specific, leading to significant differences in inputs and outputs across different works. Additionally, the outputs of these RCA methods are usually coarse-grained, such as service or resource level. This makes them non-actionable

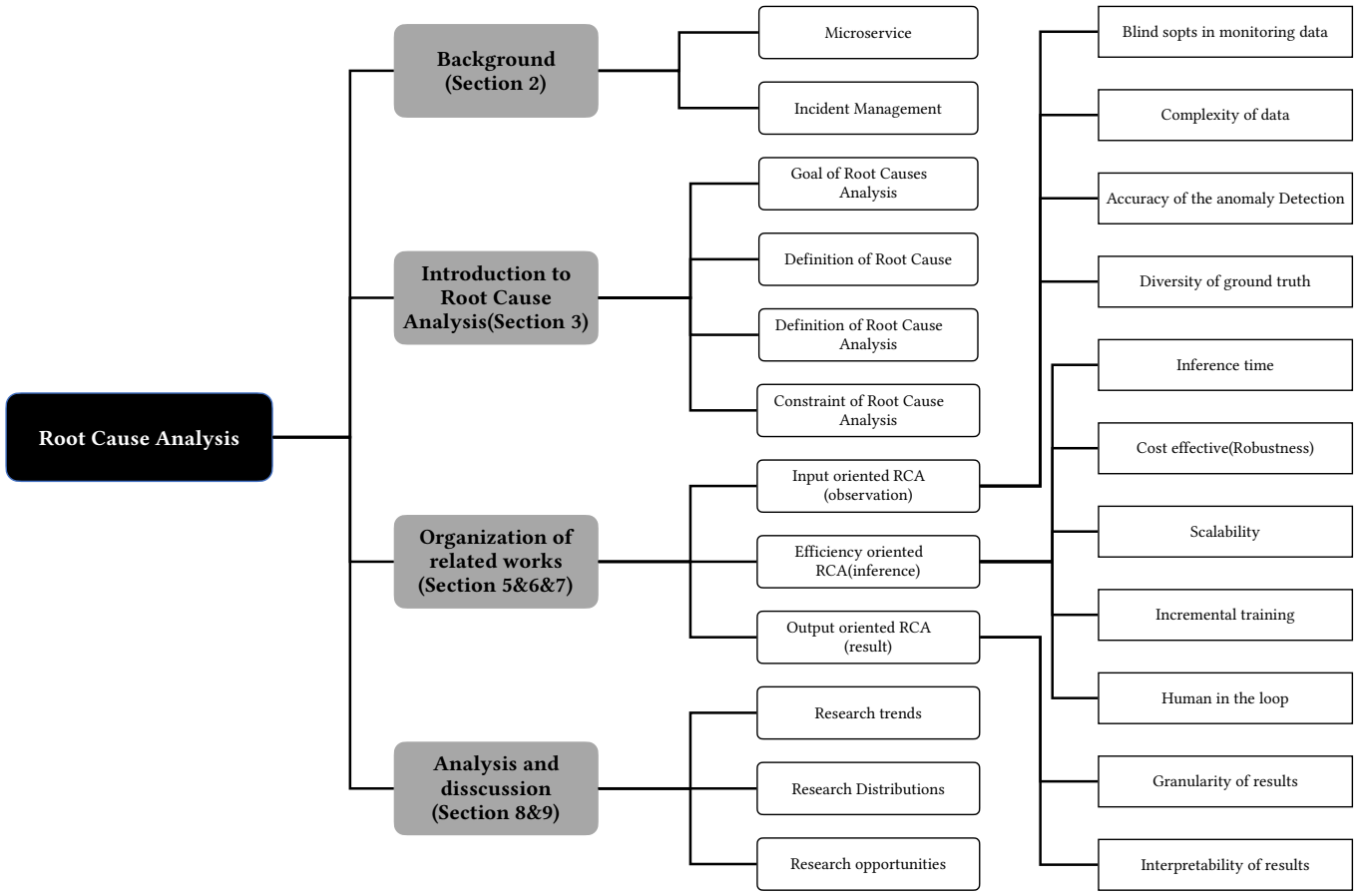

Fig. 1. Structure of the paper

and less valuable for practical industrial applications. Furthermore, many papers treat Anomaly Detection (AD) and Root Cause Analysis (RCA) separately [29]. This separation hinders application operators who aim to create a unified pipeline for anomaly detection and root cause identification in multi-service applications. To address these challenges, we propose a general definition and pipeline for microservices RCA, which integrates both AD and RCA tasks, overcoming the lack of comparability, and non-actionability of previous methods, thus making them more applicable to industrial scenarios.

B. INCIDENT MANAGEMENT

As illustrated in Fig.2, incident management primarily consists of three phases: preparation, emergency response, and review. Drawing on Google's incident management practices [30], we further break down the process into six stages: incident prevention, detection, localization, mitigation, resolution, and improvement. Incident prevention 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. 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 fully. Once the system is fully restored, the final phase is a postmortem review, which involves analyzing the incident to extract lessons learned and implementing measures to prevent future occurrences.

Building on the abovementioned incident management process, we can see that Root Cause Analysis (RCA) is critical during the incident localization and resolution stages. RCA is essential for accurately diagnosing the underlying issues that trigger incidents, enabling effective mitigation, and ensuring a thorough resolution. The ultimate goal of incident management is to reduce the Mean Time to Repair (MTTR) and extend the Mean Time Between Failures (MTBF). To achieve this, RCA must identify how to mitigate and resolve incidents quickly. Therefore, RCA itself needs to be swift, with a low rate of false positives and false negatives (avoiding misdirection of SREs and ensuring no root causes are overlooked), and it must be interpretable (so that SREs can rapidly understand the incident and decide on the appropriate actions for mitigation

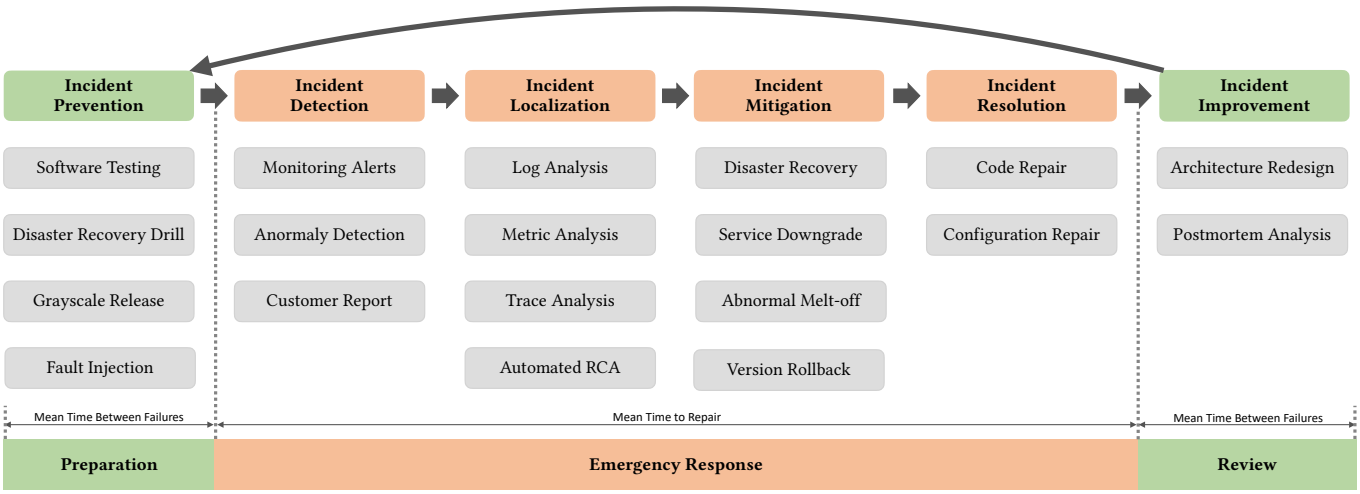

Fig. 2. Incident Management Loop

and resolution).

However, previous research, studies, and surveys have not fully defined Root Cause Analysis from this perspective. For example, a recent survey by Soldani et al. [16] mentions that RCA involves identifying the reasons behind an application or service-level anomaly, to explain the corresponding failure. This definition, however, does not clarify the inputs and outputs of RCA, nor does it explore its relationship with downstream tasks like mitigation. Some papers attempt to define RCA but are often limited to specific contexts. For instance, ChangeRCA [5] formalizes identifying root cause changes using a parameterized model, stating that the goal of RCA is to identify the Root Cause Change. They introduce a new task, Root Cause Change Analysis (RCCA), but focus exclusively on code changes, overlooking other types of changes. Similarly, Zhang et al. [17] treat RCA as a ranking problem, where the root cause service or instance is ranked higher than others, or the root cause component is prioritized over other components. However, this approach does not consider the primary objective of RCA in terms of rapid incident mitigation and resolution.

III. PRELIMINARIES

A. TERMINOLOGIES

a) Goal of RCA: The primary objective of Root Cause Analysis (RCA) in microservices is to enable Site Reliability Engineers (SREs) to swiftly and efficiently identify both the "what" and the "how" behind a given anomaly. This involves determining what specifically caused the current situation and understanding the mechanisms that led to the anomaly. Additionally, RCA aims to provide actionable insights on how to resolve the issue. The results of the analysis should be easily verifiable, ensuring that the conclusions drawn and the recommendations provided can be confirmed and validated with confidence.

b) Telemetry Data: Telemetry data is inherently *heterogeneous*, *multi-source*, and *multi-dimensional*, capturing diverse aspects of system behavior, including logs, metrics, traces, events, alerts, and profiling information. Despite all describing the same runtime system, each type of telemetry data focuses on different components, offering a *fragmented* but *complementary* view of the system's operation. Due to the distributed and complex nature of microservices, telemetry data can sometimes be *inaccurate* or *incomplete*, making it difficult to obtain a fully reliable depiction of the system state at all times. Moreover, failure-related data is typically *sparse*, complicating efforts to detect and diagnose issues. The volume of telemetry data generated is often exceptionally *large*, particularly in large-scale systems, which poses significant challenges for real-time processing and analysis. Additionally, telemetry data is highly *dynamic*, continuously evolving as the system operates, and reflecting rapid changes in system behavior. These characteristics highlight the complexity of telemetry data, necessitating sophisticated techniques for effective monitoring and analysis.

c) Incident: An incident encompasses four stages: Detection, Triaging, Diagnosis, and Mitigation [4], [31], [32]. It refers to an overall description of a service's performance when the service quality does not meet expectations.

d) Event: An event describes a change in a resource or object. It includes a timestamp, a context (location), an action, and an entity, indicating what happened, where, when, and who.

e) Root cause: Similar to telemetry data, root cause is also complex. The complexity of root cause comes from the fact that it is a relative and abstract concept, and in different contexts, it can have different meanings. For example, in the context of a cloud incident, the Site Reliability Engineering (SRE) team might prioritize identifying which service acted as the root cause of cascading failures that impacted other dependent services. Their focus is typically on diagnosing system-level interactions and understanding which service failure propagated anomalies throughout the infrastructure. However, from a developer's perspective, the root cause could be more granular. Developers are often more interested in pinpointing the specific component, such as a misconfigured library or faulty code segment, that triggered the service's

failure. Thus, while the SRE team seeks to understand which service disrupted overall operations, the development team focuses on identifying the internal component responsible for the service disruption.

f) Root Cause, Trigger, and Result: A root cause and a trigger are specific types of events. An event can serve as both a root cause and a trigger, although they pertain to different aspects. For instance, Example III-A0f illustrates a configuration change as the root cause and an unusually high volume of requests as the trigger, leading to an Out of Memory (OOM) result. The key difference between a root cause and a trigger is that there can be a significant time gap between the root cause and the resulting incident, whereas the trigger and the result typically occur in closer succession.

Illustration of Root cause and Trigger [33]

Incident name: Out of Memory (OOM)
Root cause: A config file change that introduces a memory leak
Trigger: A surprisingly high volume of requests
Incident: OOM

g) Incident Propagation Graph.: An Incident Propagation Graph is a composite of single or multiple failures, illustrating the relationships between events. It indicates which events are root causes and which are triggers, thereby mapping the complex interactions and dependencies between events during an incident.

$$\text{Root Cause} \xrightarrow{\textbf{trigger(optional)}} \text{Result} \tag{1}$$

The basic unit of the Incident Propagation Graph is called the *incident propagation chain*, as illustrated in Equation 1. We use chemical reaction equations to illustrate the chain, which contains a root cause, a trigger, and a result. For example, a configuration change (root cause), combined with a high volume of requests (trigger), causes another observed result: OOM III-A0f. The trigger is optional since, in some cases, the root cause and trigger are the same. For example, a wrong deployment command can lead to the server being down; in this case, the deployment operation can be described as the root cause, resulting in the server being down. The Incident Propagation Graph combines many incident propagation chains that collectively describe the changing system.

B. DEFINITION OF RCA

The standard Root Cause Analysis (RCA) process, as shown in Fig. 3, comprises three key parts: Observation, Inference, and Result.

a) Observation: The Observation part entails collecting and telemetry data from various sources. In this service runtime model, the inputs to RCA include a dependency graph of **monitoring units**, where each unit represents an individual service along with its **static code**, **configurations**, and **runtime resources**. Once deployed with **global configurations**, these services produce **telemetry data**, which consists of logs, traces, metrics, and events. In addition to real-time data, **historical telemetry data**, and **incident tickets** are often employed to compare with current observations, such as identifying failure patterns based on past behavior [6]. This data is crucial for understanding service behavior across different scenarios. It is important to note that capturing a complete observation is nearly impossible due to several constraints. Factors such as storage limitations, data loss during transmission, and bugs in monitoring tools all impact the comprehensiveness of telemetry data. Furthermore, observations are highly dynamic. Changes in the code, configuration, or deployment architecture can significantly alter the observed telemetry data, rendering previously identified patterns obsolete, and finding the root cause change is also studied [5], [34].

b) Inference: The Inference part involves analyzing the collected data to pinpoint potential root causes of incidents. Various RCA techniques, including machine learning, deep learning, and heuristic methods, are applied to process telemetry data, software changes, and operational data. The performance of these methods is typically measured using metrics such as recall, precision, F1 score, and their variations. The objective is to localize root causes at different levels of granularity and interpretability, providing actionable insights into the underlying issues. As mentioned earlier, the highly dynamic nature of the input data presents a significant challenge. To maintain effective performance, RCA algorithms must be robust enough to handle changes in data patterns and mitigate the effects of data loss, while adapting to the evolving system environment.

c) Result: The Result part ideally produces an incident propagation graph that visualizes the relationships between root causes, triggers, and outcomes across one or multiple failures. This graph highlights multiple potential root causes and associated triggers, offering a comprehensive view of how incidents propagate through the system. For example, a configuration change (root cause) may not immediately cause an incident, but when combined with an increase in the number of requests (trigger), it could result in an out-of-memory (OOM) condition (outcome) III-A0f. The OOM condition, in turn, may lead to additional problems, such as a rise in error rates or the introduction of new issues following the deployment of features, creating a complex chain of causality.

The incident propagation graph identifies the ultimate root causes and their triggers and provides a clear, interpretable visualization for both operators and developers. Operators can address triggers quickly to mitigate immediate incidents, while developers can focus on resolving the underlying root causes. Additionally, the graph structure supports root cause identification at various levels of granularity, from coarse to fine, helping connect disparate root cause candidates into a cohesive understanding of the incident.

C. RCA CONSTRAINT (CEO CONSTRAINT)

In the definition of RCA, we describe an ideal framework that involves the comprehensive input, and expected result format. However, this target is not easy to realize. This is because in the context of RCA for microservices, a chain constraint exists involving three critical aspects:

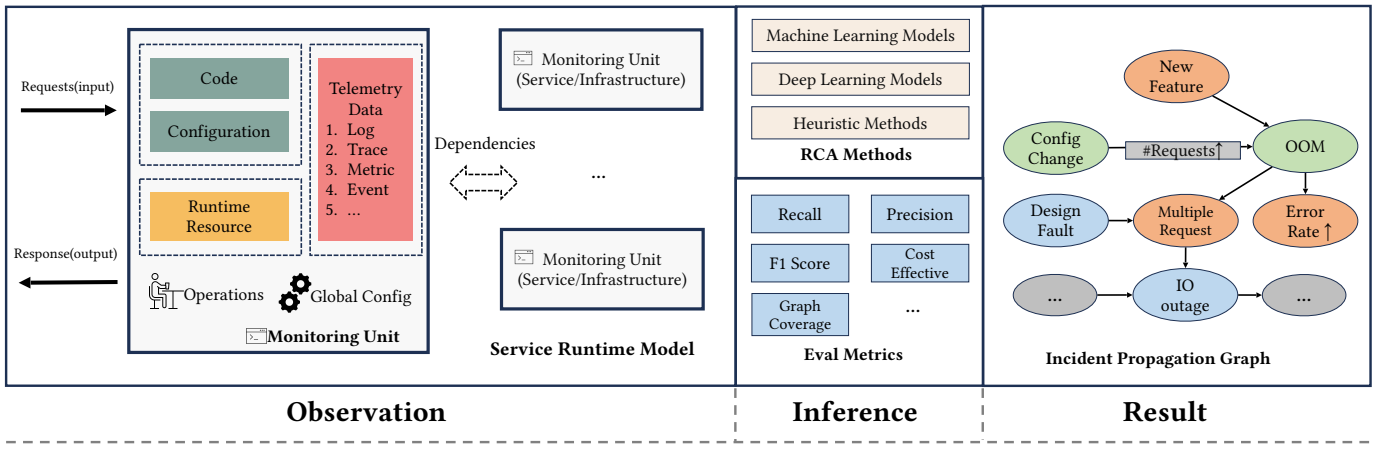

Fig. 3. Overview of Root Cause Analysis

- Computational Costs (**C**): The resources required for processing and storing observational data.
- Effective RCA (**E**): The accuracy and efficiency of identifying the root causes of issues, are described as the goal of RCA in the terminologies.
- Observational Data (**O**): The quantity and quality(including the granularity, accuracy etc.) of telemetry data available for analysis.

The RCA Chain Constraint posits that achieving Effective RCA (**E**) requires sufficient Observational Data (**O**), which increases Computational Costs (**C**). Thus, a trade-off exists between these aspects: (1) Effective RCA (**E**) necessitates more Observational Data (**O**); (2) More Observational Data (**O**) results in higher Computational Costs (**C**). Balancing these aspects involves making strategic trade-offs to achieve an optimal solution. Consequently, current research focuses on three main areas:

- *Introducing new type(granularity) of observation data*. Since if there is more information that can be extracted from the original observation, the model can output a more fine-grained root cause, and a more confidential explanation of the result, like how the error is propagated, and causes the current issue.
- *Making the RCA model better use the existing data, learning the complex relation between the data, and inferring the root cause among the data and even beyond.* Since we cannot get all the detailed runtime information of the microservice system, we must fully utilize the current information to infer precise and explainable results, and give feedback to the observation data collection.
- *Improve the efficiency of the model*, like improving the inference speed, scalability, robustness, and interpretability.

These three areas correspond to the gaps mentioned in the introduction. To achieve the goals of RCA and overcome its constraints, this paper categorizes current research into three perspectives: input-oriented, efficiency-oriented, and output-oriented RCA.

a) Input-oriented RCA: The input-oriented RCA perspective examines the types of inputs current methods utilize, the associated challenges, and the solutions proposed to address these challenges. This perspective allows systematic examination and comparison of inputs and the corresponding methods. For example, unsupervised methods are considered input-oriented as they target large amounts of unlabeled data in daily operations.

b) Efficiency-oriented RCA: The efficiency-oriented RCA perspective focuses on the model's efficiency, such as inference speed (essential for quick incident response by SREs), incremental training (reducing training overhead), robustness(robust to data loss), and model generalizability (adapting to changing observational data over time).

c) Output-oriented RCA: The output-oriented RCA perspective focuses on the outputs generated by different methods. It categorizes and analyzes research based on output types, challenges in generating these outputs, and approaches to overcome them. This perspective helps understand the effectiveness of RCA methods in achieving desired outcomes. For example, methods that produce more interpretable and fine-grained data are considered output-oriented as they focus on providing detailed and explainable information for downstream tasks like mitigation.

Note that papers can focus on multiple challenges, we discuss them separately in each section.

IV. SURVEY METHODOLOGY

This section outlines the scope and the paper collection process for our survey.

A. 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"[35], "why intermittent slow queries occur in databases"[36], and "why failures happen in microservices" [15], [2], [3].

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 such as Service Level Objectives (SLOs). These systems are often characterized by complex environments with intricate service interactions. We exclude papers centered on fault localization, as this is a more specific task that involves identifying vulnerable parts of the code.

B. PAPER COLLECTION

We collected RCA papers from various top conferences and journals, including the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), the ACM SIGSOFT Symposium on the Foundations of Software Engineering/European Software Engineering Conference (FSE/ESEC), the International Conference on Very Large Data Bases (VLDB), the International World Wide Web Conference (WWW), the International Conference on Software Engineering (ICSE), the IEEE International Conference on Software Testing, Verification and Validation (ISSTA), and the International Conference on Automated Software Engineering (ASE), among others.

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 [37]. 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 opensource Kubernetes [24]. Through this comprehensive search methodology, we identified and collected 106 top papers directly related to Root Cause Analysis.

C. PAPER ANALYSIS

To ensure a thorough and rigorous analysis of the collected papers, the first two authors undertook an extensive reading and examination of the full text of each paper. This process aimed to extract comprehensive information on several key aspects of the research presented. Specifically, for each paper, we identified the scenarios addressed by the research, the inputs and outputs of the proposed methodologies, and the evaluation methods used to assess the results. Additionally, we documented the datasets utilized in the study, noted whether the code and datasets were available as open-source, and highlighted the specific challenges the research aimed to tackle. Finally, we detailed the methods or models employed in the research.

Through careful manual analysis, the first two authors extracted and organized relevant information, identifying recurring patterns and themes across the papers. These themes were then used to categorize and systematically organize the papers, providing a coherent structure for the survey. This thematic organization helps readers navigate the survey and understand the key insights and findings derived from the analyzed papers.

When the two primary authors had differing interpretations or findings, they conducted discussion sessions with additional co-authors. These discussions were instrumental in resolving disagreements and ensuring a consensus on the extracted data and the categorization of papers. 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.

V. CHALLENGES OF INPUT

In this section, we discuss the challenges associated with input data for RCA. As illustrated in Sec. III-C, achieving effective RCA requires sufficient observational data, which increases computational/storage costs. Starting from the constraint, we divide the challenges of input into two main parts: (1) The first part deals with the case that there are always blind spots that we cannot observe due to various reasons (Section V-A). (2) The second part focuses on how to model the complexity of the input since the input data of RCA can be heterogeneous, multi-source, and multi-dimensional (Section V-B). They all describe the same runtime system, but each focuses on different components. Unlike the first part, this part discusses the challenges encountered when most observational data are collected without considering the computational/storage constraints.

Additionally, since Anomaly Detection (AD) is a precursor to the RCA process, the quality of AD results is crucial for ensuring accurate RCA. In Sec. V-C, we presented previous works addressing AD inaccuracies.

A. REDUCING BLIND SPOTS THROUGH INFERRING UNOBSERVED DATA

The limitations in the input data arise from two main factors. The first is the absence of topology information. Since failures propagate across services, tracing error propagation typically requires constructing a graph to infer dependencies. However, in some cases, the service topology (or the associated tracing data) is either unavailable or difficult to obtain. For instance, there may be a lack of appropriate monitoring tools, leaving

Category	SubCategory			Paper
Blind Spots	Causal Discovery	Constraint-based	PC-based	[38], [39], [28], [40], [41], [42], [43]
			Granger-based	[44], [45], [46], [47], [48], [49], [50]
			Other	[51], [52], [53]
		Learning-based		[54], [2], [55], [56], [11], [57], [58]
	Causal Inference			[59]
Data Complexity	Intricate Service Dependencies			[5], [57], [60], [61], [38], [62], [63], [64], [42], [65], [66], [47], [67]
	Multidimensional Data Complexity			[68], [13], [40], [2], [63], [69], [70], [71], [72], [67], [38], [46]
	Data Path Ambiguity			[73], [74], [75], [76]
	Scarcity of Incident-Related Data			[9], [77], [72]
	Difficulty in Obtaining Labeled Data			[78], [29], [55], [15], [59], [8], [10], [56]

TABLE I
CHALLENGES OF INPUT

only service metrics as available data. In such cases, inferring or constructing possible relationships based on the observed data (e.g., metrics) is necessary. The second factor is the incomplete availability of telemetry data. For example, direct telemetry data for a specific component under monitoring may be missing, leaving only indirect observations, such as data from third-party services. In such scenarios, causal inference techniques are employed to infer the unobserved data.

1) Construct Graph by Causal Discovery: In microservice architectures, dependencies between services can cause issues to propagate along service chains, leading to unpredictable degradation in Quality of Service (QoS). Therefore, identifying these dependencies is crucial for accurately diagnosing the root causes of microservice issues. By learning the causal structures within microservice topologies, we can capture these dependencies and infer how disruptions in one microservice affect the entire system. [55]. Causal discovery methods can be divided into constraint-based and learning-based approaches. Constraint-based methods excel in smaller, well-defined microservice systems with clean data, offering high interpretability through statistical tests [51]. In contrast, learning-based methods are better suited for large-scale, dynamic environments with complex, nonlinear dependencies, handling noisy or incomplete data more effectively and scaling to higher dimensions [56].

Statistical-based graph construction. Statistical-based graph construction refers to a set of techniques that use statistical methods to infer causal relationships between different components in a system. By analyzing data—most commonly time series metrics—these methods construct causal graphs that represent how variables influence one another. These graphs help in diagnosing system failures by identifying potential root causes through causal dependencies.

PC-based methods. The PC (Peter-Clark) [79] algorithm is a widely used constraint-based method that constructs causal graphs step-by-step through a series of conditional independence tests. It systematically removes edges between conditionally independent variables, leaving only the most likely causal connections. This method is particularly effective in high-dimensional data environments, such as large-scale microservice systems with complex operational metrics. Examples of PC-based methods include Microscope [38], CloudRanger [39], CauseInfer [28], MicroCause [40], HRLHF [41], ServiceRank [42], and CloudRCA [43]. MicroCause [40] improves the traditional PC algorithm by addressing its limitations in handling time series data. The PCTS (Path Condition Time Series) algorithm enhances the PC algorithm by capturing the propagation delays between different monitoring metrics, which the original PC algorithm overlooks by assuming data is independent and identically distributed (iid). Additionally, PCTS simplifies the causal graph by representing each node as a monitoring metric, rather than individual time points, making the graph more manageable for root cause analysis.

Granger-based methods. Granger causality is primarily used for time series data. It tests whether the past behavior of one microservice helps predict another, allowing the identification of time-dependent causal relationships. For example, DyCause [44] uses Granger causality testing to analyze dependencies between microservices. It collects performance metrics like request latency, and when an anomaly is detected, it applies a sliding window to the time series data. In each window, Granger causality testing checks if the historical data of one service can improve the prediction of another service's future state. If so, it indicates a causal relationship between the services, helping to build a dynamic causality map over time. This method is beneficial when the system involves microservices whose interactions evolve over time. Granger-based techniques include DyCause [44], Sieve [45], MicroCU [46], FRL-MFPG [47], LOUD [48], GrayScope [49], and TS-InvarNet [50].

Other methods. Chain-of-Event [51] uses the Chain-of-Event (CoE) model for causal discovery, automatically learning causal relationships between events from historical data. It first generates a naive event-causal graph (NEG) that connects all potentially related events. Then, the CoE model automatically learns the causal link weights and event importance scores, avoiding manual configuration and improving accuracy. By tracing event chains, the model calculates root cause scores for each event, identifying the most likely root cause. CauseRank [52] introduces G-GES (Group-based Greedy Equivalent Search), a causal discovery algorithm that is part of the CauseRank method, designed to tackle performance diagnosis challenges in Online Transaction Processing (OLTP) systems.

G-GES groups related metrics and uses these groups as nodes to build a simpler causal graph. By incorporating domain knowledge, such as DBAs' expertise and known system dependencies, G-GES improves the accuracy of identifying failure paths. It addresses three key challenges: 1) managing large numbers of abnormal metrics, 2) modeling complex failure propagation, and 3) ensuring efficient and accurate fault localization. In IPCCC'16 [53], the initial causal graph is built using the FP-Growth algorithm. The system mines historical monitoring data to find associations between symptom events, identifying potential causal rules. These frequent co-occurrences form the initial causal graph, which is then used for root cause analysis. At this stage, the graph is purely data-driven, relying on patterns in historical data without requiring prior knowledge.

Learning-based graph construction. Traditional methods like the PC algorithm fail to handle cyclic dependencies and unclear relationships common in enterprise networks and microservices. Murphy [54] employed Markov Random Field [80](MRF), a probabilistic model that allows cycles and bidirectional dependencies in the graph. By learning joint distributions from historical data and applying Gibbs sampling for inference, the MRF model effectively captures complex interactions, improving the accuracy of root cause analysis in these environments. MULAN [2] uses the Vector Autoregression (VAR) model to capture dynamic causal relationships between system entities from historical time-series data. The VAR model is well-suited for handling multi-modal data, like system logs and performance metrics, as it can model the interactions between different data sources. It also predicts future values based on past observations. Causal Bayesian Networks (CBN) are graphical models that represent causal relationships using Directed Acyclic Graphs (DAGs), where nodes are variables and edges show causal dependencies. CBN constructs causal graphs by analyzing data or system structures, leveraging conditional independence to simplify complex relationships. CBN-based techniques include Sage [55], Sleuth [56] and ExplainIt [11]. The Topological Causal Discovery (TCD) in REASON [57] uses hierarchical graph neural networks to construct causal graphs between system entities. It captures both within-level and cross-level causal relationships, creating a comprehensive map of how system components are interconnected causally. CORAL [58] uses an Incremental Disentangled Causal Graph Learning method for causal discovery. It separates causal relations into "state-invariant" and "state-dependent" types. As new data comes in, the causal graph is updated incrementally, adjusting only the state-dependent relations while keeping the state-invariant ones unchanged. This approach efficiently handles real-time data, enabling quick updates to the causal graph for root cause identification.

2) Infer Beyond Observation by Causal Inference: Previous causal inference methods rely on good observability [15], [59], [55], [54], [56], [14]. For example, CIRCA [15] uses Regression-based Hypothesis Testing (RHT) to detect causal relationships between metrics and identify faults. However, its effectiveness is limited because it requires all relevant variables to be observable, which is often not feasible in complex microservice environments where data can be missing or incomplete.

LatentScope [59] overcomes the limitations of previous causal inference methods. In complex microservice environments, full visibility is often unfeasible due to missing or incomplete data. To address this, LatentScope models root cause candidates (RCCs) as latent variables and infers their states through related observable metrics. It constructs a dual-space graph that separates latent RCCs from observable metrics, allowing it to effectively manage heterogeneous and unobservable RCCs. LatentScope also introduces the Regression-based Latent-space Intervention Recognition (RLIR) algorithm, which quickly infers and identifies root causes by analyzing the causal relationships between latent and observable variables. While Sage [55] also mentions latent variables, it primarily uses them to capture system randomness, such as server performance fluctuations or network delays, and still heavily depends on observable data for causal inference. This reliance means that Sage's effectiveness diminishes significantly when data is incomplete or missing. In Sage, latent variables are mainly used to explain performance fluctuations rather than to solve the problem of unobserved root causes. In contrast, LatentScope places latent variables at the core of its causal inference process. By modeling unobservable RCCs as latent variables and inferring them with observable data, LatentScope directly addresses the challenges of missing or unobservable data. Its dual-space graph and RLIR algorithm further enhance its ability to connect and separate latent RCCs from observable metrics, providing accurate root cause analysis even with limited observability.

B. CHALLENGES DUE TO DATA COMPLEXITY

Even when sufficient observational data is available, several challenges persist due to the complexity inherent in the data. To begin with, tracing causal fault propagation is particularly difficult because of the intricate cross-layer service dependencies involved [5]. Moreover, the multi-dimensional nature of the data, including the complex relationships between system metrics and logs, further complicates the analysis process [13], [68]. In addition, complex datasets often result in the construction of graphs with ambiguous data paths, where this ambiguity in call graphs can lead to inaccurate root cause identification [73]. Furthermore, incident-related data tends to be much scarcer than normal operational data, especially within the same timeframe, making it difficult to gather sufficient information for effective analysis. Finally, even if adequate incident-related data is collected, ground truth labels for root cause analysis are typically unavailable in industry settings. Such information is usually only obtainable post-incident after the SRE team has detected and resolved the issue, and telemetry data has been stored for subsequent post-mortem analysis.

1) Intricate Service Dependencies: In microservice architectures, intricate and dynamic dependencies between services create significant challenges for anomaly detection, fault propagation analysis, and root cause identification. These dependencies are complex and multifaceted, shaped by several factors such as frequent updates, asynchronous interactions, and

the lack of centralized control. Additionally, the complexity is further exacerbated by diverse and evolving business logic dependencies, heterogeneous communication protocols, and the high degree of service autonomy. This makes it difficult to maintain accurate, up-to-date models of the system, further complicating efforts to ensure system reliability and timely issue resolution. To address these challenges, researchers have proposed various methods, which can be grouped into two main categories based on how they tackle issues related to intricate service dependencies.

Maintaining an accurate model of service dependencies. Static dependency graphs quickly become outdated, hindering effective monitoring and diagnosis. To overcome this, several approaches focus on dynamically constructing or updating service dependency models that reflect the current state of the system in real-time. ServiceRank [42] constructs an Impact Graph based on real-time monitoring data, dynamically determining causal relationships between services without relying on a static system topology. This method adapts to the evolving nature of microservices, capturing current service interactions and enhancing the accuracy and efficiency of root cause identification. FacGraph [66] captures the system's changing topology during both normal and abnormal states by using correlation graph construction. Employing a PC-algorithm-based correlation graph, FacGraph dynamically maps out service dependencies as they evolve and utilizes Frequent Subgraph Mining to identify recurring anomaly patterns. This approach allows for pinpointing root causes without constant centralized monitoring, reducing overhead and improving efficiency. Microscope [38] automatically discovers service dependencies by analyzing network-related system calls, such as `socket()` and `connect()`. This method accurately captures real-time dependencies without modifying source code or relying on potentially error-prone statistical inference. By directly monitoring system interactions, Microscope [38] ensures precise dependency mapping, enhancing the effectiveness of performance diagnostics. FRL-MFPG [47] proposes the Microservice Fault Propagation Graph (MFPG-FC), which combines current microservice dependencies with historical fault data to create a propagation graph that accurately represents fault paths and their impacts. By integrating historical patterns, FRL-MFPG [47] ensures accurate modeling of fault propagation even as the environment evolves, improving the stability and efficiency of fault diagnosis. ChangeRCA [5] integrates service dependency analysis with layered differential analysis to track both the direct and indirect effects of service changes. By comparing behaviors before and after changes and incorporating historical data, ChangeRCA effectively identifies root causes that might be missed by methods focusing solely on individual service changes. This comprehensive approach provides more accurate fault localization and insights into the system's underlying interactions.

Tracing anomaly propagation through service dependencies. The service dependency is complex and indirect at some times, including asynchronous interactions, caching, and queues. Anomalies can spread via indirect paths, making propagation unpredictable and difficult to model with traditional methods. To address this, several approaches focus on modeling these complex interactions and anomaly propagation paths. MicroRCA [64] employs attributed graphs to model anomaly propagation paths, capturing both service call relationships and shared host dependencies. This comprehensive view allows MicroRCA to focus on critical nodes in the propagation path, narrowing down the scope of root cause analysis and managing complexity effectively. AutoMAP [63] utilizes an Anomaly Behavior Graph to dynamically generate service correlations and map out intricate propagation routes, including indirect paths through shared resources. GAMMA [62] addresses bottleneck detection challenges due to complex interactions by employing an attention-based Graph Convolutional Network (GCN). This method effectively captures dependencies arising from asynchronous calls, caching, and queues. By weighting information from neighboring services, the GCN focuses on critical interactions, enabling accurate and efficient identification of bottlenecks. TraceAnomaly [60] uses Deep Bayesian Networks to learn normal trace patterns, handling complex dependencies without supervision. By creating a unified trace representation that encodes both invocation paths and response times, TraceAnomaly [60] considers the entire trace context, improving anomaly detection accuracy and effectively dealing with intricate service call relationships. Wu et al. [61] propose a two-stage approach based on a service dependency graph to diagnose performance issues in highly distributed microservice architectures with complex dependencies. The first stage identifies potentially problematic services, while the second stage employs an autoencoder to detect specific anomalous metrics related to service abnormalities. This approach captures relationships between services and precisely identifies abnormal metrics through reconstruction errors, aiding in pinpointing specific performance bottlenecks. REASON [57] introduces a hierarchical Graph Neural Network (GNN) to capture both intra-level and inter-level non-linear causal relationships in interdependent networks spanning multiple levels. This enables accurate modeling of fault propagation across different layers of the system, enhancing the precision of root cause analysis by effectively handling the complexities of multi-layer dependencies. MonitorRank [65] combines a random walk algorithm on the call graph with pseudo-anomaly clustering to handle unreliable call graphs and dynamic dependencies. The random walk explores possible root causes without assuming static dependencies, offering flexibility in handling changing system interactions. Pseudo-anomaly clustering identifies correlations caused by external factors, supplementing missing information in the call graph. This combination makes root cause identification more precise and adaptive to the dynamic nature of microservices.

2) Data Complexity: In microservice architectures, dealing with the complexity of multidimensional data is another significant challenge. These complexities arise from multidimensional data and varying data modalities.

a) Multidimensional Data: Multidimensional data refers to various system-level performance metrics, including CPU usage, memory usage, network input/output, and disk read-/write [69]. These metrics together represent the system's overall performance. Handling these data presents a significant challenge in fault diagnosis due to the complexity of relation-

ships between various metrics. Several approaches have been proposed to address this challenge by leveraging dependency modeling, dynamic adjustment, and automated learning.

Dependency Modeling and Adaptive Learning MS-RANK [70] models relationships between multiple performance metrics using a dependency graph. By dynamically adjusting the weight of each metric through adaptive learning, it captures evolving dependencies, leading to more accurate diagnoses. FChain [69] learns the normal fluctuation patterns of these metrics and identifies abnormal changes. FChain then uses these changes, combined with the propagation paths and inter-component dependencies, to accurately pinpoint the faulty component. Similar to MS-RANK, it models dependencies between multiple metrics, allowing it to handle complex multidimensional data and improve fault localization without relying on predefined dependency graphs. CMMD [13] takes this further by using Graph Neural Networks (GNNs) to automatically learn complex dependencies between metrics and services from historical data. Unlike MS-RANK, CMMD doesn't rely on predefined dependency graphs. It also employs genetic algorithms to efficiently identify anomalous metric combinations, improving root cause detection.

Time Series and Causal Analysis MicroCU [46] uses a time-sensitive approach, dividing the anomaly period into intervals and applying Granger causality tests to capture changing dependencies between services. It corrects sparse data through causal unimodalization, reducing bias and improving fault propagation accuracy. FluxRank [71] addresses the challenge of handling diverse metrics in real-time. Using a non-parametric approach, it combines absolute derivatives and Kernel Density Estimation (KDE) to efficiently detect significant changes, making failure localization more practical without complex tuning.

Pattern Mining and Dimensionality Reduction TraceContrast [68] focuses on analyzing microservice traces by treating them as sequences with multiple attributes. It uses contrast sequence pattern mining to detect rare, critical combinations of attributes linked to anomalies and applies spectral analysis to reduce dimensionality for efficient root cause detection. Microscope [38] reduces the complexity of multidimensional data by capturing real-time service dependencies through system calls. Processing fewer metrics and using statistical methods to handle non-communication dependencies, it enhances causal inference across diverse system data.

b) Heterogeneous Data: As systems became more complex and data more varied, methods needed to adapt to the heterogeneity of data types and system components. Researchers developed approaches to effectively integrate different data modalities and address system heterogeneity.

Data Integration and Representation. This category focuses on unifying different data types such as metrics, logs, and traces into a common model. AutoMAP [63] addresses this by integrating metrics like CPU, memory, and I/O into a unified model through an anomaly behavior graph. It dynamically builds relationships between services and uses weighted multi-metric learning to select the most relevant metrics based on historical data, improving diagnostic accuracy. Similarly, DiagFusion [72] leverages event embedding techniques like FastText to map logs, metrics, and trace data into a unified vector space. By using lightweight pre-processing and representation learning, it effectively integrates diverse data types, utilizing their complementary information to enhance fault diagnosis. MULAN [2] further advances this approach by using contrastive learning to extract both modality-invariant and modality-specific features from heterogeneous data such as system logs and performance metrics. It employs a KPI-aware attention mechanism to fuse these features into a unified causal graph, improving the accuracy of causal relationship modeling. MicroCause [40] uses the PCTS algorithm to integrate different types of monitoring data, such as KPIs and metrics, into a unified causal graph. It fully leverages the time delays and causal relationships between these data types to build an accurate dependency graph. Additionally, the TCORW (Temporal Cause Oriented Random Walk) algorithm combines causal relationships, anomaly information (e.g., time of occurrence and severity), and domain-based metric priorities to rank and infer the root causes of failures.

Group-based analysis. This category organizes similar elements or data types into groups to avoid direct comparisons across heterogeneous elements, thereby improving the precision of anomaly detection. WinG [67], for instance, groups elements with similar call characteristics—such as virtual machines, operating systems, and containers—to avoid direct comparisons between these different types of elements. It then evaluates anomaly scores within these groups, allowing for more precise detection and root cause localization.

In addition to the traditional telemetry data(log, trace, metric), Raccoon [34] takes the user-reported incident ticket and the change history as the input, and wants to find which change is the root cause of the incident. They design a causal knowledge representation method at the user-perceived functional level, leveraging fault tree and software product line frameworks, while incorporating multiple knowledge sources and employing a Tree GNN model for causal inference. ChangeRCA [5] further uses the telemetry data and change history to find the erroneous change.

3) Data Path Ambiguity: In microservice architectures, accurately tracing the flow of data and control across numerous interconnected services is challenging due to data path ambiguity. This ambiguity arises from asynchronous communications, lack of global transaction identifiers, interleaved logs from concurrent transactions, and redundant or irrelevant components in service dependency graphs. These issues make it difficult to construct precise models of service interactions, hindering effective RCA. To address this, TraceDiag [75] proposes a framework that employs reinforcement learning (RL) to automatically learn optimal pruning strategies. By learning from historical data, RL effectively identifies and removes redundant components, enhancing RCA efficiency and reducing manual intervention. This adaptive approach balances the pruning process to prevent both over-pruning and under-pruning, ensuring that essential components are retained while irrelevant ones are discarded.

Another significant challenge is *dealing with ambiguities in call data within complex, interdependent microservices*. Even with a well-constructed, ambiguity-free call graph, in-

correct traversal and interpretation can lead to inaccurate root cause identification. To tackle this issue, CMDiagnostor [73] integrates the AmSitor algorithm, which uses a regression-based method to accurately determine associations between upstream and downstream calls, particularly in ambiguous situations. AmSitor models the relationship between upstream and downstream traffic using linear regression, estimating weights that represent the expected number of downstream calls per upstream call. By filtering out connections with low or negligible coefficients and iteratively applying regression across different time slices of call data, it retains only those upstream-downstream pairs that consistently show strong connections. This results in a clear and accurate call graph. Building upon this, CMDiagnostor employs a series of pruning strategies—such as AmSitor-based, Metric Similarity-based, and Anomaly Detection-based Pruning—to effectively navigate the call graph. These strategies filter out false positives and focus on the most relevant paths, ensuring that the root cause localization process remains precise and effective, even in large-scale systems with complex interdependencies.

Moreover, instead of using a snapshot of the entire system for incident detection, traditional methods often *neglect the importance of issue extraction*. This approach can lead to outdated analysis, high false alert rates, and difficulties in integrating information from various sources. GIED[74] addresses this by automatically extracting the issue impact topology early in incident management, considering both symptoms and affected services. By focusing on the specific parts of the system impacted by an issue, GIED allows for more accurate incident detection and efficient root cause localization. It utilizes a Graph Neural Network (GNN)-based model and the PageRank[81] algorithm to analyze the extracted topology, improving detection accuracy and reducing false positives.

Finally, in scenarios without transaction IDs, *extracting control flow graphs from multi-threaded logs* presents a challenge due to interleaved logs from different threads. This interleaving makes it difficult to separate actual execution paths from noise, leading to confusion where logs from different transactions may appear falsely related. To overcome this, Jia et al. [76] propose a two-stage edge mining algorithm. The first stage identifies frequent successor groups—logs that frequently occur close together in time—to uncover potential true relationships. The second stage refines these groups to find immediate successors, removing redundant and incorrect links. This approach leverages statistical associations between log entries, capturing real relationships within execution paths and reducing confusion caused by interleaved logs. By broadening the search for potential connections and then narrowing it down to the most relevant ones, the method effectively minimizes noise and reconstructs accurate control flow graphs.

4) Scarcity of Incident-related Data: The lack of high-quality incident-related data can lead to several issues in both supervised and unsupervised RCA methods. In unsupervised approaches, noise in the data often poses a significant challenge, as normal patterns can be misinterpreted as anomalies due to their variability, which reduces the precision of anomaly detection. In supervised methods, an imbalance between normal and abnormal data leads to models favoring the more common normal data, which results in missed anomalies or false alarms. This imbalance can reduce the overall accuracy of both anomaly detection and root cause localization, making it difficult to effectively diagnose incidents in complex systems.

Sample scarcity is a challenge for root cause analysis in cloud computing platforms, as there are often too few failure events to train effective models. To address this, CloudRCA [43] uses cross-platform transfer learning, sharing data between Alibaba's different cloud platforms (MaxCompute, Realtime Compute, Hologres), especially for common modules like hosts and networks. By pooling data from these platforms, the method compensates for the lack of data on individual platforms, improving the model's ability to generalise and accurately identify new failures. This approach is especially helpful as cloud-native architectures become more unified

One of the major challenges MicroCBR [77] faces is the lack of incident-related data. In microservice environments, data like logs, metrics, and traces can be incomplete due to deployment configurations, security constraints, or performance requirements. This missing data makes it difficult for traditional diagnostic methods, which often rely on specific homogeneous data types, to effectively identify faults. To address this issue, MicroCBR [77] integrates heterogeneous data sources, including metrics, logs, traces, and command outputs. This approach improves data coverage by allowing available sources to fill in for missing data. Command data, in particular, serves as a customizable post-incident source, providing additional insights when other types are unavailable. By combining multiple data types, MicroCBR enhances system diagnosis, ensuring that even with incomplete data, useful insights can still be obtained, making the process more robust and effective.

In Microrank [9], the imbalance in the coverage of normal and abnormal traces in the data makes traditional spectrum-based methods less effective, as some requests frequently cover the same service instances while others do not. To address this imbalance, MicroRank [9] incorporates a personalized PageRank algorithm that assigns different weights to traces based on their ability to localize root causes. The algorithm considers both the scope of the traces (i.e., how many service instances they cover) and the frequency of different types of traces. By doing so, MicroRank [9] balances the influence of frequently and infrequently occurring traces, ensuring that the spectrum analysis accurately reflects the importance of each trace in diagnosing root causes. This approach enhances the precision of root cause localization by mitigating the bias introduced by unbalanced trace coverage.

Diagfusion [72] uses a data augmentation mechanism that generates more samples for rare failure types by randomly selecting and modifying events from the original data. This approach helps balance the training data, allowing the model to better learn the characteristics of all failure types and improving its ability to diagnose rare failures accurately.

5) Difficulty in Obtaining Labeled Data: Obtaining labeled data for RCA is inherently challenging, since there are no ground truth labels for RCA in the industry, only if a failure is detected and resolved by the SRE, and the telemetry data are

stored for post-mortem analysis. Thus, unsupervised methods are important in industry because they reduce the human effort for labeling data, and the need for storing historical telemetry data of a failure. Given the inherent difficulty of obtaining labeled data, various unsupervised methods have been proposed to address this challenge from different perspectives, such as statistical analysis [78], [29], causal inference [55], [59], and pattern mining [8], [10], [56].

Statistical Methods. Statistical methods primarily rely on detecting changes in data distributions, often without the need for pre-labeled datasets or prior assumptions about system behaviors. These methods are particularly useful in real-time settings where timely anomaly detection is crucial. Instead of relying on labeled data, ϵ-Diagnosis [78] uses unsupervised two-sample hypothesis testing and time series similarity analysis to identify root causes. By comparing abnormal and normal samples, the algorithm efficiently detects significant changes in metrics, providing a practical solution to the labeled data challenge, particularly in real-time diagnosis scenarios. BARO [29] builds on this idea by employing a more sophisticated approach for multivariate time series. It incorporates Multivariate Bayesian Online Change Point Detection to effectively detect anomalies in complex time series data without relying on labeled datasets. Additionally, BARO uses a nonparametric hypothesis testing method, RobustScorer, which is robust to inaccuracies in anomaly detection and does not require pre-labeled data.

Causal Inference-Based Methods. In contrast to statistical methods, causal inference-based approaches aim to uncover the underlying causal relationships between variables, offering a deeper understanding of how system anomalies emerge from complex interdependencies. These methods are particularly powerful when dealing with latent variables and missing information. Sage [55] utilizes Conditional Variational Autoencoders (CVAE) to model the distribution of latent variables from historical tracing data collected by cloud monitoring systems. By generating counterfactual scenarios, Sage simulates how different conditions might affect system performance, allowing it to infer the root causes of QoS violations without relying on pre-labeled data. This process involves using the observed data to train the model and then applying it to generate hypothetical cases where potential issues could arise, thus enabling the system to diagnose problems based on the learned patterns from historical data. Unlike Sage [55], CIRCA [15] addresses the challenge of limited labeled data by constructing a CBN and using regression techniques to handle incomplete fault data distributions. CIRCA [15] adjusts anomaly scores without relying on labeled data, making it effective in scenarios with scarce labeled information. This model focuses more on refining anomaly scores based on partial data, which complements Sage's focus on generating counterfactuals for hypothesis testing. LatentScope [59] addresses the difficulty by introducing unsupervised methods like the Regression-based Latent-space Intervention Recognition (RLIR) algorithm and its enhancement, LatentRegressor. Unlike traditional supervised models that require labeled data, RLIR operates in a latent space, allowing it to infer the status of unobservable variables without the need for labeled data. LatentRegressor further refines this process by using ridge regression, which is more robust to noise, reducing computational overhead and enhancing accuracy in real-world applications.

Pattern Mining-Based Methods. Unlike statistical or causal inference-based approaches, pattern mining methods focus on discovering commonalities and deviations within large datasets. These methods directly identify patterns associated with failures through automated clustering and comparison, reducing reliance on labeled data. In cloud system log analysis, the large volume and complexity of logs make manual labeling of incident-indicating logs time-consuming and error-prone. By leveraging automated clustering techniques, Onion [8] groups similar logs into "log cliques" and uses contrast analysis to compare logs from anomalous servers with those from normal servers, effectively identifying incident-indicating logs. This approach avoids the need for extensive manual labeling and, compared to traditional template-based log analysis methods, is more adaptable and maintains high accuracy and efficiency even when labeled data is incomplete. Nezha [10] takes this idea further by analyzing multi-modal data, such as logs, traces, and metrics, before and after faults occur. It constructs an event graph and compares execution patterns to detect anomalies. Nezha [10] automatically pinpoints potential root causes by detecting patterns that deviate from expected paths, thereby reducing reliance on manual labeling and handling larger, more complex datasets. Sleuth [56] enhances pattern mining techniques by adopting an unsupervised learning approach that leverages a GNN to model causal relationships between services in a microservices architecture. Then it generates counterfactual queries to identify the root causes of anomalies. This approach allows Sleuth to operate effectively without any labeled data. Moreover, Sleuth's design includes few-shot learning capabilities, ensuring it remains highly adaptable and generalizable across different microservice applications, even with minimal or no labeled data available.

C. ACCURACY OF ANOMALY DETECTION

In large-scale microservice systems, accurately detecting anomalies presents several key challenges: *parameter sensitivity in statistical methods*, *complex service dependencies*, and *diverse anomaly types*. Addressing these challenges is crucial for ensuring that anomaly detection can effectively support root cause analysis (RCA). Below, we examine how different methods tackle these issues.

Parameter Tuning and Data Limitations. Firstly, existing methods like Kernel Density Estimation (KDE) and $epsilon$-statistical tests often struggle with *parameter tuning and data limitations*, leading to inaccurate anomaly detection. PatternMatcher [82] addresses this by using a two-sample hypothesis test for coarse-grained anomaly detection, which compares data distributions before and during anomalies. This method reduces parameter sensitivity and improves detection reliability by avoiding the complex tuning required in KDE.

Complex Dependencies and Diverse Anomaly Types In addition, the *complex dependencies and diverse anomaly types* in microservices create further challenges. MicroHECL [83]

offers a solution by combining machine learning and statistical methods to handle different anomaly types more effectively. It uses OC-SVM for performance anomalies, Random Forest for reliability anomalies, and the 3-sigma rule for traffic anomalies, ensuring more precise detection across multiple anomaly types.

Complex Dependencies in Multivariate Time Series. Eadro [1] takes a step further by integrating anomaly detection and root cause localization in an end-to-end manner, using multimodal learning to create a unified system representation from logs, KPIs, and traces. By jointly optimizing both tasks, Eadro enhances detection and localization accuracy. However, Eadro's reliance on simpler detection methods like N-Sigma and SPOT limits its effectiveness in handling *complex dependencies in multivariate time series*. To overcome this, BARO [29] introduces Multivariate Bayesian Online Change Point Detection (BOCPD), which models dependency structures more accurately, improving anomaly detection in complex systems. Additionally, its RobustScorer method ranks root causes based on data distribution changes, making root cause analysis less sensitive to detection timing.

VI. CHALLENGES OF EFFICIENCY

In this section, we explore the various challenges related to the efficiency of the methods. Efficiency is a critical aspect of algorithm design and implementation, directly impacting a system's practical applicability and performance. The discussion will cover key areas, including inference time, cost-effectiveness, robustness, scalability, incremental training, and the integration of human-in-the-loop approaches. By examining these dimensions, we aim to provide a comprehensive understanding of the factors that influence the efficiency and practicality of the algorithms under the scope of RCA.

A. INFERENCE TIME

Inference time is critical in evaluating root cause analysis (RCA) algorithms. Faster inference time enables quicker response to issues, minimizing downtime and improving system reliability. Lots of works address their quick inference time, we categorize this portion as techniques based on pruning [84], [8], [45], [83], [75], [54], [85], [74], [86], [68], [73], [46], [15], [14], [59], parallel processing [68], [55], [10], [54], [15], [38].

1) Reducing the search space: Since the most input for RCA, i.e., telemetry data, often lacks failure-related information, irrelevant data can be quickly pruned, thereby accelerating the RCA process. Additionally, if the results generated by the algorithm do not meet certain thresholds or patterns, the process can be terminated early, further enhancing efficiency. Almost all methods [83], [68], [14], [8], [74], [86], [46], [54] employ some form of pruning strategy to reduce the search space and improve efficiency. These strategies may involve similarity metrics (e.g., Pearson's correlation coefficient [87] in MicroHECL [83] and metric similarity pruning in CMDiagnostor [73]), event features (e.g., event pruning and Chi-square pruning in TraceContrast [73]), or topology (e.g., the DBSCAN algorithm [88] and influence topology filtering in GIED [74]). All these pruning strategies aim to eliminate irrelevant or low-relevance paths, nodes, or features to narrow down the problem and focus resources on deeper analysis.

Some methods organize and simplify the problem by exploiting the hierarchical nature of the data or structure. TraceDiag [75] is an adaptive and interpretable approach designed to enhance the efficiency of root cause analysis (RCA) in large-scale microservice systems. It leverages reinforcement learning (RL) to automatically learn a pruning policy that eliminates redundant components from service dependency graphs. This policy operates based on three main criteria: latency-based pruning, which removes components with low latency values; anomaly-based pruning, which filters out components with low anomaly indicators; and correlation-based pruning, which discards components with low correlation to end-to-end service latency. The RL-based pruning policy learns an optimal sequence of actions through a filtering tree, ensuring that the pruned graph retains only the components most relevant to RCA. HALO [84] optimizes inference time by leveraging the hierarchical structure of telemetry data to reduce the search space intelligently. It begins by identifying and organizing attributes into an Attribute Hierarchy Graph (AHG), which guides an efficient attribute-level search phase. This phase strategically narrows down the focus to a subset of relevant attributes. Subsequently, a value-level search is conducted along the hierarchical paths defined by the AHG, employing a top-down approach coupled with an adaptive early-stopping mechanism to further minimize the number of attribute value combinations evaluated. Additionally, HALO [84] refines the search results through reverse truncation to enhance the compactness of the fault-indicating combinations, thereby significantly streamlining the fault localization process without the need for pruning techniques. Onion [8] employs a "downward-closure based pruning" strategy during the clustering process, which stops the splitting process at an appropriate level, thus avoiding generating trivial log groups with low impact. This pruning strategy significantly improves efficiency by reducing unnecessary computations and focusing on critical information. Sieve [45] reduces the search space by loading the application under various load conditions to obtain metrics and a call graph, reducing the dimensionality of metrics using clustering to identify representative metrics, and identifying dependencies between components using the call graph and Granger Causality tests. The call graph helps to reduce the number of component pairs that need to be checked for relationships. In contrast, clustering reduces the number of metrics to consider by selecting representative metrics from each cluster. Squeeze [89] use a "bottom-up then top-down" pruning strategy to accelerate anomaly root cause analysis while balancing speed and accuracy. In the bottom-up phase, the method filters normal attribute combinations and clusters potential abnormal ones based on their deviation scores, significantly reducing the search space. The top-down phase then localizes the root cause within each cluster using a heuristic approach, guided by the generalized potential score (GPS), which enhances root cause identification by addressing cumulative forecast errors. This two-step approach efficiently narrows the search space and localizes root causes with improved precision and computational efficiency.

MicroHECL [83] dynamically constructs a service call graph based on aggregated 30-minute window data. Starting with the initial anomalous service of availability issue, it utilizes a pruning strategy that eliminates irrelevant service calls by assessing the similarity of change trends in quality metrics between successive edges (service calls) in the propagation chain. Specifically, the Pearson correlation coefficient [87] measures this similarity. If the coefficient is below a certain threshold, the edge is pruned, preventing the addition of potentially irrelevant anomalous nodes to the current anomaly propagation chain. Murphy [54], RCSF [85], GIED [74] and TraceRCA [86] also use some kinds of thresholds to filter the unrelated intermediate results. TraceContrast [68] first extracts the critical path, then transforms critical paths into event sets. Then, it identifies anomalous traces and affected paths. Using a parallel contrast sequential pattern mining algorithm, it uncovers candidate root causes. It mentions three types of pruning: Event pruning removes events unique to normal paths, Chi-square pruning eliminates patterns with a chi-square statistic value below a threshold when compared to their prefixes, and Minimum support pruning cuts off patterns with support lower than a specified threshold in the anomaly paths. CMDiagnostor [73] reduces the search space by using three key pruning strategies: AmSit-based Pruning (ASP), which removes irrelevant downstream paths; Metric Similarity-based Pruning (MSP), which eliminates paths where metric trends between successive calls are dissimilar; and Anomaly Detection-based Pruning (ADP), which prunes paths where metrics are normal. These strategies help focus the analysis on the most relevant parts of the graph, significantly narrowing down the potential root cause candidates. TraceDiag [75]uses three specific pruning methods to enhance root cause analysis by removing unnecessary nodes from trace data. The latency-based pruning method eliminates services with low latency metrics (such as average and maximum exclusive latencies) since they are less likely to be the root cause of an incident. The anomaly-based pruning method filters out components with low anomaly indicators, based on metrics like NormalizedCount and RankScore, as these services are unlikely to be involved in the incident. Lastly, the correlation-based pruning method removes nodes that show a low correlation between their latency and the overall end-to-end delay, as they are less likely to be relevant to the incident. MicroCU [46] implements pruning by enforcing a temporal order constraint on the path search within the temporal causal graph, ensuring that anomalies propagate in a strict time sequence. Applying this constraint, MicroCU [46] eliminates paths that do not conform to the required temporal order, thereby reducing the number of generated paths and improving their relevance. Additionally, MicroCU [46] validates paths based on causal peak timestamps, discarding those that fall outside the detected abnormal period. To further refine the search, it calculates path correlation strength while avoiding loops by restricting nodes to appear only once per path. The final ranking of potential root causes is determined by a weighted combination of path correlation strength and metric correlation strength, which together ensure that only the most relevant paths are considered, resulting in more accurate and efficient identification of root cause microservices.

2) Parallel Processing: Parallel processing is a common optimization methodology for speeding up the inference. TraceContrast [68] employs a parallelized contrast sequential pattern mining algorithm and Apache Spark's distributed computing capabilities to efficiently identify and analyze patterns in large-scale trace data, enabling rapid identification of root causes in complex microservice systems. Sage [55] trains its model in parallel. Nezha [10] traverses the event graph and mines the pattern in parallel. Murphy [54] and CIRCA [15] support further optimization with parallelism support. Microscope [38] uses a parallelized PC-algorithm to construct causality graphs based on a single metric and it leverages a conditioned graph traversing algorithm to locate the root cause.

B. COST EFFECTIVENESS AND ROBOSTNESS

Cost-effectiveness and robustness are crucial factors in assessing RCA algorithms since not all the systems can apply the monitoring infrastructure, and even those systems with the monitoring infrastructure, the collected data is will also lost [46]. To this end, effective RCA solutions should provide accurate results with minimal resource expenditure and the ability to function with less input data while maintaining or even enhancing performance.

MicroCU [46] only needs to use very sparse API logs to localize the root cause, based on the causal unimodalization, which effectively transforms causal curves into unimodal shapes, enhancing the interpretability and reliability of sparse data. Moreover, the dynamic causality discovery and the algorithm's robustness to different imputation methods contribute to its high accuracy, even under conditions of significant data sparsity. SpanGraph [90] demonstrates strong performance in few-shot learning scenarios, even with limited data. In tests using the SockShop [91] and Trainticket [92] datasets, SpanGraph achieved high F1 scores of 93% and 88.95% with just 1% of the training data. The model's performance improved across precision, recall, and F1-score as the data proportion increased. This highlights SpanGraph's efficiency, reliability, and ability to generalize well with minimal data, making it a competitive solution for fault localization in microservices systems where data is scarce.

Microrank [9] uses trace data to construct the trace coverage tree. For normal and abnormal traces, it has normal and abnormal coverage trees. Based on the trace coverage tree, it constructs the real-time call graph for further root cause analysis. This method does not take care much of the single traces, on the other hand, it uses the statistical feature of the trace, thus, Microrank [9] can accept some data missing. Similarly, Sage [55] also constructs the Causal Bayesian Network by the traces, it does not require tracking individual requests to detect temporal patterns, making it robust to tracing frequency. Murphy [54] is even more robust in their evaluation since its use of a Markov Random Field (MRF) framework for reasoning in cyclic graphs, online model training to ensure up-to-date accuracy, counterfactual analysis for identifying root causes even without explicit causal data, and an efficient diagnosis process that prunes the search space to reduce complexity while maintaining precision.

Diagfusion [72] tackles two primary challenges: (1) representing diverse data formats and (2) addressing imbalanced failure types. To achieve this, DiagFusion [72] uses fastText [93] for unified event representation across modalities, leveraging data augmentation to handle limited labeled data and imbalance in failure types. Additionally, it constructs a dependency graph from traces and deployment data to understand failure propagation paths and applies a graph neural network (GNN) to localize the root cause instance and classify the failure type.

C. SCALABILITY

Scalability refers to an algorithm's ability to efficiently manage increasing volumes of data and complexity without a significant loss in performance. In the context of RCA, scalable solutions are crucial for handling the expanding volume and variety of hierarchical telemetry data generated by microservices. To achieve high scalability, many approaches leverage the hierarchical nature of the data, dividing the original telemetry data into different levels of granularity and abstraction [3], [51], [14]. Additionally, because most telemetry data does not indicate failure, algorithms can quickly prune data that lacks clear evidence of an issue, thereby avoiding unnecessary computations and enhancing scalability, as we mentioned in previous sections.

Groot [3] and Chain-of-Event [51] achieve scalability in RCA by constructing a global service dependency graph to map interactions across distributed systems, this graph allows Groot and Chain-of-event to identify critical nodes that have a widespread impact on the system. The event graph aggregates low-level metrics, logs, and traces into higher-level features, reducing the data volume while preserving essential information, thus improving the scalability. RCD [14] employs a hierarchical learning approach by dividing the variables into smaller subsets and using the Ψ-PC algorithm to identify potential root causes within each subset. This method reduces the number of conditional independence (CI) tests needed, enhancing efficiency. After analyzing all subsets, RCD combines the results to identify candidate root causes. By focusing on localized learning around specific nodes (F-NODE), RCD further refines the potential root causes, improving both accuracy and runtime. This hierarchical strategy allows RCD to manage complex causal relationships more effectively. HALO [84] first identifies the relationships among attributes to construct the Attribute Hierarchy Graph (AHG), then generates attribute search paths by performing random walk on AHG. The second phase contains a top-down search along the hierarchically arranged attribute paths to identify the best attribute value combinations. By using this technique, HALO can scale up to 1.2 million records.

D. INCREMENTAL TRAINING

Change is common in microservice, there are service updates and service dependency changes every day. However, it is usually time-consuming and requires a large amount of data to train a robust model for new types of failure and dependency [58]. Thus, incremental training(online training) is important for maintaining the relevance and accuracy of RCA algorithms over time. It involves updating the model continuously as new data becomes available, allowing the system to adapt to changing conditions and new failure patterns. To achieve this, there are commonly two ways: incremental training that adopts the new data to the model [55], [56], [58], and online training that only uses fresh data [54].

a) Incremental training: CORAL [58] uses Long Short-Term Memory [94] (LSTM) networks and Variational Graph Autoencoders [95] (VGAE) to capture both temporal and structural patterns in evolving data. Specifically, LSTM can remember long-term data features, allowing it to update its internal state incrementally with new data without the need for full retraining. When new telemetry data arrives, LSTM [94] can continue to learn from this data based on its existing training results, thereby updating the model's prediction capabilities. This incremental learning approach avoids the need for full retraining with every data update, saving computational resources and enabling the model to adapt to new data distributions quickly. For VGAE [95], incremental learning is realized through the updatability of the graph structure. The service dependency graph in a microservice system is often dynamic, with services being added, removed, or dependencies changing. During incremental learning, VGAE [95] updates the graph structure and its latent representations (embeddings) locally, rather than rebuilding the entire graph model. For the same target, Sage [55] utilizes Graph Variational Autoencoders (GVAE) and Causal Bayesian Networks (CBN) to implement incremental learning in response to changes in a microservice architecture. The GVAE allows for dynamic reshaping and incremental updates by adjusting the network structure and parameters without retraining the entire model. Meanwhile, the CBN captures and updates causal relationships between microservices, guiding selective retraining based on changes in the RPC graph or performance metrics. Together, these components enable Sage [55] to efficiently adapt to microservice changes, reducing retraining time and maintaining model accuracy by updating only the relevant parts of the model and preserving causal integrity. Sleuth [56] improves upon Sage [55] by using Graph Neural Networks (GNNs) to enable few-shot and zero-shot learning, allowing it to quickly adapt to new microservice applications with minimal data. Unlike Sage [55], 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 makes Sleuth more scalable and efficient, especially in dynamic environments where rapid deployment and high accuracy are essential.

Different from the previous papers, Murphy [54] is inherently an online training method. Every time Murphy is called, it utilizes data from the week before the incident. This ensures that the models are continuously updated with the latest application topology and software versions, avoiding the use of outdated data. Including the most recent incident data is crucial, as it allows the model to capture new metrics patterns that may not have occurred previously, enhancing the accuracy of fault diagnosis.

E. HUMAN IN THE LOOP

Integrating human expertise into the Root Cause Analysis (RCA) process can significantly enhance the findings' accuracy and contextual relevance. Human-in-the-loop approaches, which combine algorithmic analysis with human insights, create a more robust and adaptable system. However, incorporating human input can sometimes require substantial effort. Thus, better utilization of human domain knowledge is an important topic. Humans can incorporate into multiple processes, including preparing high-quality inputs [15], [6], [3], [36], [41] and the evaluation process [6], [10], [4], [96], [97], [98].

a) Inputs: Groot [3] allows Site Reliability Engineers (SREs) to add new event types, construct event graphs, and maintain rules for filtering and building these graphs. This capability enables human experts to inject their knowledge directly into the RCA system. However, it also introduces the overhead of manual configuration. To mitigate this, Chain-of-Event [51] leverages historical incidents to automatically learn a Naive Event Graph while still allowing experts to refine parameters, thereby ensuring more effective utilization of human knowledge. Similar to Groot [3], Déjàvu [6], LogKG [99], CIRCA [15] and iSQUAD [36] require engineers to label the correctness of the result during the process or provide some predefined pattern to help the model to distinguish the root cause better. To stimulate the real environment, MEPFL [100] needs student volunteers to play the role of users and manually execute the scenarios that may involve the target microservice. MicroCause [40] needs professional operators to manually collect the online failure ticket and find out the actual root cause. HRLHF [41] draws inspiration from the outstanding results of Reinforcement Learning with Human Feedback (RLHF) in aligning large language models (LLMs) with human intent during training, effectively utilizing human feedback to reduce the uncertainty in dependency graph discovery. By integrating human feedback into reinforcement learning, HRLHF aims to construct dependency graphs with high accuracy while minimizing the need for human intervention. TraceDiag [75] leverages reinforcement learning (RL) to acquire an automated, interpretable, and adaptable pruning policy that effectively removes redundant components, enhancing the efficiency and accuracy of the RCA process. The policy is based on graph pruning rules derived from experienced engineers and comprehensive trace analysis, ensuring domain knowledge and industry best practices are incorporated. The pruned service graph is then utilized for RCA using causal methods.

b) Evaluation: There are no ground truth root causes in the actual industry service. Thus, in most cases, humans play an evaluation role to validate the performance of the RCA methods. Nezha [10] generates a suspicion list for SRE to check, which enables SRE to manually filter the miss-alarm. SynthoDiag [101], it is deployed to online service and needs SREs to manually check the performance of the RCA methods. RCACopilot [4], FSE Companion'24 [96], LM-PACE [97] and ICSE'23 [98] are four papers utilizing Large Language Model in root cause analysis, all of them needs human experts to evaluate the result. The human evaluation also needs to be simple, otherwise it also will cost lots of human effort. GMTA [102], Eadro [1], and Groot [3] all provided a user-friendly interface to interact with SREs.

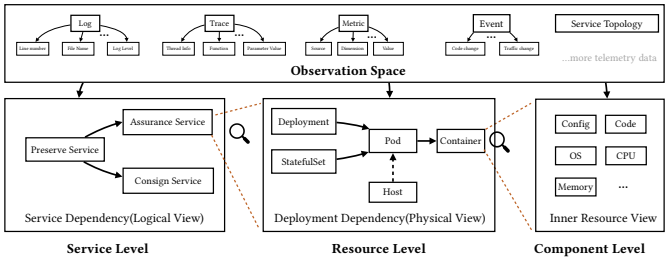

Fig. 4. Multi-level root causes

c) Others: Some works only target to reduce the effort of looking up the redundant and large volume telemetry data, thus speeding up the operator to make decisions of which component is the root cause. For example, the aim of GMTA [102] is to visualize the trace graph and do further analysis of visualizing service dependencies, making architectural decisions, analyzing the changes in service behaviors, detecting performance issues, and locating root causes. It does not directly output a list of root cause candidates. More human effort is needed to inspect the anomaly manually.

VII. CHALLENGES OF OUTPUT

Root Cause Analysis (RCA) is designed to equip Site Reliability Engineers (SREs) with the ability to quickly and accurately identify both the "what" and the "how" of an anomaly. To be truly effective, the output of RCA must be fine-grained and interpretable. A **fine-grained** output means isolating the ultimate root cause, enabling SREs to address it, thereby directly resolving all associated abnormal behaviors. **Interpretability** ensures that these results are easy to understand, reducing the time SREs spend on verification. Therefore, the challenges of RCA outputs can be distilled into two key aspects: producing highly detailed results and ensuring those results are easy to understand.

A. DIFFERENT GRANULARITY OF ROOT CAUSE

In the literature, various methods yield root causes at differing levels of granularity. To fully grasp this concept, it's important to understand that the granularity of identified root causes is closely tied to the granularity of the input data. Simply put more fine-grained observation data enables the inference of more detailed root causes. To illustrate this, we use an example from the TrainTicket case [103] to demonstrate the hierarchical nature of root cause granularities. From this view, we can comprehensively understand the relationships between the input data and the output result.

Root causes can be categorized into three primary levels: service level, resource level, and component level, with each successive level providing finer granularity. Each level is derived from the observation space, where finer-grained levels generally encompass the results from coarser-grained levels.

For example, the assurance service has logical dependencies on the preserve service. Within the assurance service, which is deployed on Kubernetes, corresponding resources are utilized. Drilling down further, within a specific resource (such as a container), more detailed resources and attributes can be observed. Depending on the scenario, the root cause could be at any of these levels [1]. For instance, if the SRE team needs to identify which team to contact, the service-level root cause might suffice. On the other hand, if the development team is tasked with fixing the issue, they may require more granular information, such as the root causes related to configuration changes or code changes.

1) At the service level: The root cause pertains to identifying which service is responsible for the observed anomaly (often indicated by a violation of Service Level Objectives, or SLOs) [39], [104], [63], [42], [41], [100], [83], [60], [86], [105], [55], [1], [2], [73], [54], [9], [13], [64], [65], [38], [75], [62], [14], [70], [106], [107], [108], [71], [69], [109], [76]. This helps SREs narrow down the search space from potentially thousands of services to just a few. For example, the system in Ant Group consists 3000+ microservices running on over 1 million virtual containers [110]. Resolving the root cause among these services is relatively difficult. At this level, the provided telemetry data(input) sometimes includes only service-specific metrics, such as CPU usage, service logs, and inter-service calling traces. Although helpful, the result of the RCA is too fine-grained, since the possible failure search space is still large within a single service. The underlined reason may be the model might not leverage fine-grained data, thereby limiting its ability to pinpoint the exact root cause. It can only determine which service is responsible for the current issue.

2) At the resource level: Multiple runtime resources are managed in the cloud-native environment, like deployment, stateful set, and pod. Some approaches [57], [72], [58], [56], [74], [78], [44], [67], [111], [112] pinpoint the specific resource, such as a pod or container, that is the root cause of the failure. This provides SREs with a more direct focus for further investigation. At this level, the ability to identify the root cause is largely due to the availability of more granular data, including metrics and logs specific to individual pods, containers, and hosts. By leveraging this fine-grained information, the methodologies can correlate anomalies directly with specific resources, rather than just higher-level services. This allows for a more precise diagnosis, facilitating quicker and more targeted remediation efforts. Consequently, the identification at the resource level not only reduces the search space further but also enhances the accuracy of the troubleshooting process, ultimately leading to more efficient incident resolution in cloud-native environments.

3) At the component level: Where methods [6], [5], [15], [113], [29], [49], [3], [51], [40], [28], [8], [90], [114], [102], [46], [115], [85], [116], [117], [45], [101], [89], [118], [11], [66], [50], [53], [52], [48], [47], [61], [119], [99], [120], [121], [122], [123] the inner component within some service or resource at fault, e.g., which lines of code, which operation, or which change event is the root cause. This level of granularity is achievable due to the availability of highly detailed observational data. For instance, logs may contain specific code snippets, traces might include function calls and attributes such as OS versions and the tools used for data collection, and metrics could detail the exact objects being monitored. We also categorize methods that identify specific telemetry data, such as a particular log [8] or a detailed metric, as the root cause at the component level. This is because such telemetry data inherently includes information about the corresponding component, such as file names, line numbers for logs, or the specific component associated with a metric or trace. By utilizing such fine-grained inputs, these approaches can output similarly detailed diagnoses, pinpointing the exact component responsible for the issue. This capability is critical for SREs and developers as it not only identifies the problematic component but also provides actionable insights that can be directly linked to the codebase or configuration settings. As a result, the troubleshooting process is accelerated, enabling more effective debugging and quicker resolution of incidents at a micro-level within the system.

4) At Multi-dimension level: There could be multiple output levels at the same time. In this approach, the algorithm considers the characteristics of the input data to identify root causes across various levels simultaneously, including service, resource, and component levels [10], [68], [59], [7], [84], [43], [77]. This level of analysis is highly adaptive, allowing it to provide the most context-appropriate diagnosis depending on the nature of the anomaly and the data available. For instance, an issue might be traced back to a specific service at the service level, while concurrently identifying a particular pod at the resource level and pinpointing a faulty code snippet at the component level. This multi-dimensional output is particularly valuable in complex, cloud-native environments where issues often span multiple layers of the infrastructure. By offering a comprehensive view that integrates insights from multiple levels, this approach empowers SREs and developers to address problems more holistically, ensuring that all potential root causes are considered and that remediation efforts are both targeted and effective. For example, Nezha [10] can localize three types of output, i.e., service, code region, and telemetry type; TrinityRCL [7] can output application, service, host, metric and code level root causes.

Through these multi-level outputs, it becomes clear that the granularity of the output is fundamentally determined by two key factors: **the utilization rate of the input data** (which varies depending on how different models process the data) and **the granularity of the input data itself**. Regardless of the model's effectiveness in utilizing data, the finest granularity of the output is limited by the detail present in the input. At best, we can pinpoint the smallest component within the observed data as the root cause of the anomaly, or we might determine that the root cause lies outside the current observation space [59]. Therefore, achieving finer granularity in root cause analysis hinges on two main challenges: enhancing the utilization of input data and increasing the granularity of the observed data. These aspects have been discussed in detail in Section V.

[1]In industry practice, the root causes usually are combinations of these levels, composing which service, which resource, and which component

B. INTERPRETABLE LOCALIZATION RESULTS

Interpretability is another crucial objective in root cause analysis, as it allows Site Reliability Engineers (SREs) to rapidly comprehend what transpired within the running service, why anomalies occurred, and how to resolve them. Similar to the granularity of root causes, interpretability can be classified into various levels, ranging from identifying a *single root cause* to constructing a *fault propagation graph*. The single root cause approach assumes there is only one underlying cause, while the fault propagation graph illustrates the dependencies and causal relationships among multiple root causes. Between these two extremes, intermediate levels exist, such as identifying *multiple root causes* or tracing a *fault propagation chain*. Additionally, we will also discuss the methods that output *explanations* of root causes/incidents.

1) Single Root Cause: This level is suited for models designed to identify only one root cause, without considering scenarios involving interactions between multiple root causes. The output is typically a ranked list of suspected root causes, with only one being the ground truth. For instance, although Eadro [1], BARO [29], and related works [40], [5], [7], [100], [63], [39], [70], [104], [42], [83], [60], [85], [116], [117], [45], [15], [58], [28], [113], [72], [44], [74], [84], [59], [2], [38], [115], [65], [8], [114], [14], [55], [56], [90], [101], [68], [75], [86], [36], [77], [43], [118], [106], [107], [67], [111], [11], [66], [50], [112], [53], [52], [48], [47], [61], [119], [108], [71], [69], [99], [109], [76], [120], [121], [122], [123] produce outputs at varying levels of granularity, as discussed in Section VII-A, they all focus on identifying a single root cause, implying that only one root cause exists within the current observation space. However, in reality, multiple root causes can coexist—such as a configuration change being the root cause and a surge in requests acting as the trigger described in Section III. In such cases, the absence of the trigger would mean the root cause might not lead to the observed anomaly. Thus, the current models may overlook complex scenarios where multiple root causes work together to produce the observed issue.

2) Multiple Root Causes: At this level, the model recognizes that real-world scenarios often involve multiple interacting root causes [62], [13], [124], [89]. It generates a list of potential root causes reflecting this complexity, providing a more comprehensive understanding of the issue than single root cause approaches. For example, GAMMA [62] addresses the challenge of multiple bottlenecks in microservices architecture (MSA), which can occur independently, dependently, or in a cascading manner, a scenario that has been largely overlooked by existing research that mainly deals with single bottlenecks. However, while these methods account for multiple root causes, they typically do not address the interactions and dependencies between them. In other words, they identify the possible root causes but do not explain how these root causes interact to create the current situation. This limitation leaves a gap in understanding the causal relationships that lead to the observed problem, which is crucial for effective remediation and prevention strategies.

3) Fault Propagation Graph: The fault propagation graph could be considered as the most interpretable illustration of the root causes since it contains the single root causes as the nodes and the possible propagation path as the edges. In this graph, the node root cause can be multi-dimensional as mentioned in the previous section, including the service level, resource level, and component level. This graph-based approach offers the most detailed and comprehensive explanation of how the current anomalies have occurred by capturing the intricate relationships and dependencies between multiple root causes. However, achieving this level of detail is challenging due to several factors.

(1) Obtaining accurate ground truth is difficult, as SREs and other operational staff may have varying or incomplete understandings of the incidents, leading to inconsistent or incorrect interpretations.

(2) There is a lack of standardized evaluation metrics for fault propagation graphs [49], making it hard to compare methods or validate their effectiveness at this level. These challenges have limited the development and adoption of strategies that fully meet the definition of a fault propagation graph.

As far as we know, ServerRCA [125] is the only method that formally claims that they output a fault propagation chain, indicating how the fault propagates from the ultimate root cause to the intermediate root cause. However, due to the aforementioned limitations, it still uses the $HR@K$ metric to evaluate the fault propagation chain, where the root node of the true fault propagation chain is the root cause. Nevertheless, many existing methods do maintain an internal graph structure to represent the spread of anomalies [51], [49], [54], [10], [46], [3], [57], [73], [9], [6], [63], [39], [41], [104], [83], [126], [115], [85], [15], [58], [28], [113], [44], [102], [64], [38], [90], [68], even if their final output is simplified to a root cause list. To acknowledge the significance of these methods, we discuss those that incorporate an internal fault propagation graph. Despite the mentioned challenges, these methods often revert to producing a ranked list of root causes, as the complexity of fully utilizing the graph structure remains an unresolved challenge in the field.

Three notable works focusing on fault propagation paths are Groot [3], Chain-of-Event [51], and GrayScope [49]. Specifically, Groot [3] employs a user interface to visually display the propagation path of anomalies, making it easier for SREs to verify root causes. Chain-of-Event [51] addresses three key questions: how different events influence each other, how the model arrives at its conclusions, and how suspicious a given event chain appears to SREs. These questions help provide a detailed illustration of the incident, aiding SREs in understanding the fault propagation. Since there is no standard definition for evaluating the precision of failure propagation paths, GrayScope [49] relies on human evaluation to assess their method's effectiveness. Other works like [54], [10], [46], [57], [73], [9], [6], [39], [104], [83] internally maintained a fault propagation graph, but they did not address or partially address the interpretability issue in terms of fault propagation graph.

There are also works providing the graph similarity metrics. For example, Brandón et al. [126] propose a graph similarity engine that is used to detect the anomalies in the graph. By

composing the graph elements(node, edge, context), the graph similarity is calculated to detect the anomaly. However, they did not emphasize this point, which is possible to be used for evaluating the interpretability issue.

4) Explanation of Root Cause/Incident Ticket: The field of root cause and incident explanation can be divided into two main components. The first involves explaining root cause candidates, including identifying the types of root causes and determining the specific category of potential faults. This is typically achieved by leveraging historical incident databases, which store past error data along with human-provided explanations. By matching new cases with similar past incidents, models can offer SREs relevant insights and references. The second component, distinct from traditional root cause analysis (RCA) methods, focuses on incident tickets as input. This emerging area of research, which gained traction around 2023, has been largely driven by advances in large language models (LLMs). LLMs enable the provision of detailed incident explanations within operational processes, filtering out irrelevant issues and compiling pertinent information. In our reviewed literature, five works [98], [96], [97], [4] have explored this area, highlighting the potential of LLMs in enhancing incident management and explanation.

Failure type is the most frequently used explanation, typically the action/symptom of the component or resources. Combining output multi-level root causes and the corresponding possible failure types can provide more interpretable results for SREs. For example, Déjàvu [6] considers bad requests, unavailable third-party services, failed disks, slow SQL queries, etc. as their failure types. MEPFL [100] also categorizes the failure categories into configuration-related, resource-related instance-related, and interaction-related. Their implementations usually contain using the previous known failure database to retrieve similar cases [6], [63], [99]. The most common failure types are discussed by numerous studies and surveys [32], [127], [128], [129], [130], [110], [131], [31], [132], thus we did not extend it.

Different from other works we introduced in previous sections, the input of these four papers are all incident descriptions, like incident tickets. ICSE'23 [98] did a rigorous study at Microsoft, on more than 40,000 incidents and compared several large language models in zero-shot, fine-tuned, and multi-task settings using semantic and lexical metrics. RCACopilot [4] is a system that leverages historical incident data and predefined rules to match incoming incidents with the most appropriate handlers based on the type of alert. For instance, if an alert pertains to a database connection issue, the system automatically assigns it to a database administrator. By integrating this capability, RCACopilot enhances the efficiency of incident resolution by reducing the manual effort traditionally required to investigate logs and trace data sources. Moreover, RCACopilot goes a step further by generating explanatory narratives for each incident. These narratives provide engineers with a comprehensive understanding of the incident, including context, the identified root cause, and recommended actions for resolution. This capability is crucial in helping on-call engineers quickly grasp the situation and respond effectively. FSE Companion'24 [96] uses a standard RAG and In-context-learning pipeline to retrieve the possible root cause of the provided incident. Specifically, it involves collecting and cleaning incident data, summarizing the incidents using GPT-3.5-turbo, and creating embedding vectors with a sentence transformer model. These embeddings are used to build a retrieval index with the Faiss library for efficient similarity searches. When a new incident occurs, relevant examples are retrieved from the index and included in a prompt provided to a large language model (e.g., GPT-4), which then generates the root cause analysis based on the new incident description and the retrieved in-context examples. To generate calibrated confidence scores, LMPACE [97] involves prompting the model in two distinct stages. In the first stage, the model quantifies the strength of evidence derived from historical incidents, allowing for an equitable assessment of the current incident. Subsequently, the second stage prompts the model to evaluate the candidate root cause produced by the root-cause predictor, utilizing the same set of references. A transformation is then applied to a validation set to estimate the confidence level based on the two scores generated from the prompting stages. Additionally, to address the challenges of limited historical data and evolving systems in root cause localization, OCRCL [133] not only integrates incident tickets, but also service text semantics, and static service dependency structures, employing online incremental learning to incorporate historical incident information and enhance model accuracy. By leveraging contrastive learning, OCRCL [133] augments incident tickets to improve the model's ability to link incidents with root causes. While this area shows great promise, it is still in its early stages, and further research is needed to refine these methods, improve their accuracy, and expand their applicability to a broader range of incidents.

VIII. RESEARCH TREND AND DISTRIBUTIONS

In this section, we analyze the research trend and distributions of root cause analysis.

A. Publication Trend

We first analyze the publications on the RCA topic each year at each conference or journal. As shown in Fig. 5, this chart presents the yearly distribution of publications related to the RCA topic, categorized by their classification under the China Computer Federation (CCF) recommendation list, which ranks conferences and journals into three levels: A, B, and C. The data shows a clear upward trend in publications over the years, particularly in the high-tier CCF A category. From 2013 to 2024, the most notable growth occurred in the CCF A category, with a sharp rise from 2021 onwards, peaking at 15 publications in 2023 and maintaining a significant number of 13 publications in 2024. This trend suggests that RCA research is increasingly being recognized and published in prestigious venues.

Fig. 6 illustrates the distribution of the collected papers across various research venues. The majority of RCA papers are published in software engineering venues, including ESEC/FSE, ISSRE, ICWS, ASE, ICSE, TSC, TOSEM, JSS, and ISSTA. Additionally, there are significant contributions in

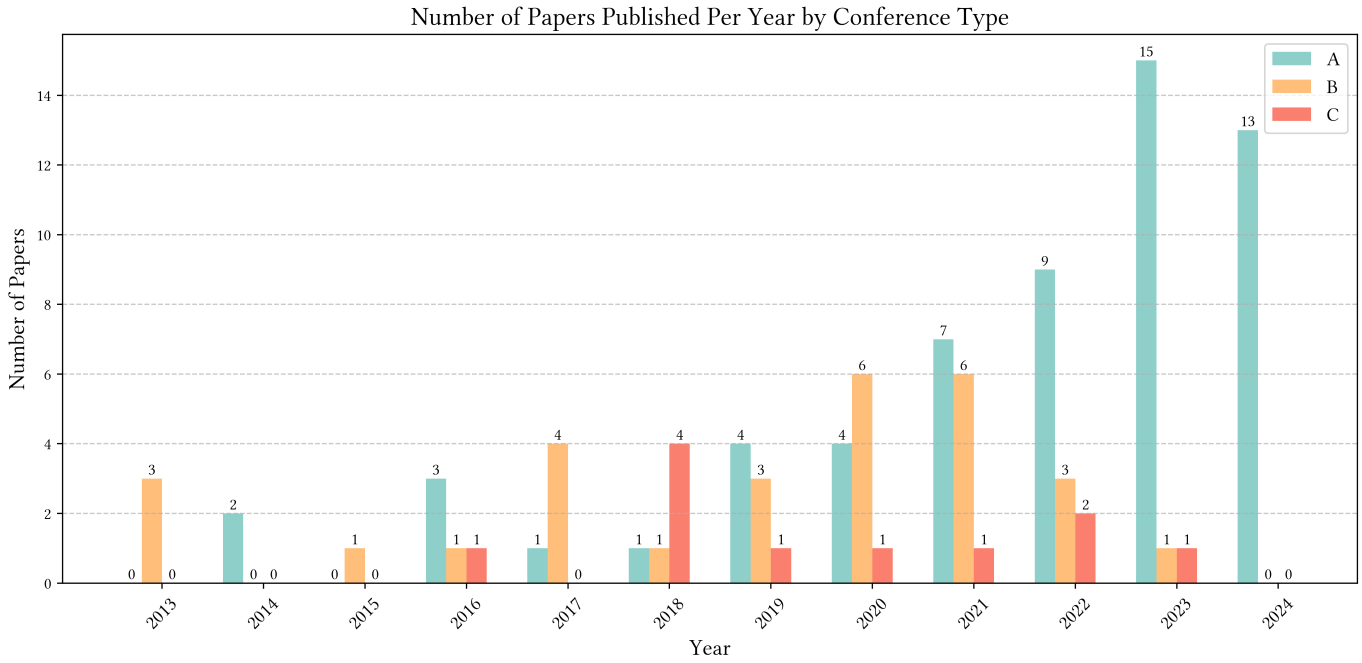

Fig. 5. Publication number of each class of the paper

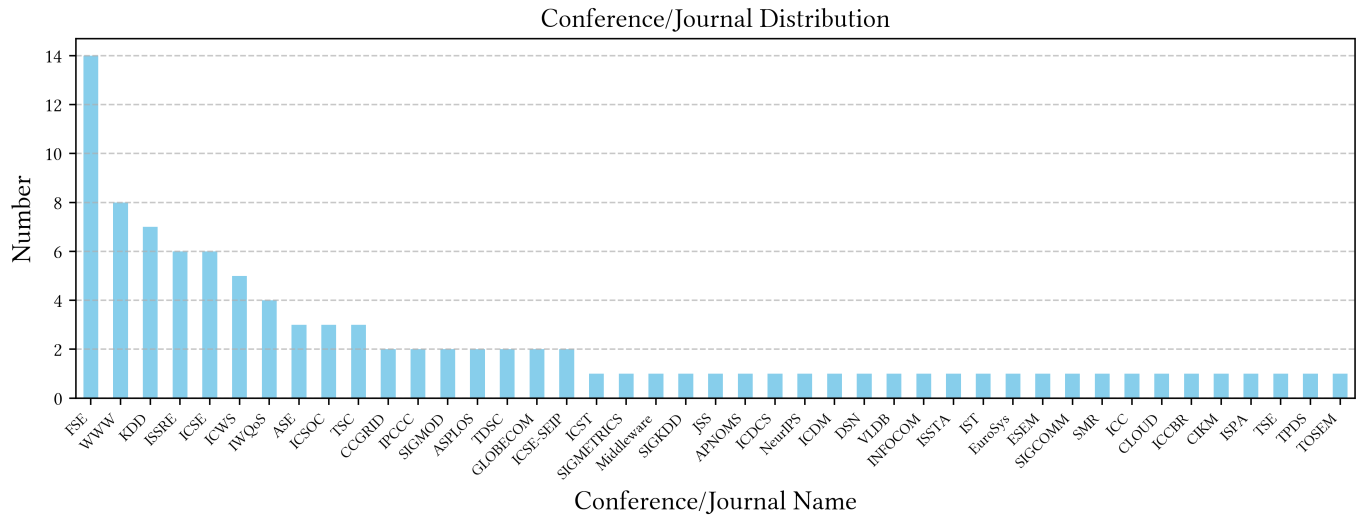

Fig. 6. Distribution of conference/journal

data mining venues, such as SIGKDD, as well as in artificial intelligence conferences like NeurIPS, systems venues such as EuroSys, and network and security venues like TDSC. This indicates that RCA is a relatively broad and interdisciplinary topic that spans multiple research fields, demonstrating its applicability and relevance across various domains.

Fig. 7 shows a steady increase in both total papers and company collaborations related to RCA since 2016, with a significant rise after 2020. While the total number of papers has grown more rapidly, peaking at 15 in 2023, the number of company collaborations has also steadily increased, maintaining a consistent ratio of around 50% of the total publications. This highlights the close connection between academic research and the industrial sector within this area.

B. Settings of RCA

Ground Truth Root Causes. The diversity of ground truth root causes directly impacts the effectiveness of RCA (Root Cause Analysis) models. In practice, the failure can occur anywhere the root cause runtime model in Fig. 9, including static files, user requests, runtime environment, and dependencies. By following the runtime model, in Fig. 8, we categorize the ground truth root causes discussed in 53 papers into six groups(the rest of the papers do not mention this information). Each group represents different types of failures that the experiments in these papers focus on. Specifically, the

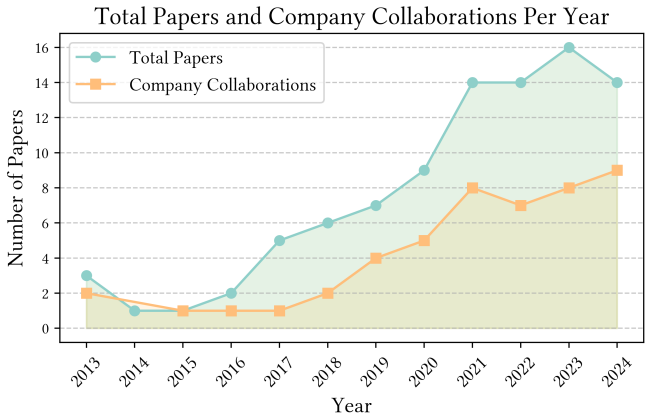

Fig. 7. Distribution of conference/journal from 2013 to 2024

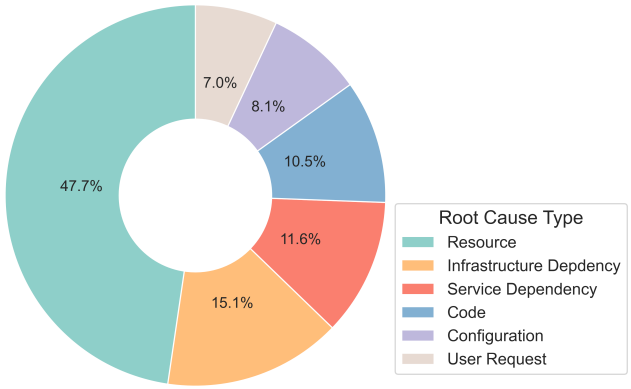

Fig. 8. The root cause distributions across 53 papers, each of which discusses the injected ground truth root cause used in their experiments.

categories are as follows: (1) *Resource*: This category refers to the resources required by a program or service, including CPU, memory, disk space, disk I/O, and network capacity. The root cause in this group relates to issues with these resources, such as outages, limitations, or corruption. (2) *Code*: This group pertains to failures stemming from the static code itself, including logic errors, low performance, poor design, or faulty changes. (3) *Configuration*: This category includes root causes associated with misconfigurations or errors in system configurations. (4) *User Request*: In this group, the root cause is linked to the nature of the user request, such as malicious requests, high volumes of requests, or poorly structured requests. (5) *Infrastructure Dependency*: This refers to failures in the underlying infrastructure, such as database errors, message queue (MQ) issues, or other foundational infrastructure errors. It is important to note that this category excludes resource-related failures (e.g., CPU resources are not considered part of the infrastructure). Instead, "infrastructure dependency" refers to failures in basic services that support the current system. (6) *Service Dependency*: This category addresses failures in service dependencies, including internal business logic failures or issues with third-party service dependencies that result in disruptions.

From Fig. 8, the high proportion of papers focusing on resource-related failures (47.7%) suggests that much of the RCA research prioritizes resource management issues such as CPU, memory, and network capacity. However, this heavy focus raises the question of whether RCA models are adequately addressing other critical failure types. In real-world systems, failures might arise from more complex or less obvious causes, such as human error, misconfigurations, or external service dependencies, which may not be fully captured in current research. While infrastructure and service dependencies make up a significant portion (15.1% and 11.6%, respectively), they may still be underexplored in comparison to real-world failure scenarios where intricate interactions between services and external systems often cause disruptions. Given the growing complexity of cloud-native and microservice-based architectures, the extent to which dependency failures are explored in RCA research could be insufficient to fully capture the challenges of modern systems. Additionally, with only 7.0% of studies focusing on user request-related failures, there appears to be a lack of emphasis on how user behaviors and malicious or abnormal requests can lead to system failures. In real-world environments, user interaction—whether intentional or unintentional—can be a major source of system disruptions, suggesting that this area warrants further investigation. The relatively low percentages for code-related (10.5%) and configuration-related (8.1%) failures indicate that these areas may not be fully explored. In practice, misconfigurations and software bugs often lead to severe outages and performance degradation. This discrepancy suggests that current RCA research might not be placing enough emphasis on these common real-world root causes, potentially overlooking critical aspects of system reliability.

Input and Output Settings. As discussed in Section III, both the telemetry data (input) and the root cause (output) are inherently complex, leading to numerous combinations of input-output relationships. Figure 10 illustrates this complexity: the X-axis represents the various input options, while the Y-axis shows the corresponding output levels. The most common output levels are service level, resource level, and component level.

Among these, the service level is the most frequent output, particularly in scenarios involving metrics (16 occurrences) and traces (10 occurrences). The resource level follows, with metrics (6 occurrences) and metric-topology combinations (2 occurrences) being the most prominent. Meanwhile, the component level exhibits considerable diversity, spanning a range of input combinations, including logs (4 occurrences) and metrics (10 occurrences). This figure underscores that, due to the complexity of input-output relationships, various configurations arise frequently in root cause analysis (RCA). The diverse combinations of inputs—such as logs, metrics, topology, and traces—across different levels (component, resource, and service) highlight the multifaceted nature of RCA scenarios. Addressing these challenges often requires flexibility and adaptability, as identifying root causes depends on multiple interacting factors, including the system's specific conditions and the nature of the telemetry data.

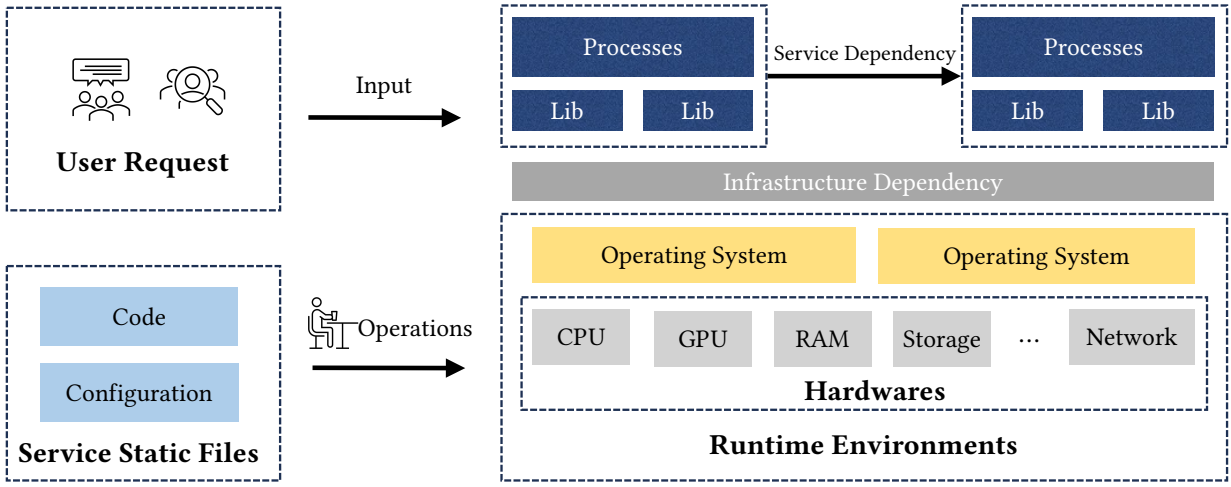

Fig. 9. Hierarchical Root Cause Model, root cause can be found all these components

TABLE II
ROOT CAUSE ANALYSIS TYPES AND DETAILED ENTITIES

Benchmark[2]	Svc #	LoC/Programming Languages	Protocol	Last Update
TrainTicket [92]	45	37746/Java, 23/Go, 292/Python, 5335/JavaScript, 9733/HTML	HTTP	2022-11-01
Online Boutique [134]	11	5881/Go, 1043/Python, 740/HTML, 634/C#, 347/JavaScript, 255/Java	gRPC	2024-10-03
Sock Shop [91]	9	4010/JavaScript, 3577/Java, 3283/Go, 1640/Python	HTTP	2023-12-05
HotelReservation [135]	10	7298/Go	gRPC	2024-06-28
SocialNetwork [135]	12	5753/C++	Thrift	2024-06-28

C. 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 II provides detailed information about five widely used public benchmarks in RCA research. The table includes the name of each benchmark, the number of services (Svc #), the total lines of code (LoC) along with the specific programming languages used, the communication protocol employed by each benchmark, and the date of the last update.

TrainTicket [92] is the largest benchmark in terms of both the number of services (45 services) and the total lines of code, which is distributed across several programming languages including Java, Go, Python, JavaScript, and HTML. Online Boutique [134], although smaller with 11 services, utilizes a diverse range of programming languages such as Go, Python, HTML, C#, JavaScript, and Java. Sock Shop [91], with 9 services, predominantly uses JavaScript, Java, Go, and Python for its implementation. Meanwhile, both HotelReservation and SocialNetwork are part of the DeathStarBench [135] suite, focusing on the use of Go and C++ respectively, with fewer services compared to TrainTicket [92]. The communication protocols used across these benchmarks vary, with HTTP, gRPC, and Thrift [136] being the primary protocols, reflecting different architectural and communication styles in microservice-based systems.

However, some of the benchmarks show signs of declining maintenance. For instance, TrainTicket [92] has not received any commits in over two years. Sock Shop [91] has been officially archived, signaling that it is no longer actively maintained or updated. DeathStarBench [135] claims to offer six sub-benchmarks, but in reality, only three—Social Network, Media Service, and Hotel Reservation—have been publicly released. Despite the original promise of more comprehensive coverage, the development activity for DeathStarBench has primarily focused on bug fixes in recent commits, with no significant feature updates or the release of the remaining sub-benchmarks. The only benchmark currently under active development is Online Boutique, which has maintenance through regular bug fixes and the introduction of new features.

D. Datasets

This section presents an overview of the publicly available datasets relevant to root cause analysis (RCA), as identified

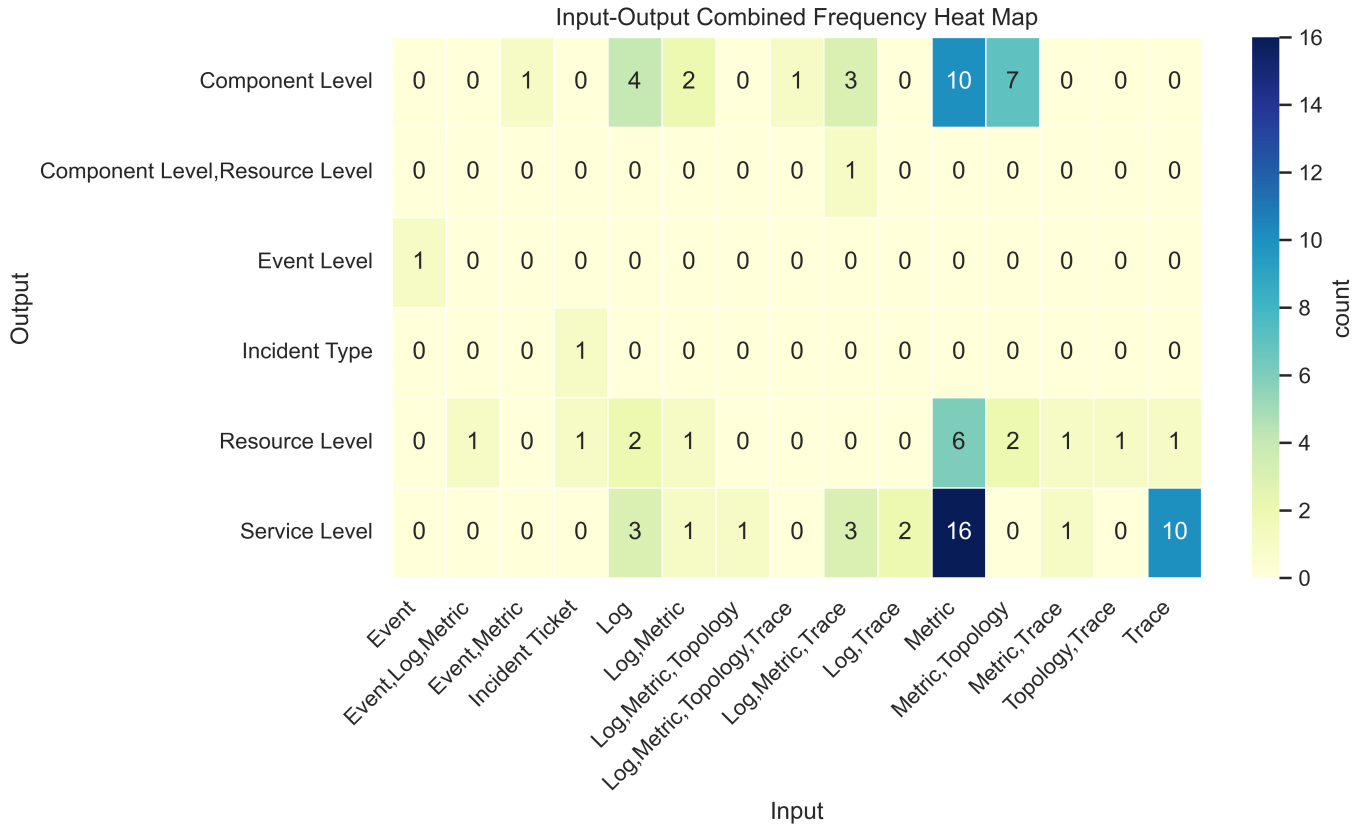

Fig. 10. Distribution of Input and Output Type

from the literature. Table III 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 [134](OB), SockShop [91](SS), TrainTicket [92](TT), DeathStarBench [135](DSB), SocialNetwork [135](SN). Notably, the datasets exhibit heterogeneity in terms of both data type and size. Some datasets exclusively contain metrics (M), while others combine multiple types, such as metrics, traces (T), and logs (L), reflecting the diverse nature of telemetry data. The data formats also vary, with most datasets stored as CSV files, though some use alternative formats such as JSON or XLSX. The dataset sizes range from relatively small files (e.g., 4.1MB for Dycause [44]) to much larger ones (e.g., 99GB for Murphy-DSB [54]), highlighting substantial differences in the amount of information each dataset provides for analysis.

E. 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 106 papers we reviewed, 21 have openly shared their code implementations, as shown in Table IV. Notable tools such as BARO [29], LatentScope [59], and Chain-of-Event [51] from recent years (2024) reflect a trend toward open-sourcing model implementations, which is instrumental in promoting transparency and reproducibility.

The chronological progression of these tools, from early contributions like Sieve [45] (2017) to more recent advancements like GIED [74] and Eadro [1], highlights the evolution of methodologies in root cause analysis. This diversity enriches the research community, enabling comparisons and adaptations of distinct models tailored to varied application scenarios. Such openness not only fosters collaboration but also provides a foundation for future innovations and standardized benchmarking, essential for advancing the field systematically.

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

IX. DISCUSSION

A. 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 [35] aligns with the general scope of this paper—encompassing anomaly detection, root cause localization, and anomaly mitigation—the specific context of RootCLAM [35] is quite different from ours. RootCLAM [35] utilizes a loan approval scenario based on

TABLE III
PUBLIC DATASET FOR ROOT CAUSE ANALYSIS(M FOR METRICS, E FOR EVENTS, AND T FOR TRACES)

Dataset	Type	Format	Amount	Used By
Dejavu-A1[137]	M	CSV	75.1MB	[6]
Dejavu-A2[137]	M	CSV	82.2MB	[6]
Dejavu-B[137]	M	CSV	1.7GB	[6]
Dejavu-C[137]	M	CSV	48.8MB	[6]
Dejavu-D[137]	M	CSV	3.7GB	[6]
RCD-SS[138]	M	CSV	16MB	[14]
ChangeRCA-OB[139]	M	CSV	60MB	[5]
BARO-TT[140]	M	CSV	1.1GB	[29]
BARO-SS[140]	M	CSV	337MB	[29]
BARO-OB[140]	M	CSV	339MB	[29]
Squeeze[141]	M	CSV	18G	[89]
Dycause[142]	M	XLSX	4.1M	[44]
GrayScope[143]	M	CSV	8.4M	[49]
LatentScope[144]	M	JSON	2.1G	[59]
MicroCU[145]	M	npy	118M	[46]
Murphy-DSB[146]	M, T	JSON	99GB	[54]
AIOps Comp-2020[147]	M, T	CSV	16G	[9], [111], [114], [105], [67]
Nezha-OB[148]	M, T, L	CSV	2.5GB	[10]
Nezha-TT[148]	M, T, L	CSV	351MB	[10]
GAIA[149]	M, T, L	CSV	41G	[72]
Eadro-TT[150]	M, T, L	CSV/JSON	841M	[1]
Eadro-SN[150]	M, T, L	CSV/JSON	1.3G	[1]
AIOps Comp-2021[147]	M, T, L	CSV	25G	[114], [57]
GAMMA[151]	M, L	RAW format/CSV	39GB	[62]
MEPFL-TT[152]	T	CSV	2.3G	[100]
MEPFL-SS[152]	T	CSV	59M	[100]

TABLE IV
PUBLICLY AVAILABLE TOOLS/CODES FOR ROOT CAUSE ANALYSIS

Tool	URL	Year	Tool	URL	Year
BARO[29]	[153]	2024	GIED[74]	[154]	2022
LatentScope[59]	[155]	2024	DeepTraLog[113]	[156]	2022
Chain-of-Event[51]	[157]	2024	CIRCA[15]	[158]	2022
ChangeRCA[5]	[159]	2024	RCD[14]	[160]	2022
Eadro[1]	[161]	2023	DejaVu[6]	[162]	2022
MicroCU[46]	[163]	2023	TraceRCA[86]	[164]	2021
Diagfusion[72]	[165]	2023	MicroRank[9]	[166]	2021
CMDiagnostor[73]	[167]	2023	squeeze[89]	[168]	2019
Nezha[10]	[169]	2023	Log3C[121]	[170]	2018
MicroCBR[77]	[171]	2022	Sieve[45]	[172]	2017
SwissLog[109]	[173]	2022	-	-	-

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.

Compatibility of event graph. We use an event graph to describe the failure propagation, which is compatible with all the previous output formats we have surveyed. Specifically, the single event node can represent the service, resource, and component-level root cause, since the event contains a timestamp, action, entity, and context, most of the current output is a part of the event. The failure propagation chain shows the propagation of the anomaly, and the event graph is the dynamic description of the runtime system.

B. FUTURE WORK DIRECTIONS

In the introduction, we identified three critical gaps within the current research landscape. These gaps were categorized into input, method, and output-oriented challenges. Subsequent sections of this work have demonstrated how existing methods have addressed these challenges to varying degrees. However,

in this section, we aim to highlight the challenges that remain unresolved or insufficiently addressed, thereby outlining potential directions for future research.

1) Comprehensive Telemetry Data: As we introduced in Section V, advancing Root Cause Analysis (RCA) necessitates the development of more comprehensive and realistic telemetry data, which requires both enhanced benchmarks and more sophisticated fault injection techniques.

- *Realistic benchmarks.* As discussed in Section VIII, current benchmarks are often limited in scale and complexity, typically involving small microservices with simple interactions. These constraints hinder the identification of specific real-world issues, such as network partitioning problems in benchmarks lacking high-availability features. Future benchmarks should incorporate a broader range of telemetry data types, simulate more complex service interactions, and include monitoring infrastructures capable of collecting extensive and varied telemetry data.
- *Comprehensive fault injection.* Improving RCA also requires the use of more advanced and diverse fault injection techniques. The quality of telemetry data is not only dependent on its type but also on the nature of the incidents it captures. As highlighted in Section V, previous studies have predominantly focused on injecting basic CPU, memory, and network issues, which are relatively easy to diagnose. Future research should aim to simulate faults that closely resemble real-world scenarios, including operational errors, configuration mistakes, and code anomalies, to better reflect the wide array of issues encountered in practice.

By pursuing these directions, the field can generate more comprehensive telemetry data, which in turn will enable the development of more accurate, robust, and efficient RCA models. For example, future research could explore the combination of different telemetry data types, varying levels of granularity, and diverse data volumes to design models that better address the complexities of real-world systems.

2) Explore New Types of Telemetry Data: As we introduced in Section V-A, more fine-grained data can help localize more fine-grained root causes. Currently, profiling data can be continuously collected [174], [175], which means localizing which code line causes the incident is possible. Some events, such as code updates, deployments, and attacks, could also describe the system behavior. Adding these new data, and solving the inherent challenges within the new data is worth investigating. Some studies [130], [110] show that the "change" in the system is likely to be the root cause of the incident. However, in our survey, few papers focus on this [5], [3].

3) Eliminate the Side-Effect of Anomaly Detection: In current practices, anomaly detection often serves as the trigger for Root Cause Analysis (RCA). However, this approach means that the accuracy of RCA is heavily dependent on the precision of anomaly detection. Currently, indeed there are some works [1], [29] that consider anomaly detection and root cause analysis as an end-to-end task, however, they still use follow the pattern that anomaly detection is the trigger, if there is an anomaly detected, then the RCA will start, this means there still could be false negatives. The result of anomaly detection can be utilized by the root cause analysis phase, reducing the search space by filtering some simple patterns in advance.

4) Interpretable RCA Models: Future RCA (Root Cause Analysis) models should strive for greater interpretability. With well-defined input and output interfaces, we envision future models not only identifying the root cause of an issue but also illustrating the incident propagation path through a comprehensive causal graph. This graph would clarify which event is the root cause, which acts as a trigger, and how these events interact to lead to the failure. Such detailed outputs would provide deeper insights into the development of incidents, enabling more effective mitigation strategies.

While current research has achieved some level of interpretability by offering explanations [10] and providing propagation graphs [3], [51], [49], most methods still rely on internal graphs for inference and only output single root causes [54], [10], [46], [57], [73], [9], [6], [39], [104], [83]. This limitation primarily arises from the lack of standardized evaluation metrics for failure propagation graphs (paths) [49]. Therefore, achieving better interpretability requires two key developments: the establishment of standard evaluation metrics for propagation graphs and the creation of corresponding datasets that include ground truth data. With these foundational elements in place, more interpretable RCA models can be developed.

5) Cost-effective and Robust RCA models: Developing cost-effective RCA models is essential for practical implementation in industry settings. Collecting and storing extensive telemetry data, as outlined in Section VI, is expensive and often impractical. Most companies cannot keep all the necessary data, and they only hold sampled data. Additionally, the collected telemetry data itself can also be incomplete and inaccurate due to the setting of monitoring infrastructure [176]. Therefore, future models should be designed to achieve high performance with minimal and dirty data, making RCA more accessible and practical for a broader range of organizations.

6) Intelligent Sample Algorithm: As previously mentioned, companies often retain sampled telemetry data for only short periods (e.g., a few days or months) due to the high cost and redundancy of storing all data. However, because failure-related data is rare, relying solely on random sampling may lead to the loss of critical information. Additionally, this approach can disrupt the effectiveness of existing root cause analysis (RCA) models, as changes in the data distribution caused by the sampling process may render these models less accurate. Consequently, RCA models need to be robust enough to accommodate such variations. Recent research has focused on more intelligent trace sampling methods [12], [177], [178]. However, other types of telemetry data, such as profiling, logs, and metrics, remain less explored.

7) Efficient RCA models: Efficiency is critical for RCA models, especially in high-demand environments like Alibaba, where SREs need to localize problems within 5 minutes and fix them within 10 minutes [179]. Future RCA models must be optimized for speed and quick response times to meet these stringent operational demands, ensuring rapid localization and resolution of incidents.

X. RELATED WORKS

Root cause analysis (RCA) is a critical component of incident management, enabling Site Reliability Engineers (SREs) to quickly identify and resolve incidents. Previous surveys on RCA have been successfully published, focusing primarily on the methodologies employed within RCA. However, these surveys often overlook the ultimate objectives of RCA, which can lead to a lack of clear direction.

Soldani et al.[16] and Zhang et al.[17] provide surveys that are closely related to our topic. Soldani et al.[16] focus on both anomaly detection techniques and RCA techniques. Zhang et al.[17] concentrate on root cause localization techniques and failure classification techniques. While there is some overlap between our survey and previous ones, our objectives differ. Specifically, Zhang et al. [17] aim to distinguish failure diagnosis objectives, explore multimodal data, and analyze practical applications. In contrast, we aim to provide a detailed explanation of what RCA entails, and its role within incident management, and to categorize papers by summarizing common challenges. For example, Zhang et al.[17] note that various methods produce different levels of root cause identification, ranging from service-level to instance-level. However, they do not address why these multiple levels of root causes exist. We address this by summarizing the CEO principle (SectionIII-C), which posits that the granularity of the root cause is restricted by the input granularity. Increasing input granularity raises storage costs. Therefore, to achieve more fine-grained, interpretable, and precise root causes, there are two approaches: adding fine-grained inputs and better utilizing current data (e.g., mining relations/patterns) to infer beyond unobserved data. By categorizing previous works in such a goal-driven way, we enable readers to think about and understand the current state of RCA research from a high-level perspective, identify existing challenges, and explore potential solutions to these challenges.

Sole et al. [180]'s definition of RCA is similar to ours, with the model construction, inference, and model update processes, where the training process uses domain knowledge, system knowledge, and observations, the inference process uses observations to infer the root causes and give the explanations. Our definition of RCA is the observation, inference, and result. However, they did not discuss the different types of input data(different modality), and the unified output format(the failure propagation graph).

Our purpose is not to replace or criticize previous surveys but rather to interpret the current state of RCA research from the perspective of existing challenges. This approach is complementary to other surveys. By integrating our survey with others, readers can gain a comprehensive understanding of the development of existing technologies, both from a challenge-oriented perspective and in a broader context.

XI. CONCLUSION

In this survey, we present a comprehensive overview and analysis of recent research on Root Cause Analysis (RCA). We discuss the current limitations within RCA workflows, elucidating why existing research efforts often appear fragmented and ad-hoc rather than directed toward a unified research objective. Anchored by a common goal—minimizing service downtime—this review categorizes RCA techniques according to the specific challenges they address, providing a clear framework for understanding their definitions and current research status. We also review experimental datasets and available open-source tools, offering an in-depth analysis of emerging trends, research directions, opportunities, and challenges in the RCA domain. We hope this survey serves as a valuable resource for researchers across diverse fields, enhancing their understanding of the current state of RCA research and the open avenues for future exploration.

REFERENCES

[1] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "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, 2023, pp. 1750–1762.

[2] L. Zheng, Z. Chen, J. He, and H. Chen, "Mulan: Multi-modal causal structure learning and root cause analysis for microservice systems," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 4107–4116.

[3] H. Wang, Z. Wu, H. Jiang, Y. Huang, J. Wang, S. Kopru, and T. Xie, "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, 2021, pp. 419–429.

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

[5] G. Yu, P. Chen, Z. He, Q. Yan, Y. Luo, F. Li, and Z. Zheng, "Changerca: Finding root causes from software changes in large online systems," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 24–46, 2024.

[6] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui *et al.*, "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*, 2022, pp. 996–1008.

[7] S. Gu, G. Rong, T. Ren, H. Zhang, H. Shen, Y. Yu, X. Li, J. Ouyang, and C. Chen, "Trinityrcl: Multi-granular and code-level root cause localization using multiple types of telemetry data in microservice systems," *IEEE Transactions on Software Engineering*, vol. 49, no. 5, pp. 3071–3088, 2023.

[8] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin *et al.*, "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*, 2021, pp. 1253–1263.

[9] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, 2021, pp. 3087–3098.

[10] G. Yu, P. Chen, Y. Li, H. Chen, X. Li, and Z. Zheng, "Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 553–565.

[11] V. Jeyakumar, O. Madani, A. Parandeh, A. Kulshreshtha, W. Zeng, and N. Yadav, "Explainit!–a declarative root-cause analysis engine for time series data," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 333–348.

[12] H. Huang, X. Zhang, P. Chen, Z. He, Z. Chen, G. Yu, H. Chen, and C. Sun, "Trastrainer: Adaptive sampling for distributed traces with system runtime state," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 473–493, 2024.

[13] S. Yan, C. Shan, W. Yang, B. Xu, D. Li, L. Qiu, J. Tong, and Q. Zhang, "Cmmd: Cross-metric multi-dimensional root cause analysis," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4310–4320.

[14] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu, "Root cause analysis of failures in microservices through causal discovery," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 158–31 170, 2022.

[15] M. Li, Z. Li, K. Yin, X. Nie, W. Zhang, K. Sui, and D. Pei, "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*, 2022, pp. 3230–3240.

[16] J. Soldani and A. Brogi, "Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–39, 2022.

[17] S. Zhang, S. Xia, W. Fan, B. Shi, X. Xiong, Z. Zhong, M. Ma, Y. Sun, and D. Pei, "Failure diagnosis in microservice systems: A comprehensive survey and analysis," *arXiv preprint arXiv:2407.01710*, 2024.

[18] P. Di Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *2017 IEEE International conference on software architecture (ICSA)*. IEEE, 2017, pp. 21–30.

[19] "Spring boot," https://spring.io/projects/spring-boot, 2024, accessed: 2024-08-19.

[20] "Dubbo," https://dubbo.apache.org/en/index.html, 2024, accessed: 2024-08-19.

[21] "Docker," https://www.docker.com/, 2024, accessed: 2024-08-19.

[22] "Spring cloud," https://spring.io/projects/spring-cloud, 2024, accessed: 2024-08-19.

[23] "Mesos," https://mesos.apache.org/, 2024, accessed: 2024-08-19.

[24] "Kubernetes," https://kubernetes.io/, 2024, accessed: 2024-08-19.

[25] "Jenkins," https://www.jenkins.io/, 2024, accessed: 2024-08-19.

[26] "Gitlab," https://about.gitlab.com/, 2024, accessed: 2024-08-19.

[27] J. Lewis and M. Fowler, "Microservices a definition of this new architectural term," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[28] P. Chen, Y. Qi, P. Zheng, and D. Hou, "Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1887–1895.

[29] L. Pham, H. Ha, and H. Zhang, "Baro: Robust root cause analysis for microservices via multivariate bayesian online change point detection," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2214–2237, 2024.

[30] "Incident management guide," https://sre.google/resources/practices-and-processes/incident-management-guide/, 2024, accessed: 2024-08-19.

[31] J. Chen, S. Zhang, X. He, Q. Lin, H. Zhang, D. Hao, Y. Kang, F. Gao, Z. Xu, Y. Dang *et al.*, "How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 373–384.

[32] V. Ganatra, A. Parayil, S. Ghosh, Y. Kang, M. Ma, C. Bansal, S. Nath, and J. Mace, "Detection is better than cure: A cloud incidents perspective," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1891–1902.

[33] M. E. Cournoyer, S. Trujillo, C. M. Lawton, W. M. Land, and S. B. Schreiber, "Anatomy of an incident," *Journal of Chemical Health & Safety*, vol. 23, no. 6, pp. 40–48, 2016.

[34] Y. Zhao, L. Jiang, Y. Tao, S. Zhang, C. Wu, T. Jia, X. Huang, Y. Li, and Z. Wu, "Identifying root-cause changes for user-reported incidents in online service systems," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 287–297.

[35] X. Han, L. Zhang, Y. Wu, and S. Yuan, "On root cause localization and anomaly mitigation through causal inference," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 699–708.

[36] M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X. Jiang, H. Hu, C. Luo, Y. Li, N. Qiu *et al.*, "Diagnosing root causes of intermittent slow queries in cloud databases," *Proceedings of the VLDB Endowment*, vol. 13, no. 8, pp. 1176–1189, 2020.

[37] S. Jalali and C. Wohlin, "Systematic literature studies: database searches vs. backward snowballing," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, 2012, pp. 29–38.

[38] J. Lin, P. Chen, and Z. Zheng, "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, 2018, pp. 3–20.

[39] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.

[40] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[41] L. Wang, C. Zhang, R. Ding, Y. Xu, Q. Chen, W. Zou, Q. Chen, M. Zhang, X. Gao, H. Fan *et al.*, "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*, 2023, pp. 5116–5125.

[42] M. Ma, W. Lin, D. Pan, and P. Wang, "Servicerank: Root cause identification of anomaly in large-scale microservice architectures," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3087–3100, 2021.

[43] Y. Zhang, Z. Guan, H. Qian, L. Xu, H. Liu, Q. Wen, L. Sun, J. Jiang, L. Fan, and M. Ke, "Cloudrca: A root cause analysis framework for cloud computing platforms," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4373–4382.

[44] Y. Pan, M. Ma, X. Jiang, and P. Wang, "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*, 2021, pp. 646–657.

[45] J. Thalheim, A. Rodrigues, I. E. Akkus, P. Bhatotia, R. Chen, B. Viswanath, L. Jiao, and C. Fetzer, "Sieve: Actionable insights from monitored metrics in distributed systems," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 14–27.

[46] X. Jiang, Y. Pan, M. Ma, and P. Wang, "Look deep into the microservice system anomaly through very sparse logs," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2970–2978.

[47] Y. Chen, D. Xu, N. Chen, and X. Wu, "Frl-mfpg: Propagation-aware fault root cause location for microservice intelligent operation and maintenance," *Information and Software Technology*, vol. 153, p. 107083, 2023.

[48] L. Mariani, C. Monni, M. Pezzé, O. Riganelli, and R. Xin, "Localizing faults in cloud systems," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 262–273.

[49] S. Zhang, Y. Zhao, X. Xiong, Y. Sun, X. Nie, J. Zhang, F. Wang, X. Zheng, Y. Zhang, and D. Pei, "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*, 2024, pp. 126–137.

[50] Z. Hu, P. Chen, G. Yu, Z. He, and X. Li, "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, 2022, pp. 109–119.

[51] Z. Yao, C. Pei, W. Chen, H. Wang, L. Su, H. Jiang, Z. Xie, X. Nie, and D. Pei, "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*, 2024, pp. 50–61.

[52] X. Lu, Z. Xie, Z. Li, M. Li, X. Nie, N. Zhao, Q. Yu, S. Zhang, K. Sui, L. Zhu *et al.*, "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, 2022, pp. 655–664.

[53] X. Nie, Y. Zhao, K. Sui, D. Pei, Y. Chen, and X. Qu, "Mining causality graph for automatic web-based service diagnosis," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016, pp. 1–8.

[54] V. Harsh, W. Zhou, S. Ashok, R. N. Mysore, B. Godfrey, and S. Banerjee, "Murphy: Performance diagnosis of distributed cloud applications," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 438–451.

[55] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ml-driven performance debugging in microservices," in *Proceedings of the 26th ACM International Conference on Architec-*

tural Support for Programming Languages and Operating Systems, 2021, pp. 135–151.

[56] Y. Gan, G. Liu, X. Zhang, Q. Zhou, J. Wu, and J. Jiang, "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*, 2023, pp. 324–337.

[57] D. Wang, Z. Chen, J. Ni, L. Tong, Z. Wang, Y. Fu, and H. Chen, "Interdependent causal networks for root cause localization," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 5051–5060.

[58] D. Wang, Z. Chen, Y. Fu, Y. Liu, and H. Chen, "Incremental causal graph learning for online root cause analysis," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 2269–2278.

[59] Z. Xie, S. Zhang, Y. Geng, Y. Zhang, M. Ma, X. Nie, Z. Yao, L. Xu, Y. Sun, W. Li *et al.*, "Microservice root cause analysis with limited observability through intervention recognition in the latent space," 2024.

[60] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 48–58.

[61] L. Wu, J. Bogatinovski, S. Nedelkoski, J. Tordsson, and O. Kao, "Performance diagnosis in cloud microservices using deep learning," in *International Conference on Service-Oriented Computing*. Springer, 2020, pp. 85–96.

[62] G. Somashekar, A. Dutt, M. Adak, T. Lorido Botran, and A. Gandhi, "Gamma: Graph neural network-based multi-bottleneck localization for microservices applications," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 3085–3095.

[63] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, 2020, pp. 246–258.

[64] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[65] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[66] W. Lin, M. Ma, D. Pan, and P. Wang, "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, 2018, pp. 1–8.

[67] J. Yang, Y. Guo, Y. Chen, Y. Zhao, Z. Lu, and Y. Liang, "Robust anomaly diagnosis in heterogeneous microservices systems under variable invocations," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 2722–2727.

[68] C. Zhang, Z. Dong, X. Peng, B. Zhang, and M. Chen, "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*, 2024, pp. 1–12.

[69] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "Fchain: Toward black-box online fault localization for cloud systems," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 21–30.

[70] M. Ma, W. Lin, D. Pan, and P. Wang, "Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 60–67.

[71] P. Liu, Y. Chen, X. Nie, J. Zhu, S. Zhang, K. Sui, M. Zhang, and D. Pei, "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, 2019, pp. 35–46.

[72] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin *et al.*, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 3851–3864, 2023.

[73] Q. Yu, C. Pei, B. Hao, M. Li, Z. Li, S. Zhang, X. Lu, R. Wang, J. Li, Z. Wu *et al.*, "Cmdiagnostor: An ambiguity-aware root cause localization approach based on call metric data," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2937–2947.

[74] Z. He, P. Chen, Y. Luo, Q. Yan, H. Chen, G. Yu, and F. Li, "Graph based incident extraction and diagnosis in large-scale online systems," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–13.

[75] R. Ding, C. Zhang, L. Wang, Y. Xu, M. Ma, X. Wu, M. Zhang, Q. Chen, X. Gao, X. Gao *et al.*, "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*, 2023, pp. 1762–1773.

[76] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu, "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, 2017, pp. 25–32.

[77] F. Liu, Y. Wang, Z. Li, R. Ren, H. Guan, X. Yu, X. Chen, and G. Xie, "Microcbr: Case-based reasoning on spatio-temporal fault knowledge graph for microservices troubleshooting," in *International Conference on Case-Based Reasoning*. Springer, 2022, pp. 224–239.

[78] H. Shan, Y. Chen, H. Liu, Y. Zhang, X. Xiao, X. He, M. Li, and W. Ding, "ϵ-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms," in *The World Wide Web Conference*, 2019, pp. 3215–3222.

[79] P. Spirtes, C. Glymour, and R. Scheines, *Causation, prediction, and search*. MIT press, 2001.

[80] S. Z. Li, *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[81] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[82] C. Wu, N. Zhao, L. Wang, X. Yang, S. Li, M. Zhang, X. Jin, X. Wen, X. Nie, W. Zhang *et al.*, "Identifying root-cause metrics for incident diagnosis in online service systems," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 91–102.

[83] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, "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, 2021, pp. 338–347.

[84] X. Zhang, C. Du, Y. Li, Y. Xu, H. Zhang, S. Qin, Z. Li, Q. Lin, Y. Dang, A. Zhou *et al.*, "Halo: Hierarchy-aware fault localization for cloud systems," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3948–3958.

[85] K. Wang, C. Fung, C. Ding, P. Pei, S. Huang, Z. Luan, and D. Qian, "A methodology for root-cause analysis in component based systems," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, 2015, pp. 243–248.

[86] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang *et al.*, "Practical root cause localization for microservice systems via trace analysis," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[87] H. Abe and S. Tsumoto, "Analyzing behavior of objective rule evaluation indices based on a correlation coefficient," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2008, pp. 758–765.

[88] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[89] Z. Li, C. Luo, Y. Zhao, Y. Sun, K. Sui, X. Wang, D. Liu, X. Jin, Q. Wang, and D. Pei, "Generic and robust localization of multi-dimensional root causes," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 47–57.

[90] H. Kong, T. Li, J. Ge, L. Zhang, and L. Li, "Enhancing fault localization in microservices systems through span-level using graph convolutional networks," *Automated Software Engineering*, vol. 31, no. 2, p. 46, 2024.

[91] "Sock shop," https://github.com/microservices-demo/microservices-demo, 2024, accessed: 2024-08-21.

[92] "Train ticket," https://github.com/FudanSELab/train-ticket, 2024, accessed: 2024-10-03.

[93] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[94] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[95] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[96] X. Zhang, S. Ghosh, C. Bansal, R. Wang, M. Ma, Y. Kang, and S. Rajmohan, "Automated root causing of cloud incidents using in-context

learning with gpt-4," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024, pp. 266–277.

[97] D. Zhang, X. Zhang, C. Bansal, P. Las-Casas, R. Fonseca, and S. Rajmohan, "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*, 2024, pp. 388–398.

[98] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, "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, 2023, pp. 1737–1749.

[99] Y. Sui, Y. Zhang, J. Sun, T. Xu, S. Zhang, Z. Li, Y. Sun, F. Guo, J. Shen, Y. Zhang *et al.*, "Logkg: Log failure diagnosis through knowledge graph," *IEEE Transactions on Services Computing*, 2023.

[100] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "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*, 2019, pp. 683–694.

[101] S. Zhang, J. Zhu, B. Hao, Y. Sun, X. Nie, J. Zhu, X. Liu, X. Li, Y. Ma, and D. Pei, "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*, 2024, pp. 115–125.

[102] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, and L. Su, "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*, 2020, pp. 1387–1397.

[103] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2018.

[104] M. Ma, W. Lin, D. Pan, and P. Wang, "Self-adaptive root cause diagnosis for large-scale microservice architecture," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1399–1410, 2020.

[105] G. Yu, Z. Huang, and P. Chen, "Tracerank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems," *Journal of Software: Evolution and Process*, vol. 35, no. 10, p. e2413, 2023.

[106] Y. Li, G. Yu, P. Chen, C. Zhang, and Z. Zheng, "Microsketch: Lightweight and adaptive sketch based performance issue detection and localization in microservice systems," in *International Conference on Service-Oriented Computing*. Springer, 2022, pp. 219–236.

[107] J. Yang, Y. Guo, Y. Chen, and Y. Zhao, "Tracenet: Operation aware root cause localization of microservice system anomalies," in *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2023, pp. 758–763.

[108] S. Zhang, Y. Zhao, J. Lu, B. Lyu, S. Zhu, Z. Wang, J. Yang, L. He, and J. Wu, "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, 2021, pp. 367–377.

[109] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2762–2780, 2022.

[110] Y. Wu, B. Chai, Y. Li, B. Liu, J. Li, Y. Yang, and W. Jiang, "An empirical study on change-induced incidents of online service systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2023, pp. 234–245.

[111] Y. Cai, B. Han, J. Li, N. Zhao, and J. Su, "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, 2021, pp. 1–6.

[112] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 196–206.

[113] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 623–634.

[114] C. Hou, T. Jia, Y. Wu, Y. Li, and J. Han, "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, 2021, pp. 493–500.

[115] V. Murali, E. Yao, U. Mathur, and S. Chandra, "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, 2021, pp. 288–297.

[116] H. Jayathilaka, C. Krintz, and R. Wolski, "Performance monitoring and root cause analysis for cloud-hosted web applications," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 469–478.

[117] C. M. Rosenberg and L. Moonen, "Spectrum-based log diagnosis," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–12.

[118] L. Wang, N. Zhao, J. Chen, P. Li, W. Zhang, and K. Sui, "Root-cause metric location for microservice systems via log anomaly detection," in *2020 IEEE international conference on web services (ICWS)*. IEEE, 2020, pp. 142–150.

[119] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[120] A. Amar and P. C. Rigby, "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, 2019, pp. 140–151.

[121] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "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*, 2018, pp. 60–70.

[122] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection and root cause analysis for spark," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 389–396.

[123] H. Ikeuchi, A. Watanabe, T. Kawata, and R. Kawahara, "Root-cause diagnosis using logs generated by user actions," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[124] J. Ni, W. Cheng, K. Zhang, D. Song, T. Yan, H. Chen, and X. Zhang, "Ranking causal anomalies by modeling local propagations on networked systems," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1003–1008.

[125] J. Shi, S. Jiang, B. Xu, and Y. Xiao, "Serverrca: Root cause analysis for server failure using operating system logs," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 486–496.

[126] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, p. 110432, 2020.

[127] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu *et al.*, "How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1410–1420.

[128] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-Anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin *et al.*, "What bugs live in the cloud? a study of 3000+ issues in cloud systems," in *Proceedings of the ACM symposium on cloud computing*, 2014, pp. 1–14.

[129] H. Liu, S. Lu, M. Musuvathi, and S. Nath, "What bugs cause production cloud incidents?" in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019, pp. 155–162.

[130] Y. Zhao, L. Jiang, Y. Tao, S. Zhang, C. Wu, Y. Wu, T. Jia, Y. Li, and Z. Wu, "How to manage change-induced incidents? lessons from the study of incident life cycle," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 264–274.

[131] S. Ghosh, M. Shetty, C. Bansal, and S. Nath, "How to fight production incidents? an empirical study on a large-scale cloud service," in

Proceedings of the 13th Symposium on Cloud Computing, 2022, pp. 126–141.
[132] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar, "Why does the cloud stop computing? lessons from hundreds of service outages," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 1–16.
[133] X. Huang, H. Liu, Y. Wu, Y. Zhao, C. Wu, S. Zhang, L. Jiang, T. Jia, Y. Li, and Z. Wu, "Ocrcl: Online contrastive learning for root cause localization of business incidents," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2024, pp. 524–534.
[134] "Online boutique," https://github.com/GoogleCloudPlatform/microservices-demo, 2024, accessed: 2024-10-03.
[135] "Deathstarbench," https://github.com/delimitrou/DeathStarBench/tree/master, 2024, accessed: 2024-10-03.
[136] "Apache thrift," https://thrift.apache.org/, 2024, accessed: 2024-10-03.
[137] "Déjàvu dataset," https://zenodo.org/records/6955909, 2024, accessed: 2024-10-03.
[138] "Rcd dataset," https://github.com/azamikram/rcd/tree/master, 2024, accessed: 2024-10-03.
[139] "Changerca dataset," https://github.com/IntelligentDDS/ChangeRCA, 2024, accessed: 2024-10-03.
[140] "Baro dataset," https://zenodo.org/records/11046533, 2024, accessed: 2024-10-03.
[141] "Squeeze dataset," https://zenodo.org/records/8153367, 2024, accessed: 2024-10-03.
[142] "Dycause dataset," https://github.com/PanYicheng/dycause_rca/tree/main, 2024, accessed: 2024-10-03.
[143] "Grayscope dataset," https://gitee.com/milohaha/grayscope/tree/master, 2024, accessed: 2024-10-03.
[144] "Latentscope dataset," https://github.com/NetManAIOps/LatentScope, 2024, accessed: 2024-10-03.
[145] "Microcu dataset," https://github.com/jxrjxrjxr/MicroCU/tree/main, 2024, accessed: 2024-10-03.
[146] "Murphy dataset," https://github.com/netarch/Murphy-traces, 2024, accessed: 2024-10-03.
[147] "Aiops competition dataset," https://www.aiops.cn//, 2024, accessed: 2024-10-03.
[148] "Nezha dataset," https://github.com/IntelligentDDS/Nezha/tree/main, 2024, accessed: 2024-10-03.
[149] "Gaia dataset," https://github.com/CloudWise-OpenSource/GAIA-DataSet, 2024, accessed: 2024-10-03.
[150] "Eadro dataset," https://zenodo.org/records/7615394, 2024, accessed: 2024-10-03.
[151] "Gamma dataset," https://www.kaggle.com/datasets/gagansomashekar/microservices-bottleneck-detection-dataset, 2024, accessed: 2024-10-03.
[152] "Mepfl dataset," https://github.com/FudanSELab/Research-ESEC-FSE2019-AIOPS, 2024, accessed: 2024-10-03.
[153] "Github - phamquiluan/baro: [fse'24 - best artifact award] baro: Robust root cause analysis for microservice systems." https://github.com/phamquiluan/baro, 2024, accessed: 2024-10-27.
[154] "Github - intelligentdds/gied: Graph based incident extraction and diagnosis in large-scale online systems (ase'22)," https://github.com/IntelligentDDS/GIED, 2024, accessed: 2024-10-27.
[155] "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, 2024, accessed: 2024-10-27.
[156] "Github - fudanselab/deeptralog," https://github.com/FudanSELab/DeepTraLog, 2024, accessed: 2024-10-27.
[157] "Github - netmanaiops/chain-of-event," https://github.com/NetManAIOps/Chain-of-Event, 2024, accessed: 2024-10-27.
[158] "Github - netmanaiops/circa: Causal inference-based root cause analysis," https://github.com/NetManAIOps/CIRCA, 2024, accessed: 2024-10-27.
[159] "Github - intelligentdds/changerca," https://github.com/IntelligentDDS/ChangeRCA, 2024, accessed: 2024-10-27.
[160] "Github - azamikram/rcd: Root cause discovery: Root cause analysis of failures in microservices through causal discovery," https://github.com/azamikram/rcd, 2024, accessed: 2024-10-27.
[161] "Github - bebillionaireusd/eadro," https://github.com/BEbillionaireUSD/Eadro, 2024, accessed: 2024-10-27.
[162] "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, 2024, accessed: 2024-10-27.
[163] "Github - jxrjxrjxr/microcu," https://github.com/jxrjxrjxr/MicroCU, 2024, accessed: 2024-10-27.
[164] "Github - netmanaiops/tracerca: Practical root cause localization for microservice systems via trace analysis. iwqos 2021," https://github.com/NetManAIOps/TraceRCA, 2024, accessed: 2024-10-27.
[165] "Github - aiops-lab-nku/diagfusion," https://github.com/AIOps-Lab-NKU/DiagFusion, 2024, accessed: 2024-10-27.
[166] "Github - intelligentdds/microrank: Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," https://github.com/IntelligentDDS/MicroRank, 2024, accessed: 2024-10-27.
[167] "Github - netmanaiops/cmdiagnostor," https://github.com/NetManAIOps/CMDiagnostor, 2024, accessed: 2024-10-27.
[168] "Github - netmanaiops/squeeze: Issre 2019: Generic and robust localization of multi-dimensional root cause," https://github.com/lizeyan/Squeeze, 2024, accessed: 2024-10-27.
[169] "Github - intelligentdds/nezha: The implementation of multimodal observability data root cause analysis approach nezha in fse 2023," https://github.com/IntelligentDDS/Nezha, 2024, accessed: 2024-10-27.
[170] "Github - logpai/log3c: Log-based impactful problem identification using machine learning [fse'18]," https://github.com/logpai/Log3C, 2024, accessed: 2024-10-27.
[171] "Github - fengrui-liu/microcbr: Official repository for microcbr," https://github.com/Fengrui-Liu/MicroCBR, 2024, accessed: 2024-10-27.
[172] "Sieve by sieve-microservices," https://sieve-microservices.github.io/, 2024, accessed: 2024-10-27.
[173] "Github - intelligentdds/swisslog: The implementation of swisslog in issre'20 and tdsc'22," https://github.com/IntelligentDDS/SwissLog, 2024, accessed: 2024-10-27.
[174] "Opentelemetry ebpf profiler," https://github.com/open-telemetry/opentelemetry-ebpf-profiler, 2024, accessed: 2024-08-21.
[175] "Auto profiling," https://www.deepflow.io/docs/features/continuous-profiling/auto-profiling/, 2024, accessed: 2024-08-21.
[176] "Github issue of prometheus," https://github.com/prometheus/prometheus/issues/10445/, 2024, accessed: 2024-09-14.
[177] L. Zhang, Z. Xie, V. Anand, Y. Vigfusson, and J. Mace, "The benefit of hindsight: Tracing {Edge-Cases} in distributed systems," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 321–339.
[178] S. He, B. Feng, L. Li, X. Zhang, Y. Kang, Q. Lin, S. Rajmohan, and D. Zhang, "Steam: observability-preserving trace sampling," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1750–1761.
[179] Alibaba, "Design architecture," 2023, gitHub repository. [Online]. Available: https://help.aliyun.com/document_detail/2362206.html
[180] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," *arXiv preprint arXiv:1701.08546*, 2017.