The Journal of Systems and Software 216 (2024) 112145

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

MicroIRC: Instance-level Root Cause Localization for Microservice Systems ☆

Yuhan Zhu [a], Jian Wang [a,*], Bing Li [a], Yuqi Zhao [a], Zekun Zhang [a], Yiming Xiong [a], Shiping Chen [b]

[a] School of Computer Science, Wuhan University, Wuhan, China
[b] CSIRO Data61, Australia

ARTICLE INFO

Dataset link: https://github.com/WHU-AISE/MicroIRC.git

Keywords:
Microservice
Root cause localization
Graph neural network
Service instance

ABSTRACT

The use of microservice architecture is gaining popularity in the development of web applications. However, identifying the root cause of a failure can be challenging due to the complexity of interconnected microservices, long service invocation links, dynamic changes in service states, and the abundance of service deployment nodes. Furthermore, as each microservice may have multiple instances, it can be difficult to identify instance-level failures promptly and effectively when the microservice topology and failure types change dynamically. To address this issue, we propose MicroIRC (Instance-level Root Cause Localization for Microservice Systems), a novel metrics-based approach that localizes root causes at the instance level while exhibiting robustness to adapt to dynamic changes in topology and new types of anomalies. We begin by training a graph neural network to fit different root cause types based on extracted time series features of microservice system metrics. Next, we construct a heterogeneous weighted topology (HWT) of microservice systems and execute a personalized random walk to identify root cause candidates. These candidates, along with real-time metrics from the anomalous time window, are then fed into the trained graph neural network to generate a ranked root cause list. Experiments conducted on five real-world datasets demonstrate that MicroIRC can accurately locate the root cause of microservices at the instance level, achieving a precision rate of 93.1% for the top five results. Furthermore, compared to the state-of-the-art methods, MicroIRC can improve the accuracy of root cause localization by more than 17% at the service level and more than 11.5% at the instance level. Remarkably, it exhibits robustness in scenarios involving new failure types, achieving an accuracy of 84.2% for the top result amid dynamic topological changes.

1. Introduction

Microservice architecture has become increasingly popular due to its ability to provide high availability, scalability, and elastic scaling to dynamically changing workloads of software applications (Chen et al., 2023). As a result, more and more Web-based systems are designed and built using microservice architecture. However, due to the complexity of interconnected microservices, long service invocation links, dynamic changes in service states, and numerous service deployment nodes, it is difficult to locate the root cause of system abnormalities (Ma et al., 2022; Chen et al., 2019a; Mariani et al., 2020). Ensuring the reliability of large microservice systems has thus become an urgent and challenging issue (Wang et al.; Qiu et al., 2020; Kandula et al., 2009; Marwede et al., 2009).

Towards this challenge, various root cause analysis methods (Soldani and Brogi, 2023; Liu et al., 2019; Gan et al., 2021; Yu et al., 2021; Zhou et al., 2019; Du et al., 2017a; Yuan et al., 2019; He et al., 2018a; Lin et al., 2016) have been proposed. These methods leverage different kinds of data for observability, including metrics, logs, and traces. Among them, metrics receive the most attention (Wu et al., 2021b). Existing metrics-based root cause analysis approaches can be approximately categorized as graph theory-based methods (Akoglu et al., 2015; Wu et al., 2021a; Zhang et al., 2021; Chen et al., 2019b; Aubet et al., 2018; Kim et al., 2013; Wang et al., 2018; Ma et al., 2019; Wu et al., 2020b; Ma et al., 2020; Zhang et al., 2020; Meng et al., 2020; Lin et al., 2018) and machine learning-based methods (He et al., 2022; Chen et al., 2022; Li et al., 2022a,b; Wu et al., 2020a; Brandón et al., 2020; Wang et al., 2019; Weng et al., 2018; Du et al., 2018; Liu et al., 2021). These approaches analyze multi-dimensional metrics data collected from services, virtual machines, and physical machines to identify root causes. While these works have made significant progress, they still face certain limitations, such as excessively large localization granularity and the inability to handle new types of anomalies that arise in microservice systems.

☆ Editor: Dario Di Nucci.

* Corresponding author.
E-mail addresses: jianwang@whu.edu.cn (J. Wang), bingli@whu.edu.cn (B. Li).

https://doi.org/10.1016/j.jss.2024.112145
Received 21 November 2023; Received in revised form 11 June 2024; Accepted 18 June 2024
Available online 22 June 2024

The necessity of root cause localization at the instance level. After containerization technologies have been widely adopted (Zhou et al., 2023; Usman et al., 2022), microservice systems usually use container orchestration tools, such as Kubernetes, for operation and management. These tools facilitate the rapid deployment of multiple instance replicas for a single microservice, route requests to appropriate replicas via load balancing, and adjust the number of replicas based on request volume and available resources (Zhao et al., 2022). Typically, different replicas are deployed on various physical machines (Fu et al., 2022), necessitating comprehensive troubleshooting across all replicas of a microservice to identify the root cause of failures. The failure of even a single instance can impact the entire microservice system due to the load balancing strategy employed. Service-level root cause localization methods, lacking detailed metrics data for individual instances, require instance-by-instance troubleshooting to pinpoint the root cause.

Risks from new types of anomalies. The emergence of new types of anomalies in microservice systems poses a challenge for existing root cause localization methods, particularly those relying solely on historical data in conjunction with chaos engineering techniques (Zhang et al., 2022) for model training.While chaos engineering tools can realistically simulate various failures, they do not comprehensively cover all types of failures encountered in real-world scenarios. Consequently, if a root cause localization method relies exclusively on historical failure data for training, the model may yield unpredictable results when confronted with new types of anomalies not anticipated by chaos engineering. This leads to an inability to accurately locate the root cause.

Challenges posed by the dynamic creation and destruction of multiple-instance replicas. To guarantee the quality of service under different load pressures while minimizing resource consumption, microservice systems usually use autoscaling to dynamically create and destroy instance replicas (Cheng et al., 2023; Tong et al., 2023; Meng et al., 2023). In addition, strategies such as restarting or scaling are commonly implemented to mitigate the system impact of instance failures (Xie et al., 2023; Baarzi and Kesidis, 2021). However, these practices contribute to a constantly evolving microservice topology, complicating the task of tracking dynamic changes related to the root cause of failures and accurately pinpointing the root cause.

To address the above issues, we propose MicroIRC (Instance-level Root Cause Localization for Microservice Systems), a novel metrics-based approach designed to localize root causes at the instance level suitable for dynamic topologies while exhibiting robustness to adapt to new types of anomalies that may arise in microservice systems. To capture fine-grained time-series metrics features, we first simulate failure injection and collect multidimensional metrics data containing instance-level metrics. We then design a graph neural network model named MetricSage to fit the metrics features within each anomalous time window. MetricSage employs graph convolution to aggregate metrics features across the whole system. A heterogeneous weighted topology is built, taking advantage of real-time topology information and highlighting microservice instances. By using a personalized random walk algorithm in conjunction with the graph neural network model, MicroIRC achieves a more resilient and fine-grained localization of root causes. Experiments conducted on five real-world datasets demonstrate that MicroIRC can accurately locate the root cause of microservices at the instance level, achieving a precision rate of 93.1% for the top five results. Furthermore, when compared to the state-of-the-art methods, MicroIRC can improve the accuracy of root cause localization by more than 17% at the service level and over 11.5% at the instance level. In the case of dynamic changes in topology, the accuracy can also reach 84.2% for the top result. Remarkably, it exhibits robustness when new types of anomalies occur.

The main contributions of this paper are summarized as follows:

- We design a novel graph neural network-based model that uses time-series sampling points as nodes to learn time-series metrics features. When combined with multi-dimensional metrics data, the model can accurately capture instance-level metrics features.
- We propose MicroIRC, a system that combines a personalized random walk with a graph neural network model to pinpoint root causes at the instance level. We show that MicroIRC is adept at adapting to dynamic changes in topology and capable of addressing emerging failure types, ensuring its applicability in dynamic microservice systems.
- We conduct comprehensive experiments using five real-world datasets, which demonstrate the effectiveness of the proposed MicroIRC with injected failures and emerging failure types at both the instance and service levels. Our code has been released.

The remainder of the paper is organized as follows. Section 2 demonstrates the motivation examples to further address the problem. Section 3 describes related work on root cause localization. In Section 4, the details of the proposed MicroIRC are presented. Section 5 performs the experimental evaluation. Section 6 discusses limitations and threats to validity. Section 7 summarizes the work and puts forward the future work.

2. Motivation

This section presents empirical analysis on the three challenges mentioned above.

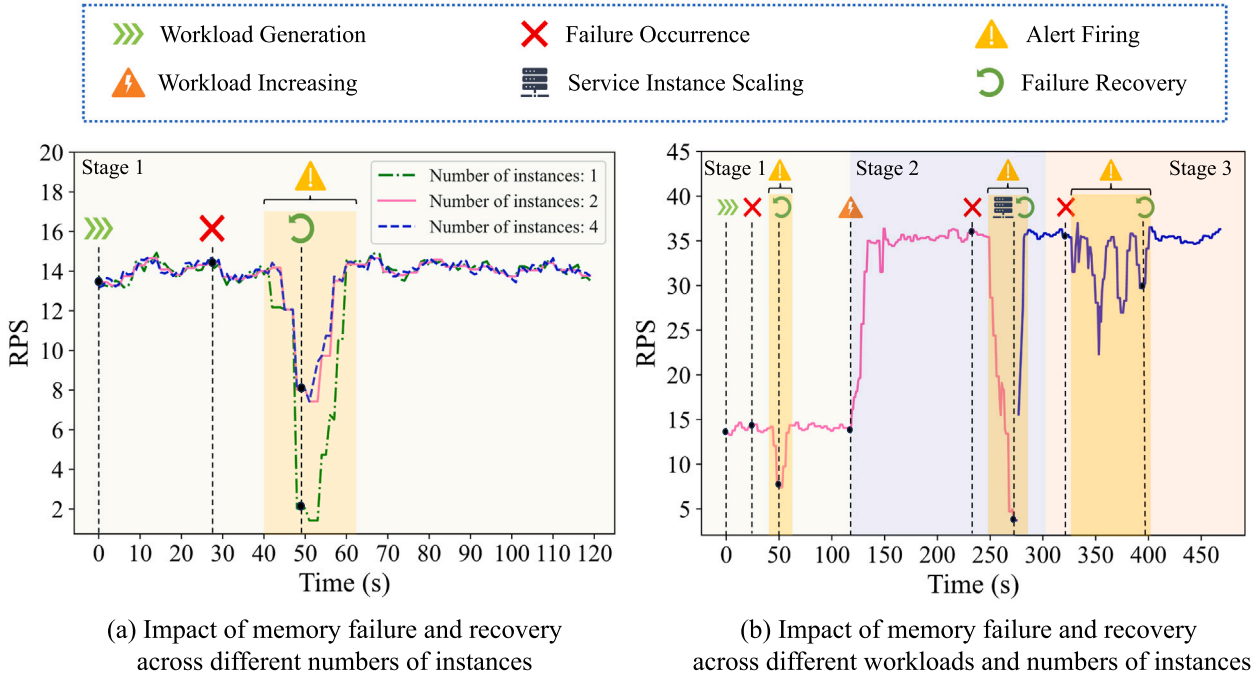
Impact of a single instance failure under different workloads and numbers of instances. We conducted experiments to assess the necessity of localizing the root cause at the instance level within a microservice system deployed on Kubernetes, a container orchestration platform. Our experimental setup included a master server equipped with an Intel Xeon (Ice Lake) Platinum 8369B, 8G RAM, and 4vCPU, along with two worker servers, each powered by an Intel Xeon (Ice Lake) Platinum 8369B, 32G RAM, and 4vCPU. We deployed an example microservice system Bookinfo[1] and initially simulated 100 users. Each service within this system was configured with multiple instances to adapt to changing workloads, as described in previous work (Yu et al., 2019).

In the first stage, following the generation of a workload, we simulated real-world scenarios by randomly injecting memory pressure into some instances. Then, we collected the request our findings suggest that merely increasing the system's scalability by adding more instances does not compensate for the need to accurately identify the true root cause of the failure per second (RPS) metric of the microservice system to monitor its quality of service (QoS). As illustrated in Fig. 1(a), we observed that the RPS significantly declined in the event of a failure, irrespective of the number of service instances, which ranged from 1 to 4. Furthermore, our findings suggest that merely increasing the system's scalability by adding more instances does not compensate for the need to accurately identify the true root cause of the failure.

In Stage 2, as depicted in Fig. 1(b), we conducted a more in-depth analysis of the impact of increased system workload on the issue by doubling the workload on the system and injecting the same failure. The results revealed a pronounced increase in the magnitude of deviation and a sharper decline in the RPS, suggesting that the problem worsens with heavier workloads. We further simulated Site Reliability Engineers (SREs) to address this scenario. Upon detecting a failure — evidenced by a decrease in RPS and the triggering of an alarm — we swiftly increased the number of instances from 2 to 4. This adjustment aimed to maintain the request load per instance at the same level as observed in Stage 1. This intervention successfully mitigated the failure, underscoring the effectiveness of our scaling approach in addressing the problem.

Furthermore, our investigation delved into the impact of the number of instances. In Stage 3, under the same workload, we re-injected the same failure. Contrary to expectations, the doubling of the number

[1] https://istio.io/docs/examples/bookinfo.



(a) Impact of memory failure and recovery across different numbers of instances

(b) Impact of memory failure and recovery across different workloads and numbers of instances

Fig. 1. An example of multiple failure occurring and recovery across different workloads and numbers of instances.

of instances did not prevent the system's RPS metric from dropping to a level comparable to that observed in Stage 1 during the alert period, as illustrated in Fig. 1(b). This finding underscores the challenge of promptly adjusting load-balancing policies in response to the state of multiple instances within existing microservice systems. Particularly in scenarios of unexpected high-traffic, the collapse of the entire microservice system usually starts with the failure of a single instance, which then gradually propagates across the entire system due to the dependencies among microservices (Chakraborty et al., 2023). Therefore, the ability to swiftly pinpoint the root cause at the instance level is crucial, enabling SREs to swiftly identify and address issues.

Insight 1: Merely focusing on service-level failures is insufficient. There must be an emphasis on utilizing instance-level monitoring metrics data to pinpoint the exact failed instance within multi-instance replicas.

Impact of new types of anomalies on root cause localization accuracy. Chaos engineering tools can simulate failures, yet they are unable to encompass all possible scenarios. This limitation presents challenges to root cause localization methods, especially when confronted with new anomalies that these tools have not accounted for. To illustrate the impact of such new anomalies on localization accuracy, we analyzed DejaVu (Li et al., 2022b), a state-of-the-art root cause localization method. Table 1 details the impact of different proportions of new anomalies on localization accuracy. According to the datasets[2] released by DejaVu, the introduction of merely 10% new anomalies leads to an average accuracy reduction of 28.08%. As the proportion of new anomalies increases, the accuracy of root cause localization further declines. Consider a practical scenario where a system functions smoothly under typical load conditions but encounters performance issues during high loads. Historical operational data may fail to capture such specific conditions adequately. The emergence of novel anomalies, previously unrecorded, renders existing methods ineffective in these situations. Moreover, in multi-instance environments, service topology undergoes changes with the scaling of instances. Historical data struggles to represent the frequently changing instance topologies adequately.

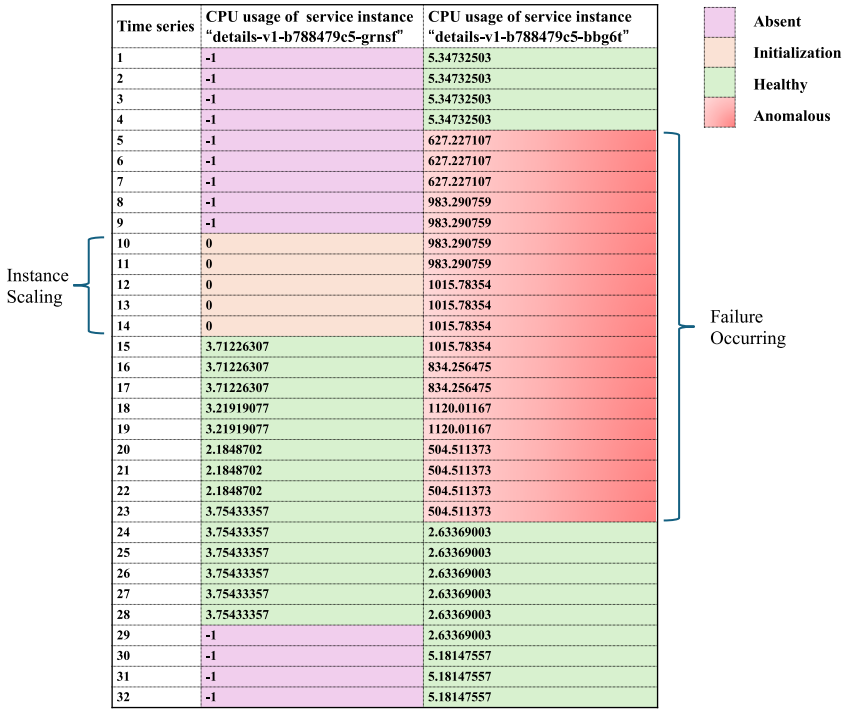
Table 1
Impact over different proportional new anomalies of DejaVu.

Proportion of new anomalies	Accuracy of root cause localization	Reduction percentage
0	0.773	–
0.1	0.556	28.08%
0.2	0.533	30.96%
0.3	0.444	42.47%
0.4	0.400	48.22%
0.5	0.378	51.10%
0.6	0.311	59.73%

Insight 2: Chaos engineering tools serve as means for simulating failures but fall short in encompassing all possible failure types. Consequently, methods that rely solely on supervised models face challenges in adapting to novel types of failures.

Impact of dynamic changes in topology when localizing root causes using metrics data. In a dynamically changing topology, instance metrics exhibit new features, as shown in Fig. 2. We categorize the dynamic features of instances into four states: Absent, Initialization, Healthy, and Anomalous, corresponding to values −1, 0, normal, and abnormal values, respectively. The CPU usage metric reflects the average CPU usage of each instance over the past minute, expressed as a percentage. Although normal CPU usage varies across instances, significant fluctuations are observed during failures, which we utilize to identify failing instances. In metrics feature extraction, we analyze the dynamic creation and destruction of instances using graph convolution alongside multi-instance pooling. This approach disregards the Absent feature's impact on the model, facilitating the exploration of dynamic topologies. Existing methods do not differentiate metrics features in dynamic environments or to explore potential correlations, such as the link between an instance's failure and the subsequent creation or destruction of service replicas.

[2] https://github.com/NetManAIOps/DejaVu.



Time series	CPU usage of service instance "details-v1-b788479c5-grnsf"	CPU usage of service instance "details-v1-b788479c5-bbg6t"
1	-1	5.34732503
2	-1	5.34732503
3	-1	5.34732503
4	-1	5.34732503
5	-1	627.227107
6	-1	627.227107
7	-1	627.227107
8	-1	983.290759
9	-1	983.290759
10	0	983.290759
11	0	983.290759
12	0	1015.78354
13	0	1015.78354
14	0	1015.78354
15	3.71226307	1015.78354
16	3.71226307	834.256475
17	3.71226307	834.256475
18	3.21919077	1120.01167
19	3.21919077	1120.01167
20	2.1848702	504.511373
21	2.1848702	504.511373
22	2.1848702	504.511373
23	3.75433357	504.511373
24	3.75433357	2.63369003
25	3.75433357	2.63369003
26	3.75433357	2.63369003
27	3.75433357	2.63369003
28	3.75433357	2.63369003
29	-1	2.63369003
30	-1	5.18147557
31	-1	5.18147557
32	-1	5.18147557

Fig. 2. An example of metrics features of dynamic changes in topology when failure occurring and instance scaling.

Insight 3: Variations in the number of instances, driven by changes in workload and the occurrence of failures, necessitate attention to dynamic topologies. This requirement underscores the need for root cause localization methods that are adaptable to such fluctuating scenarios.

3. Related work

Research on localizing the root cause of failure based on metrics often employs causality graphs or microservice topology graphs for analyzing metrics correlations and ranking root causes. These approaches can be divided into two categories: graph theory-based methods and machine learning-based methods.

3.1. Graph theory-based methods

Graph theory-based methods construct graph models to represent the dependencies between microservices (Akoglu et al., 2015). Many methods use causal inference between services to analyze root cause localization problems. For example, Aubet et al. (2018) modeled the microservice communication process as a casual graph model and constructed a learning phase to update the graph with information from vertices, neighbors, and edges. Similarly, MonitorRank (Kim et al., 2013) and CloudRanger (Wang et al., 2018) construct failure propagation graphs based on dynamic causality and locate root causes using a two-stage heuristic algorithm. Dycause (Pan et al., 2023) between user and kernel applications using a dynamic causal dependency model to locate root causes with user-state metrics, aiming to overcome the challenges posed by data noise in causal inference. MicroRCA (Wu et al., 2020b) enhances accuracy by linking abnormal performance to resource utilization, taking both microservices and host metrics into account. However, it lacks support for automatic updates of failure certainty. Weng et al. (2018) observed that multi-tier application failures could occur at both the service and physical levels, thus considering deployment level failures alongside service level ones. MicroHECL (Liu et al., 2021) mines service correlations and constructs service invocation relationship graphs from performance metrics, improving root cause localization efficiency by pruning unrelated service nodes based on trace data.

However, these studies only consider application-level metrics and do not comprehensively cover the root causes of failures. To address this limitation, CausIL (Chakraborty et al., 2023) incorporates load-balancing data from multiple instances of a single service into key metrics, leveraging key metrics to infer causal relationships that encompass instances. This approach demonstrates that instance-level metrics can reveal hidden correlations within the microservice topology, offering a more nuanced understanding of failure dynamics.

3.2. Machine learning-based methods

Machine learning-based methods train models based on microservice failure history and real-time metrics. By fitting the characteristic performance of different anomalous root causes in the metrics dimension, these methods can localize root causes of failures. For example, DejaVu (Li et al., 2022b) proposes an efficient combination of failure types and topology types by constructing failure dependency graphs and feature fitting of recurring failure types based on the graph attention network (GAT) to obtain high root cause accuracy, but it has poor support for new failure types. MicroEGRCL (Brandón et al., 2020) employs a dynamic service call graph and a graph neural network for root cause localization. It involves an attention mechanism to enhance edge features, constructs classical failure modes, and classifies failures by comparing the similarity between an anomaly and failure modes. Wu et al. (2020a) created a service dependency graph by utilizing a ranked list of reconstruction errors and then employed an autoencoder to detect abnormal service metrics. Du et al. (2018) collected real-time performance data and built a deep learning model for failure root cause classification. GRANO (Wang et al., 2019) uses machine learning and statistical methods for failure diagnosis and visualization. Weng et al. (2018) compared current and historical performance states to determine abnormal components. These machine learning-based approaches require a large scale of data for model training. The selection

of metrics likewise has a large impact on the effectiveness of root cause localization, leading to the difficulty of covering all potential root causes in the relevant models.

In addition to metrics-based methods, trace-based methods, such as TLCluster (Sun et al., 2023), Microrank (Yu et al., 2021), Sage (Gan et al., 2021), and TraceRCA (Li et al., 2021), face challenges in uniformly collecting trace data with comprehensive failure information. Although trace data can intuitively demonstrate service invocation relationships, these methods often necessitate alterations to the program's source code to gather complete trace data. Moreover, failures can change the code execution path, potentially omitting critical segments that report trace data, thereby creating incomplete links. Log-based methods (Huo et al., 2023; Soldani et al., 2022; Du et al., 2017b; He et al., 2018b), leveraging semantic analysis, can utilize business semantic logs and code exception execution stacks to penetrate into code-level failures. However, similar to trace-based methods, forming a uniform standard for log output remains challenging. Most methods analyze logs by extracting common templates, but failures often manifest outside these commonalities, making it difficult to filter logs related to failures.

To summarize, while the majority of existing methods focus on service-level root cause identification, recent studies have begun to consider service instances. Nonetheless, CausIL (Chakraborty et al., 2023), for instance, treats service instance metrics data as merely supplementary for inferring causal relationships between services, without directly integrating instances into the problem modeling phase. Conversely, some approaches, such as the one discussed by Soldani et al. (2022), adopt a more intrusive method to analyze failure propagation between instances by leveraging log templates. Moreover, the application of existing root cause localization methods in scenarios of dynamic topology, particularly for metrics-based approaches, remains unexplored. This gap exists because current method model training predominantly relies on a stable topology. Our method shifts focus towards localizing the root cause in scenarios involving multiple microservice instances, addressing both recurring and new failure types, and detaches model training from dependency on stable topology. Meanwhile, leveraging Service Mesh (Sedghpour and Townend, 2022) and cloud-native[3] observability (Zhou et al., 2021) components, our approach enables non-intrusive localization of instance-level root causes solely based on metrics data.

4. The proposed method

In this section, we delve into the details of MicroIRC. MicroIRC makes comprehensive use of multidimensional metrics of microservice systems and ultimately determines the root cause for anomalies through a personalized random walk combined with a graph neural network model.

In a microservice system, let $\mathcal{S}$ denote the set of microservices, $\mathcal{SI}$ represent the set of all instances across these microservices, and $\mathcal{N}$ define the set of all hosts. Furthermore, we categorize the various failure types injected into the system as set $\mathcal{A}$. The root cause localization process aims to evaluate and rank each service node ($s \in \mathcal{S}$) and instance node ($si \in \mathcal{SI}$) within the failure subgraph based on their likelihood of being the root cause. The node with the highest score is identified as the root cause, denoted by r, along with its associated failure type, denoted by a. Consequently, the final root cause result is represented as:

$$\mathcal{R} = \left\{ r_i, a_j | max\left(Rank_{r_i} \right), r_i \in \mathcal{S} \cup \mathcal{SI}, a_j \in \mathcal{A} \right\}. \tag{1}$$

4.1. Method framework

As shown in Fig. 3, the proposed MicroIRC consists of three main steps. Step 1 involves utilizing a monitoring system to simulate the injection of various failure types into the system. The metrics data from these simulated failures are then classified and labeled according to their respective failure types. Subsequently, MetricSage, a graph convolutional neural network inspired by GraphSage (Hamilton et al., 2017), is trained with historical metrics data. This training process aims to facilitate root cause analysis for different failure types at the instance level. Upon the occurrence of a failure, Step 2 collects real-time failure data and service call data. A subset of microservice heterogeneous topology is constructed by abnormal service call data. Real-time metrics data are then utilized to assign weights, creating a weighted microservice heterogeneous topology. In Step 3, a personalized random walk algorithm is executed on the weighted heterogeneous topology obtained in Step 2 to derive a root cause candidate set. This candidate set and the real-time data features are then fed into MetricSage for feature weighting. The outcome of this analysis is the identification of the final root cause and the corresponding failure type.

4.2. Method details

The details of each step in the proposed MicroIRC are described as follows.

4.2.1. Metrics modeling and model training

When a microservice system is running, we simulate the microservice traffic and inject multidimensional failures of different types into the service cluster using chaos engineering tools. We collect multidimensional metrics data (including service-level, instance-level, and host-level) for the entire cluster during different failure injection time windows. The details of multidimensional failures and multidimensional metrics data are shown in Section 5.1. We define each failure injection time window, along with the intervals capturing the complete fluctuation of metrics data, as the anomalous time window. Within each anomalous time window, we will collect metrics data individually at fixed intervals (for example, every five seconds) to be used as time-series metrics source data for the current failure. All subsequent methodological modeling and root cause localization analyses are conducted within these defined anomalous time windows.

For the service-level metrics data, we define the microservices at both sides of each response message as s_1 and s_2. Then, the single microservice response time data at a certain moment can be expressed as $[s_1, s_2]$. All the microservice response time data with an invocation relationship will form a vector $\mathbf{x}(\mathcal{S})$, as shown in Eq. (2).

$$\mathbf{x}(\mathcal{S}) = \langle [s_1, s_2], \ldots, [s_k, s_l] \rangle. \tag{2}$$

The time-series matrix consisting of the microservice response time data vector $\mathbf{x}(\mathcal{S})$ sampled at a time interval t ($\{t_1, t_2, \ldots, t_p\}$) is denoted as $\mathbf{S}_t$.

$$\mathbf{S}_t = \begin{bmatrix} [s_{1t_1}, s_{2t_1}], & \cdots, & [s_{kt_1}, s_{lt_1}] \\ \cdots, & \cdots, & \cdots \\ [s_{1t_p}, s_{2t_p}], & \cdots, & [s_{kt_p}, s_{lt_p}] \end{bmatrix} = \begin{bmatrix} \mathbf{x}(\mathcal{S})_{t_1} \\ \cdots \\ \mathbf{x}(\mathcal{S})_{t_p} \end{bmatrix}. \tag{3}$$

Regarding the instance-level metrics data, we set each service instance belonging to microservice s_m as si_{mn}. We collect metrics data, including CPU, memory, network, QPS, and success rate for each microservice instance, and denote them as si_{mn_C}, si_{mn_M}, si_{mn_N}, si_{mn_Q}, si_{mn_S}, respectively. The metrics data vector of instance si_{mn} at a certain moment can be expressed as:

$$\mathbf{x}\left(si_{mn}\right) = \langle si_{mn_C}, si_{mn_M}, si_{mn_N}, si_{mn_Q}, si_{mn_S} \rangle. \tag{4}$$

The metrics data is sampled at a time interval t ($\{t_1, t_2, \ldots, t_p\}$) and represented by the time-series metrics feature matrix $\mathbf{SI}_t$ of all service instances as:

$$\mathbf{SI}_t = \begin{bmatrix} \left[\mathbf{x}\left(si_{11}\right), & \cdots, & \mathbf{x}\left(si_{mn}\right)\right]_{t_1} \\ & \cdots & \\ \left[\mathbf{x}\left(si_{11}\right), & \cdots, & \mathbf{x}\left(si_{mn}\right)\right]_{t_p} \end{bmatrix}. \tag{5}$$

[3] https://www.cncf.io.

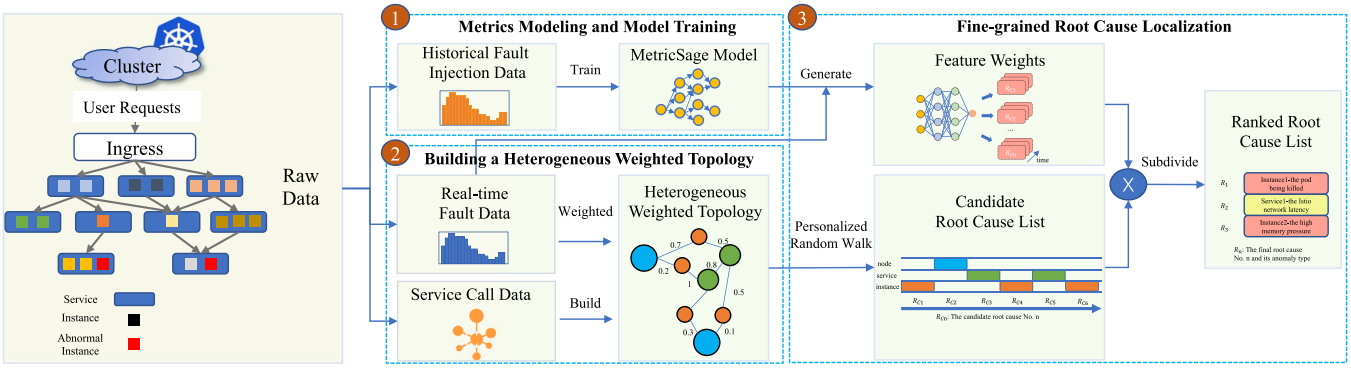


Fig. 3. Overall framework of MicroIRC.

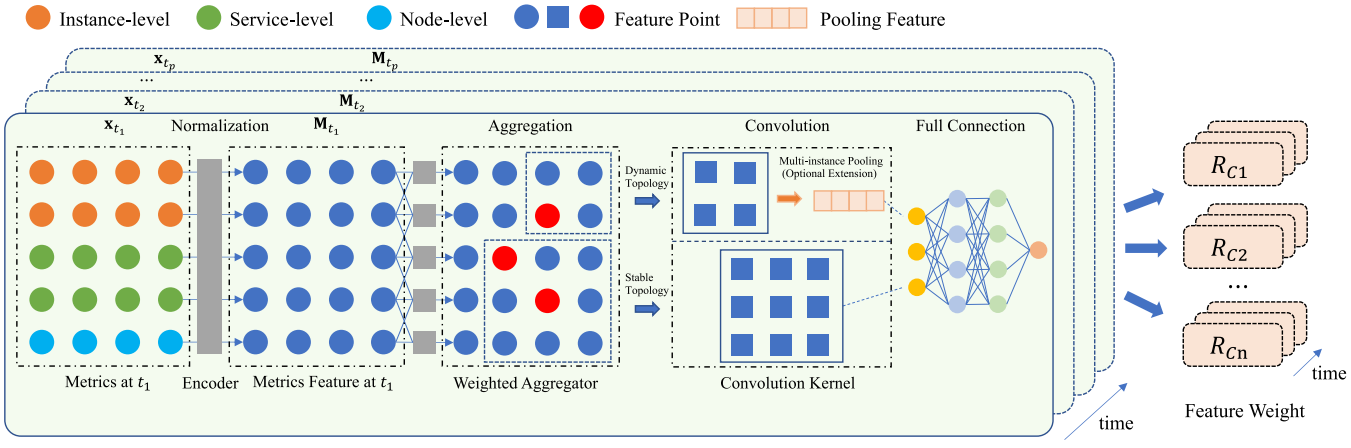


Fig. 4. Structure of MetricSage.

As for the host-level metrics data, we designate each host node as n_k and collect its metrics data of CPU, network, and memory as n_{kC}, n_{kM}, and n_{kN}, respectively. At each given moment, the vector of individual time-series metrics data for node n_k can be represented as:

$$\mathbf{x}\left(n_k\right) = \langle n_{kC}, n_{kM}, n_{kN} \rangle. \tag{6}$$

The metrics feature matrix $\mathbf{N}_t$ that samples the metrics data from all host nodes at time interval t is expressed as:

$$\mathbf{N}_t = \begin{bmatrix} \mathbf{x}\left(n_1\right)_{t_1}, & \cdots, & \mathbf{x}\left(n_k\right)_{t_1} \\ \cdots & \cdots & \cdots \\ \mathbf{x}\left(n_1\right)_{t_p}, & \cdots, & \mathbf{x}\left(n_k\right)_{t_p} \end{bmatrix}. \tag{7}$$

The global time-series metrics within an anomalous time window can be expressed as:

$$\mathbf{X}_t = [\mathbf{S}_t, \mathbf{SI}_t, \mathbf{N}_t], t \in \mathcal{TW}. \tag{8}$$

where $\mathcal{TW}$ denotes the set of time windows with different injected failures.

The modeling process of MetricSage, a graph neural network model is illustrated in Fig. 4. The capability of graph neural networks to model both local and global patterns enables the exploration of time-series metrics to propagate and aggregate information between nodes. This allows for the identification of global metrics features across the entire microservice system as well as local metrics features between the anomalous service and its multiple instances pertaining to a particular failure type.

To extract features of a single node, a single sample point at t_p in an anomalous time window is input for normalization and constitutes a node $\mathbf{v}_{t_p}$ in the graph network. The feature of each node $\mathbf{v}_{t_p}$ contains all the metrics of the entire microservice system at the current moment, denoted as $\mathbf{x}_{t_p}$.

When the service topology is stable (i.e., the number of services, instances, and hosts are unchanged), the dimensions of the features at all moments are consistent ($\mathbf{x}_{t_p} \in \mathbb{R}^{|\mathbf{X}_t|}$). To accommodate dynamic changes in topology, we introduce multi-instance pooling as an adaptive extension during the graph convolution process. We pool the metrics of multiple instances of service s_m at the moment of t_p, denoted as:

$$\mathbf{x}\left(s_m\right)'_{t_p} = max\left(\mathbf{x}\left(si_{m1}\right)'_{t_p} \oplus \mathbf{x}\left(si_{m2}\right)'_{t_p} \oplus \cdots \oplus \mathbf{x}\left(si_{mn}\right)'_{t_p}\right), \forall si_{mn} \in s_m, \tag{9}$$

where $\mathbf{x}\left(si_{mn}\right)'_{t_p} = \tau\left(\mathbf{x}\left(si_{mn}\right)_{t_p}\right)$, τ is the instance-pooling mask function responsible for placing the instance features of the Absent type to 0 to avoid features of non-existing instances affecting the tuning of model parameters, and $\mathbf{x}\left(s_m\right)'_{t_p}$ denotes the features that are most likely to be anomalies among the multiple instances retained after this pooling layer. For all services S, the overall characteristics of node $\mathbf{v}_{t_p}$ after this pooling layer will be represented as follows:

$$\mathbf{x}'_{t_p} = \mathbf{x}\left(s_1\right)'_{t_p} \oplus \mathbf{x}\left(s_2\right)'_{t_p} \oplus \cdots \oplus \mathbf{x}\left(s_m\right)'_{t_p}, \forall s_m \in S \tag{10}$$

It is important to note that the pooling layer will allow subsequent metrics to characterize the dimensions down to the number of services, i.e., $\mathbf{x}'_{t_p} \in \mathbb{R}^{|S|}$, independently of the number of instances, thus applying to scenarios where the topology changes dynamically.

The feature nodes within the same anomalous time window are connected in chronological order. For example, for a feature node $\mathbf{v}_{t_q}$, we create the edge E to the set of all nodes $\left(\left\{\mathbf{V}_t \middle| t > t_q \wedge t \in \{t_1, t_2, \ldots, t_p\}\right\}\right)$

that are generated later than the current moment t_q in the graph network. These edges generate the neighbor set $N(\mathcal{V})$ for each node. The edges between the nodes within the same anomalous time window enable the graph network structure to capture and retain time series features. The encoding process for feature $\mathbf{x}_{t_p}$ of node $\mathbf{v}_{t_p}$ can be expressed as:

$$\mathbf{M}_{t_p} = normalize\left(\mathbf{x}_{t_p}\right), \mathbf{x}_{t_p} \in \mathbf{X}_t \tag{11}$$

where $normalize(\bullet)$ denotes the metrics for each dimension are normalized using the L2 paradigm, and $\mathbf{M}_{t_p}$ denotes the metrics feature after input into the encoding process. When considering all the sampling points within a single anomalous time window, the input data $\mathbf{X}_t$ undergoes the encoding process using the MetricSage model. This process generates the microservice cluster's metrics features $\mathbf{M}_t$ corresponding to that anomalous time window. Inspired by the word embedding model, the encoder receives a certain number of metrics features, which is similar to the number of vocabulary items. Additionally, the multidimensional metrics features of microservices for each time-series node are analogous to the feature vector of the vocabulary.

In MetricSage, the feature vector $\mathbf{x}_{t_p}$ of each node and the feature vector set ($\left\{\mathbf{x}_{t_u}, u \in N(\mathcal{V})\right\}$) of its neighboring nodes are input into the aggregator for convolutional aggregation. The aggregation process is performed iteratively for a number of sampling n (one of the graph network parameters). The kth round of aggregation for node $\mathbf{v}_{t_p}$ and its weight matrix $\mathbf{W}$ can be expressed as:

$$\mathbf{M}_{t_p}^{k} = \sigma\left(\mathbf{W} \cdot \left(\left\{\mathbf{M}_{t_p}^{k-1}\right\} \cup \left\{\partial \subseteq \left\{\mathbf{M}_{t_u}^{k-1}, \forall u \in N(\mathcal{V})\right\} \middle| |\partial| = n\right\}\right)\right) \tag{12}$$

where ∂ denotes randomly sampling n nodes from the set of neighboring nodes $N(\mathcal{V})$, and σ denotes the Rectified Linear Unit (ReLU Glorot et al., 2011) activation function. Each recursive process is called a "convolutional layer", and the number of convolutional layers can be selected according to the size of the microservice cluster. Due to the complexity of the metrics data, it is essential to balance the trade-off between the model training time and the degree of node feature aggregation.

The node metrics features are aggregated and trained to facilitate root cause subdivision based on different failure types resulting from failure injection. The resulting MetricSage model, denoted as $M(\mathcal{V}, \mathcal{E})$, is used for feature weighting in the second stage of root cause subdivision.

4.2.2. Building a heterogeneous weighted topology

The real-time topology of a microservice system can effectively represent the topological information of the system. Introducing the scalability of the number of instances implies the timeliness of the topology. In a real-time running system, it is necessary to reconstruct the heterogeneous topology each time to account for the changes in the number of instances. The first step is to collect real-time service call data $\mathcal{C}$ within the system. The data is then used to construct the system invocation topology graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, which characterizes the current operation status of the microservice system. When a failure is suspected, the response time of services is leveraged for anomaly detection based on the Birch (Zhang et al., 1996) clustering algorithm. If a failure is found, a failure subgraph is extracted around the services on both sides of the abnormal call latency. Next, we construct a heterogeneous service topology graph, denoted by $\mathcal{SG}(\mathcal{N}, \mathcal{E})$, by integrating the service-instance and instance-host connections. The resulting topology graph encompasses microservices, their instances, and the host nodes hosting the instance containers, as depicted in Fig. 5. This step focuses on identifying anomalous services and their instances, simplifying the process of localizing the root cause.

The strength of the numerical association between anomalous service node metrics and service instance metrics determines the weight of the linkage. A higher weight indicates a stronger association, increasing the likelihood of identifying the instance as the root cause during

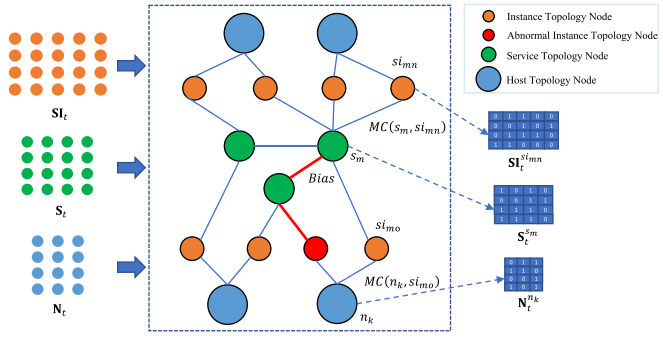


Fig. 5. Heterogeneous weighted topology.

subsequent random walk processes. When an abnormal service call latency is detected, a fixed value of $Bias$ is assigned as the maximum correlation value of the service call edge. To obtain the maximum correlation value for an edge $\left(v_{s_m}, v_{si_{mn}}\right)$ between topology nodes v_{s_m} and $v_{si_{mn}}$, we take the response time of microservices $\mathbf{S}_t^{s_m}$ and time series metrics of service instances $\mathbf{SI}_t^{si_{mn}}$ as inputs. When a service instance is running, its metrics data is affected by the resource data of the whole host. To calculate the correlation between the instance's metrics data $\mathbf{SI}_t^{si_{mo}}$ and its node's metrics data $\mathbf{N}_t^{n_k}$, we calculate the maximum value of correlation for the edge $\left(v_{si_{mo}}, v_{n_k}\right)$ between topological nodes $v_{si_{mo}}$ and v_{n_k}. This correlation value is then used as the edge weight of the instance and its node in the failure subgraph. The maximum value of the correlation (MC) can be expressed as:

$$MC(n_v, n_w) = \begin{cases} Bias, & n_v, n_w \in \mathcal{S} \text{ and } [n_v, n_w] \in [s_k, s_l]_a, \\ max(corr(a,b)), & \forall a \in \mathbf{S}_t^{n_v} \text{ and } \forall b \in \mathbf{SI}_t^{n_w} \text{ and } n_v \in \mathcal{S} \text{ and } n_w \in \mathcal{SI}, \\ max(corr(a,b)), & \forall a \in \mathbf{N}_t^{n_v} \text{ and } \forall b \in \mathbf{SI}_t^{n_w} \text{ and } n_v \in \mathcal{N} \text{ and } n_w \in \mathcal{SI}. \end{cases} \tag{13}$$

where $corr(\bullet)$ denotes the Pearson correlation function. We calculate and select the largest correlation among all the metrics dimensions as the correlation between the node pairs.

4.2.3. Fine-grained root cause localization

After all edges in the failure subgraph $\mathcal{SG}(\mathcal{N}, \mathcal{E})$ are weighted, we can get a weighted failure subgraph $\mathcal{SG}(\mathcal{N}, \mathcal{E}, \mathcal{W})$. To find the root cause node in the weighted failure subgraph, we use the random walk algorithm combined with the already obtained node correlations as a personalized array to distinguish the importance of different nodes among the random walks. The closer the distance to the centre of the failure and the stronger the correlation with it, the more likely it is to be the root cause. To consider multiple instances under a single service, service nodes and instance nodes are treated separately. Therefore, we assign MC between the instance si_{mn} and its corresponding failure service s_m to the personalization array of the instance node. For the service node s_m, we assign the average MC between it and each of its instances $\{si_{m1}, \ldots, si_{mn}\}$ to its personalization array. Similarly, for the host node n_k, we assign the average MC between it and each

of its instances $\{si_{m1}, \ldots, si_{mn}\}$ to its personalization array to consider implicit relationships between instances and hosts. The personalized process is expressed as:

$$P(n) = \begin{cases} MC(s_m, si_{mn}), & n \in \mathcal{SI}, \\ AVG\left(\sum MC(s_m, si_{mn})\right), & n \in \mathcal{S}, \\ AVG\left(\sum MC(n_k, si_{mn})\right), & n \in \mathcal{N}. \end{cases} \tag{14}$$

Afterward, the personalized random walk algorithm is executed. The equation of calculating a personalized random walk is expressed as:

$$v = (1-c)Pv + cu, \tag{15}$$

where v denotes the final scoring result of nodes, i.e., the ranking of root cause localization results, c is the probability of continuing forward for a random walk, P is the personalization array, which is used as a parameter to influence the probability that a random walk will restart from each node, and u is the scoring result of the next node. After multiple rounds of random walks, the scoring results of each node will tend to converge, resulting in a set of root cause candidates.

To enhance the precise identification of failure types and the real root causes from the metrics features of the entire microservice system, we use the MetricSage model $M(\mathcal{V}, \mathcal{E})$. The root cause candidate list R_C and the real-time metrics data $\mathbf{X}_{in-time}$ are input into the model for further feature weighting.

Specifically, the time-series metrics features within the real-time anomalous time window are fed into the graph neural network model, which produces subdivision weights (SW). As similarity weights between historical failure classifications and real-time metrics data, the final root cause weights are then calculated by multiplying the candidate set weights with the SW, as shown in Eq. (16). This step of weighting calculation is critical in deriving the ultimate results from the candidate list.

$$R = R_C \times SW. \tag{16}$$

After the feature weighting process is completed, the ranked root cause list R provides information on the instance-level root cause and the corresponding failure types.

5. Experiments

To examine the performance of the proposed MicroIRC, we conducted series of experiments to answer the following research questions.

- RQ1. Can MicroIRC outperform state-of-the-art methods regarding localization accuracy?
- RQ2. Does the use of MetricSage improve localization accuracy by uncovering hidden information?
- RQ3. How does MicroIRC compare to state-of-the-art methods in identifying and localizing new types of anomalies?
- RQ4. Can MicroIRC accurately localize the root cause in the case of dynamic changes in topology?
- RQ5. How does MicroIRC perform in terms of efficiency?
- RQ6. What are the key factors that influence the performance of MicroIRC?

To answer RQ1, we explored the accuracy of MicroIRC in multi-level root cause localization and compared it with state-of-the-art methods. To answer RQ2, we removed the MetricSage part and conducted ablation experiments to observe the difference in accuracy before and after ablation. To answer RQ3, we explored the applicability and robustness of our model and other models for new types of anomalies by randomly selecting a proportion of root causes for model training. To answer RQ4, we validated the accuracy of the MicroIRC method in a dataset with failures in the changing topology. To answer RQ5, we investigated the efficiency of root cause localization at the instance level to explore its timeliness in real-world usage scenarios. To answer RQ6, we performed a segmented experimental analysis to identify key factors influencing the performance of MicroIRC.

5.1. Datasets

The failure scenarios in the selected datasets consist of classical failure types observed from real systems, covering multiple levels of services, instances, and hosts. The details for each dataset are displayed in Table 2.

Datasets A and B were obtained from the released datasets.[4] They were collected from a real-world production microservice system, consisting of eight instances deployed on six hosts. These two datasets primarily contain metrics data for instance containers, hosts, and databases, rather than services. They will be mainly used in subsequent experiments for instance-level root cause localization.

Dataset C was derived from Google's e-commerce microservice system, Online Boutique.[5] Dataset E was derived from the example microservice system of Istio, called Bookinfo.[6] These microservice systems are deployed on a private cloud, and the hardware and software settings are detailed in Table 3. In datasets C and E, we used a Locust,[7] script to simulate the microservice traffic, and the page visit fluctuation of Wikipedia[8] on March 16, 2015, was used as the microservice entrance traffic and isotropically compressed to one hour. We then injected failures of different types into the service cluster using Chaos Mesh[9] a chaos engineering tool that can bring various types of failure simulation and helps microservice systems conveniently simulate various failures that might occur in reality during the development, testing, and production environments to find potential problems. In dataset C, we controlled the duration of each failure injection to be two minutes. In dataset E, we controlled the duration of each failure injection to be 3 min, and we scaled up and down one replica of the service where the root cause instance is located with 1 min before and after the failure to simulate the dynamic change of the topology. Therefore, dataset E will only serve as a validation of MicroIRC in terms of topological dynamics. After that, we used the Istio service mesh[10] integrated with Prometheus[11] to collect multidimensional metrics data, as shown in Table 4. The service call latency data, instance-level data, and node-level data serve as inputs to the MetricSage model. Service RPS (requests per second) and success rate are crucial metrics for evaluating service quality. The microservices' topology structure data is used to construct HWT. This data, along with the number of instances, is treated as time series data, collectively reflecting changes in the topology structure. Dataset D is randomly sampled from the 2022 CCF International AIOps Challenge Event dataset.[12] Datasets C and E, along with our proposed MicroIRC, have been released on GitHub[13] for reproducibility.

5.2. Baseline approaches

We conducted a comparison between MicroIRC and state-of-the-art methods at different root cause localization levels. DejaVu, the latest supervised method, provides a complete dataset from which Datasets A and B are also derived, where docker containers are equated to instances. CausIL is the latest causality mining method that considers

[4] https://github.com/NetManAIOps/DejaVu.
[5] https://github.com/GoogleCloudPlatform/microservices-demo.
[6] https://istio.io/docs/examples/bookinfo.
[7] https://github.com/locustio/locust.
[8] https://datahub.io/dataset/english-wikipedia-pageviews-by-second.
[9] https://chaos-mesh.org.
[10] https://istio.io/.
[11] https://prometheus.io/.
[12] https://www.aiops-challenge.com/.
[13] https://github.com/WHU-AISE/MicroIRC.git.

Table 2
Datasets statistics.

Datasets	Failure types	Failure classes	Sample sizes	Metrics types	Sampling time interval (s)
A	CPU fault Network delay Network package loss Db connection limit Db close	22	1,415,766	418	60
B	CPU fault Network delay Network package loss Db connection limit Db close	18	1,538,658	496	60
C	Container CPU hog Istio network latency Container network packet loss Container memory leak Container process termination	17	1,165,956	146	5
D	Container CPU hog Container network latency Container network packet loss Container memory leak Container process termination	18	1,006,962	146	60
E	Container CPU hog Container network latency Container memory leak	18	194,724,000	601	5

Table 3
Settings of the cloud environment.

Hardware settings			
Attribute	Master	Node1	Node2
OS	Ubuntu 20.04	Ubuntu 16.04.7	Ubuntu 20.04
CPU (kernel)	16	8	24
Memory (GB)	64	28	128
Software settings			
Kubernetes	Istio	Locust	Chaos mesh
1.20.4	1.13.4	2.10.1	2.1.2

Table 4
Dimensions of metrics at different levels collected by the monitoring tools.

Metrics types	Metrics dimensions
Service-level	Service call latency from caller, service call latency from callee, service call latency from network, request per second, success rate, number of instances.
Instance-level	CPU usage, CPU limit, memory usage, memory limit, memory usage rate, file system usage, network receive, network transmit.
Node-level	Network transmit, CPU usage, memory usage.
Microservice-system-level	Topology structure.

instance-level metrics data. Therefore, we compare the DejaVu, CausIL, and MicroIRC methods for instance-level root cause localization results. DyCause, MicroRCA, CloudRanger, and Monitor-Rank contain the most recent and classic metrics-based root cause localization methods, but these methods are only capable of localizing root causes up to the service level. We include the service-level results of DejaVu and MicroIRC for comparison. The details of the comparison methods are described as follows:

- DejaVu (Li et al., 2022b) (2022): It is a root cause localization method that uses GAT to aggregate features. It constructs a failure dependency graph by combining different topology types and failure types, which can be used to locate the root cause component with its failure type.
- CausIL (Chakraborty et al., 2023) (2023): It is a causal inference graph construction method tailored for microservice systems, taking service instances into consideration. We implemented the method in various ways with different dataset formats to locate the root cause.
- Dycause (Pan et al., 2023) (2023): It employs a dynamic dependency model and user state metrics to locate root causes.
- MicroRCA (Wu et al., 2020b) (2020): It empowers microservice topology graphs with metrics data from microservices and hosts and performs a personalized random walk to locate root cause services.
- CloudRanger (Wang et al., 2018) (2018): It is a root cause localization method based on the Pearson correlation algorithm to construct a service causal graph.
- MonitorRank (Kim et al., 2013) (2013): It performs failure-based clustering and an extended random walk to locate root causes, adding self-connected and back-propagating edges to the topology graph.

5.3. Parameter settings

When implementing MicroIRC, we accounted for the minimal fluctuations in metrics after normalization, especially noting the more significant instance fluctuations compared to those at the service level. Consequently, we established a failure tolerance level of 0.03 for services and 0.01 for instances, utilizing the Birch algorithm for anomaly detection as adopted in prior work (Wu et al., 2020b). The anomalous time window was set to 10 min on datasets *A*, *B*, *C*, and *D*, facilitating the complete collection of failure fluctuation metrics. For dataset *E*, we chose a time window of 15 min to fully document the dynamics of instance creation and destruction on top of the failure occurrence.

In heterogeneous topology, we set the failure weight to 0.8, assigning larger correlations to anomalous edges (ranging from 0 to 1). Additionally, we considered the top 10 candidate root causes in MetricSage, which are derived from personalized random walk results with a c value of 0.85 (the default value of the random walk algorithm). The graph neural network was sampled with 10 neighbors of each node, and the feature embedding dimension size was consistent with the cluster metrics type dimension, e.g., 146 in dataset *C*. The parametric discussion regarding the anomalous time window size, the candidate set size, and the sampling of neighboring nodes is elaborated upon in greater detail in RQ6. To maintain training efficiency while effectively aggregating node information, we set up two convolutional layers. The hidden layer dimensions were set to 64 and 32, respectively.

During model training, we used the Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of 0.1. We train the MetricSage model for 5000 epochs. To better fit metrics features within an anomalous time window as an input to the model, after randomly disrupting all time series, we use the number of samplings corresponding to a time window as the size of the batch size (i.e., 120 on datasets C and E, 10 on datasets A, B, and D). The implementation of MicroIRC was based on PyTorch 1.13.0. All the experiments except the efficiency experiment were conducted on a server running Windows 10 with an Intel Core i7-10700 2.90 GHz CPU, 32 GB RAM, and a 24 GB NVIDIA GeForce RTX 3090 GPU. The efficiency experiment is conducted on a MacBook Pro with an Apple M1 CPU and 16 GB of RAM to simulate SREs locating the root cause on their own desktops after the model is trained and metrics data is collected to be more realistic.

5.4. Evaluation metrics

To evaluate the performance of the method experimentally, we adopted three evaluation metrics, including $PR@K$, $AVG@N$, and Acc, to measure the experimental results.

$PR@K$ indicates the top K results in the root cause ranking list containing the real root cause. A higher value of $PR@K$ with a smaller K indicate a higher accuracy. Among them, $PR@1$, $PR@3$, and $PR@5$ are selected. $PR@K$ for root cause i in the root cause ranking list R within the given group G can be expressed as follows.

$$PR@K = \frac{1}{G}\sum_{g\in G}\frac{\sum_{i<K} R[i]\in V}{min(K,|V|)}. \tag{17}$$

$AVG@N$ is defined as the mean metric of $PR@K$:

$$AVG@N = \frac{\sum_{k=1}^{N} PR@K}{N}. \tag{18}$$

Acc indicates the position of the real root cause in the ranking list. A higher rank and a smaller root cause ranking list correspond to a larger Acc value. Acc of the root cause i in the ranking result of group G is calculated as follows with p-values (t-test) calculating based on Acc.

$$Acc = \frac{1}{G}\sum_{g\in G}\begin{cases}\frac{|V|-rank(i)+1}{|V|} & R[i]\in V\\ 0 & otherwise\end{cases} \tag{19}$$

5.5. Performance comparison (RQ1)

5.5.1. The instance level

The proposed MicroIRC can be adapted for root cause localization at different levels. However, due to the lack of metrics data for services in datasets A and B, it is unsuitable for MicroIRC to build heterogeneous weighted topology (HWT). Therefore, we constructed a topology containing only the instance and host nodes, which may slightly impact the accuracy of our method. Meanwhile, we implemented CausIL using either the instance-level metrics graph or topology graph with correlation enhancement (i.e., CausIL-correlation) or personalization enhancement (i.e., CausIL-personalization) in datasets A and B. In datasets C and D, we incorporated CausIL with the HWT (i.e., MicroIRC with CausIL's FGES part) through the personalized random walk. These variations in implementation allow for comprehensive comparisons with CausIL in terms of root cause localization.

Table 5 shows the accuracy metrics at the instance level over the four datasets. On average, MicroIRC achieves 77.25%, 87.43%, and 93.1% for $PR@1$, $PR@3$, and $PR@5$, respectively. The results show that MicroIRC outperforms the baseline methods in terms of instance-level root cause localization, exhibiting an improvement of 11.5% in $PR@1$, 13.43% in $PR@3$, and 16.73% in $PR@5$, on average across the four datasets. The p-value of the t-test is used to evaluate whether the observed improvements offered by our method over the competing methods occur by chance. All p-values are less than 0.05, except for the comparisons with DejaVu in datasets A and B, demonstrating that the differences between our approach and competing methods are statistically significant. This suggests that our approach is highly likely to be superior to others. In datasets A and B, due to the limited scale of the microservices system, the insufficient number of nodes in the graph modeling leads to a less significant improvement in performance.

DejaVu heavily relies on high data integrity, which means its effectiveness can be compromised in scenarios with limited metrics data available or unclear system component dependencies. In such cases as datasets C and D, DejaVu encounters difficulties in accurately localizing the root cause. On the other hand, CausIL performs well at uncovering hidden correlations between instances. However, in practical implementations, we observed that CausIL could be overly sensitive in discovering such correlations. This sensitivity leads to the proliferation of edges, which may lead to an uneven distribution of weights and distortion of the importance of nodes, thus affecting the accuracy of the final ranking result, particularly in $PR@1$.

Upon examining the metrics data and the results of root cause localization, we observed that in scenarios where a service contains multiple instances, it tends to be relatively easier to locate the root cause in a small community of that service and corresponding instances. However, it is hard to narrow down the final root cause further. To address this challenge, we design the HWT component in the MicroIRC method. This component enables the differentiation of different instances of the same service based on their respective metrics data. By incorporating it with MetricSage's ability to learn both global and local patterns simultaneously, we can address this problem. The improvement observed in different dataset formats validates this conclusion.

5.5.2. The service level

Due to the lack of metrics data for services in datasets A and B, we only conducted service-level root cause localization experiments on datasets C and D for comparison. The comparison results with existing approaches over different failure types including memory, CPU, and network are shown in Table 6. As most of these methods cannot localize root causes at the instance level, the evaluation metric used for performance comparison is the accuracy of root cause localization (Acc) at the service level.

According to Table 6, the proposed MicroIRC outperforms the competing methods, achieving a remarkable improvement in localization accuracy ranging from 17% to 55%. All p-values of t-test are less than 0.001, demonstrating that our approach has an almost 100% probability of being superior to others. One key factor contributing to this improvement is the use of instance-level metrics data, which provides a more comprehensive understanding of the service's runtime behavior. For example, as shown in Fig. 6, in the multi-instance scenario, the metrics fluctuation of the root cause instance usually occurs before the service metrics fluctuation, and when a dramatic fluctuation of the service metrics occurs, it actually indicates that the failure has been very serious and the failure of the instance has been going on for some time. The service-level fluctuation of metrics may be delayed. MicroIRC effectively addresses this aspect by incorporating instance-level metrics data into the analysis of services, which can precisely determine the failure type.

5.6. Ablation experiments (RQ2)

To investigate the extent of enhancement achieved by weighting features using MetricSage in the MicroIRC method, an ablation experiment was conducted to compare the ability of uncovering hidden information in instance-level root cause localization before and after weighting features.

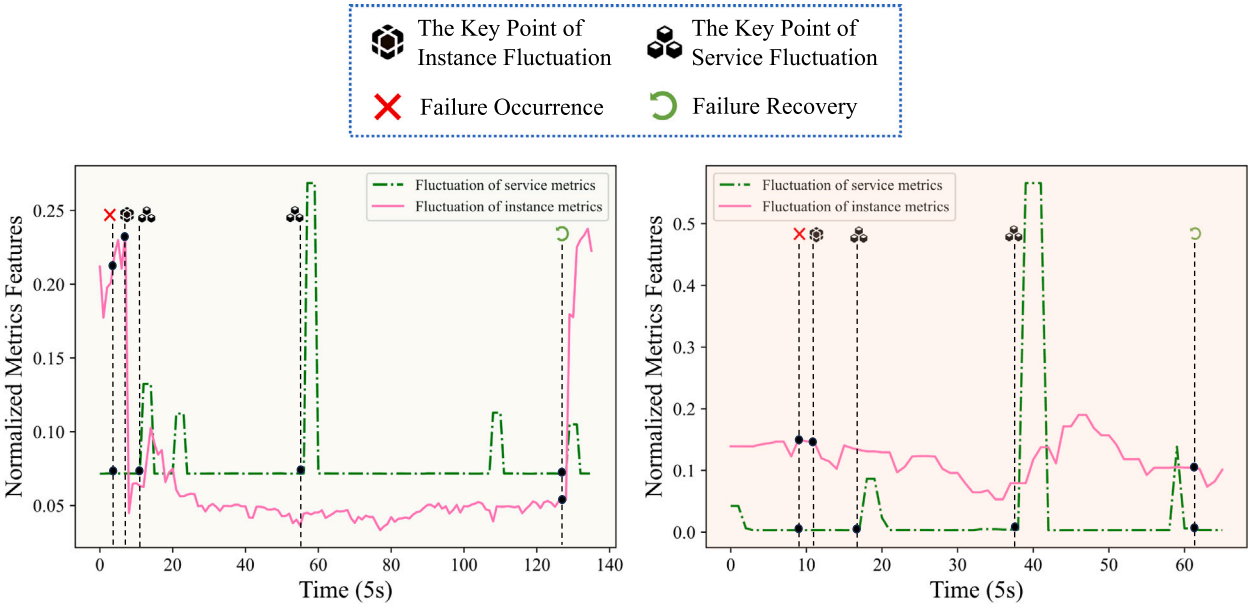
As shown in Table 5, across four datasets, MicroIRC with MetricSage exhibited substantial improvements in root cause localization performance. Specifically, compared to using MicroIRC without MetricSage (i.e., relying solely on the personalized random walk algorithm), MicroIRC improves by 56.76%, 50.5%, and 41.1% in terms of $PR@1$, $PR@3$, and $PR@5$, respectively. These results indicate that root cause

Table 5
Accuracy comparison at the instance level.

Datasets	Methods	Graph types	PR@1	PR@3	PR@5	AVG@5	Acc	p-value
A	MicroIRC	Topology	**0.935**	0.935	**0.968**	**0.944**	**0.980**	–
	DejaVu (Li et al., 2022b)	Topology	0.906	**0.937**	**0.968**	0.936	0.970	3.31e−1
	CausIL (Chakraborty et al., 2023)-correlation	Metrics	0.184	0.490	0.735	0.471	0.694	7.85e−4
	CausIL (Chakraborty et al., 2023)-personalization	Metrics	0.130	0.391	0.696	0.401	0.497	1.31e−4
	CausIL (Chakraborty et al., 2023)-correlation	Topology	0.140	0.349	0.535	0.335	0.434	1.25e−5
	CausIL (Chakraborty et al., 2023)-personalization	Topology	0.093	0.279	0.349	0.237	0.361	2.72e−4
	MicroIRC w/o MetricSage	Topology	0.219	0.250	0.313	0.250	0.411	5.54e−7
B	MicroIRC	Topology	**0.625**	0.708	**0.812**	**0.719**	**0.830**	–
	DejaVu (Li et al., 2022b)	Topology	0.574	**0.723**	0.787	0.691	0.781	2.79e−1
	CausIL (Chakraborty et al., 2023)-correlation	Metrics	0.203	0.516	**0.812**	0.494	0.689	2.88e−2
	CausIL (Chakraborty et al., 2023)-personalization	Metrics	0.203	0.500	**0.812**	0.490	0.645	1.11e−2
	CausIL (Chakraborty et al., 2023)-correlation	Topology	0.062	0.391	0.641	0.372	0.578	9.89e−3
	CausIL (Chakraborty et al., 2023)-personalization	Topology	0.094	0.266	0.484	0.272	0.479	2.04e−3
	MicroIRC w/o MetricSage	Topology	0.128	0.149	0.149	0.145	0.117	2.18e−4
C	MicroIRC	HWT	**0.833**	**0.944**	**0.944**	**0.911**	**0.939**	–
	DejaVu (Li et al., 2022b)	Topology	0.550	0.650	0.650	0.590	0.830	4.44e−3
	MicroIRC with CausIL's FGES part	HWT	0	0.667	0.833	0.612	0.692	1.62e−3
	MicroIRC w/o MetricSage	HWT	0.273	0.545	0.818	0.564	0.647	5.76e−4
D	MicroIRC	HWT	**0.697**	**0.910**	**1**	**0.891**	**0.903**	–
	DejaVu (Li et al., 2022b)	Topology	0.600	0.650	0.650	0.620	0.820	3.21e−2
	MicroIRC with CausIL's FGES part	HWT	0	0.364	0.545	0.433	0.561	1.04e−3
	MicroIRC w/o MetricSage	HWT	0.200	0.533	0.800	0.547	0.693	2.76e−3

Table 6
Accuracy comparison (*Acc*) at the service level.

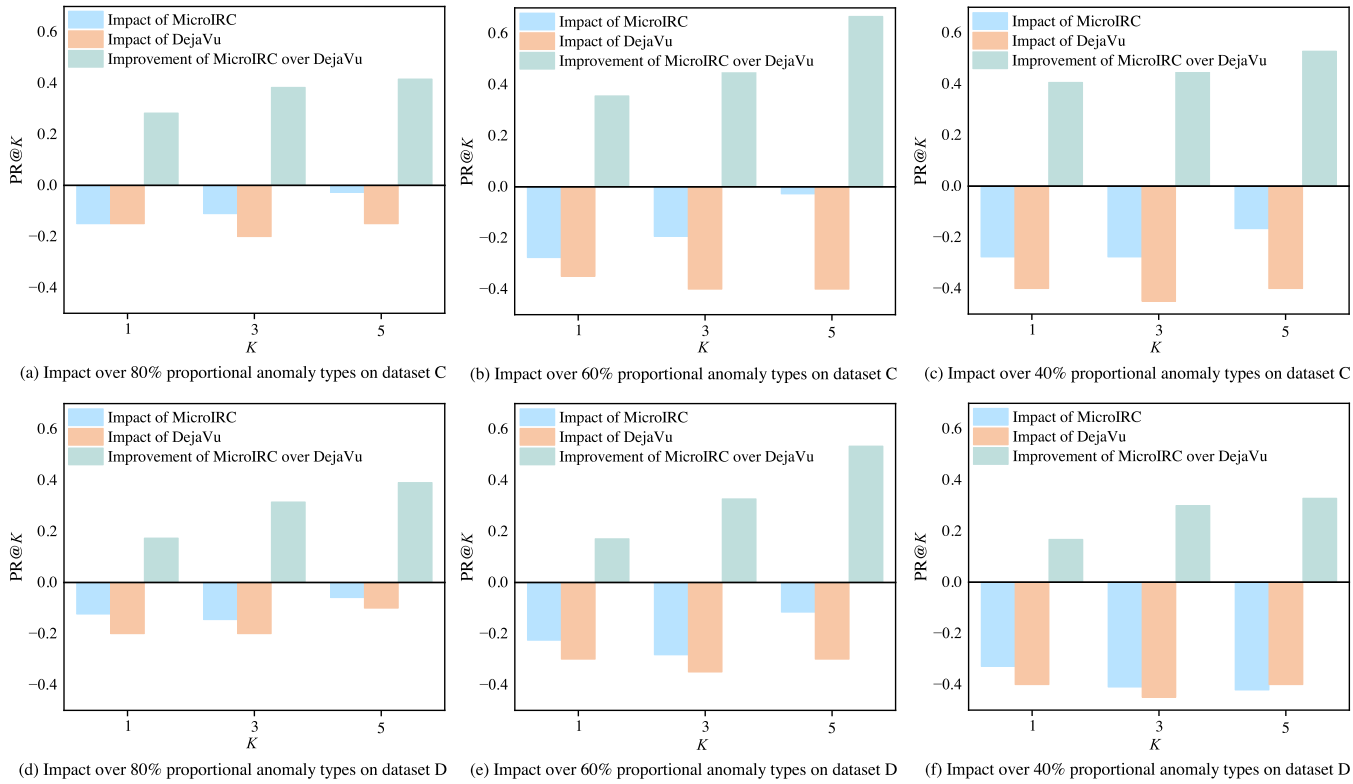
Methods	Memory		CPU		Network		p-value
	C	*D*	*C*	*D*	*C*	*D*	
MicroIRC	**0.931**	**0.893**	**0.945**	**0.913**	**0.938**	**0.903**	–
DejaVu (Li et al., 2022b)	0.747	0.682	0.719	0.629	0.734	0.732	4.55e−5
MicroRCA (Wu et al., 2020b)	0.755	0.701	0.707	0.588	0.733	0.721	2.09e−4
MonitorRank (Kim et al., 2013)	0.493	0.553	0.481	0.501	0.553	0.525	3.50e−6
CloudRanger (Wang et al., 2018)	0.393	0.511	0.433	0.488	0.421	0.479	1.00e−5
Dycause (Pan et al., 2023)	0.373	0.299	0.404	0.301	0.378	0.279	1.32e−7



(a) The fluctuation of metrics when the container process termination of paymentservice-6879f6c8c4-8pbjc occurs over the root cause instance and its service in dataset *C*

(b) The fluctuation of metrics when the container network package loss of cartservice-75d494679c-qdl5c occurs over the root cause instance and its service in dataset *C*

Fig. 6. Examples of how the instance-level metrics data provide a more comprehensive understanding of the service's runtime behavior.



(a) Impact over 80% proportional anomaly types on dataset C (b) Impact over 60% proportional anomaly types on dataset C (c) Impact over 40% proportional anomaly types on dataset C

(d) Impact over 80% proportional anomaly types on dataset D (e) Impact over 60% proportional anomaly types on dataset D (f) Impact over 40% proportional anomaly types on dataset D

Fig. 7. Impact over different proportional failure types.

localization is significantly improved by MetricSage, with particularly notable enhancements in $PR@1$ and $PR@3$.

One plausible explanation for the results lies in the utilization of MetricSage's graph convolution capability. By leveraging this capability, MicroIRC can learn both global and local metrics features, especially at the fine-grained root cause level, enabling the personalized random walk algorithm to target different types of root causes during its execution. Consequently, MetricSage can accurately classify root cause types that align with real-time system performance metrics in the candidate set.

Furthermore, the performance of the personalized random walk on HWT is improved compared to traditional topology, as evidenced by the results of datasets C and D in comparison to datasets A and B. This improvement can be attributed to the incorporation of instance-level metrics in the construction of the topology, further demonstrating that MicroIRC can effectively use instance-level metrics data to improve the accuracy of root cause localization.

5.7. Handling new types of anomalies (RQ3)

We explored the scenario of new types of anomalies on datasets C and D, where a single service has multiple instances and the topology of the microservice system is not fixed. This dynamic environment introduces variations in the container where each instance resides, leading to changes in the types of anomalies and the emergence of new failure types. RQ3 aims to explore the applicability of MicroIRC in this scenario and compare it with DejaVu, the best competing approach. Fig. 7 illustrates the accuracy of root cause localization when different proportions ($\{0.8, 0.6, 0.4\}$) of failure types were added to the model training.

As the proportion of failure types added to the training set decreases, the accuracy of MicroIRC experiences a slight decline, while DejaVu exhibits a more significant decrease. Notably, MicroIRC consistently outperforms DejaVu across different proportions of failure types. For instance, with a proportion of 80% failure types in the training set, MicroIRC outperforms DejaVu by 28.35%, 38.35%, and 41.65% on dataset C. Similarly, on dataset D, MicroIRC achieves improvements of 17.4%, 31.5%, and 39.1%. With a proportion of 0.6, on dataset C, MicroIRC achieves improvements of 35.6%, 50%, and 66.65%, while on dataset D, the improvements amount to 17.1%, 32.75%, and 53.4%. With a proportion of 0.4, on dataset C, MicroIRC improves by 40.6%, 46.7%, and 52.8%, and on dataset D, the improvements reach 16.8%, 30%, and 32.9%. Therefore, MicroIRC demonstrates robustness to dynamic changes in failure types.

5.8. Handling dynamic changes in topology (RQ4)

We validated MicroIRC on the dataset E with dynamic changes in topology, with the main goal of exploring whether the MetricSage model can fit dynamic metrics features. MicroIRC-TopoChange without MetricSage will serve as a baseline without using the MetricSage model for enhancement. The experimental results are shown in Table 7, where the addition of the optional dynamic extension of the MetricSage model (MicroIRC-TopoChange) resulted in 38.4% and 8.3% improvement of the MicroIRC method (MicroIRC-TopoChange without MetricSage) on $PR@1$ and $PR@5$, respectively, flat on $PR@3$. It shows that the MetricSage model can still fit the time-series metrics features well on top of the dynamic changes in topology, especially in $PR@1$, the same as the results in RQ1.

To further analyze the contribution of each component of the optional dynamic extension, we first explore the effect of the multi-instance pooling layer on root cause localization accuracy. After eliminating the multi-instance pooling layer, MicroIRC-TopoChange without multi-instance pooling decreases 22.2%, 9.7%, and 5.6% on $PR@1$, $PR@3$, and $PR@5$, respectively, compared to MicroIRC-TopoChange. By analyzing the reason, as shown in Fig. 8, it can be found that after subdividing the dynamically changing metrics features into Absent, Initialization, Healthy, and Anomalous, the raw metrics features show sparsity because the number of co-existing instances of the same service in the same period of time is smaller compared to the number of all

Table 7
Accuracy results for dynamic changes in topology.

Methods	PR@1	PR@3	PR@5	AVG@5	Acc	p-value
MicroIRC-TopoChange	**0.847**	**0.889**	**0.972**	**0.900**	**0.893**	–
MicroIRC-TopoChange w/o MetricSage	0.463	0.880	0.889	0.783	0.818	1.09e−1
MicroIRC-TopoChange w/o multi-instance pooling	0.625	0.792	0.917	0.722	0.830	2.00e−2
MicroIRC-TopoChange w/o mask	0.736	0.775	0.926	0.804	0.858	8.49e−3
MicroIRC-TopoChange w/o multi-instance pooling & mask	0.709	0.820	0.903	0.806	0.865	1.15e−2

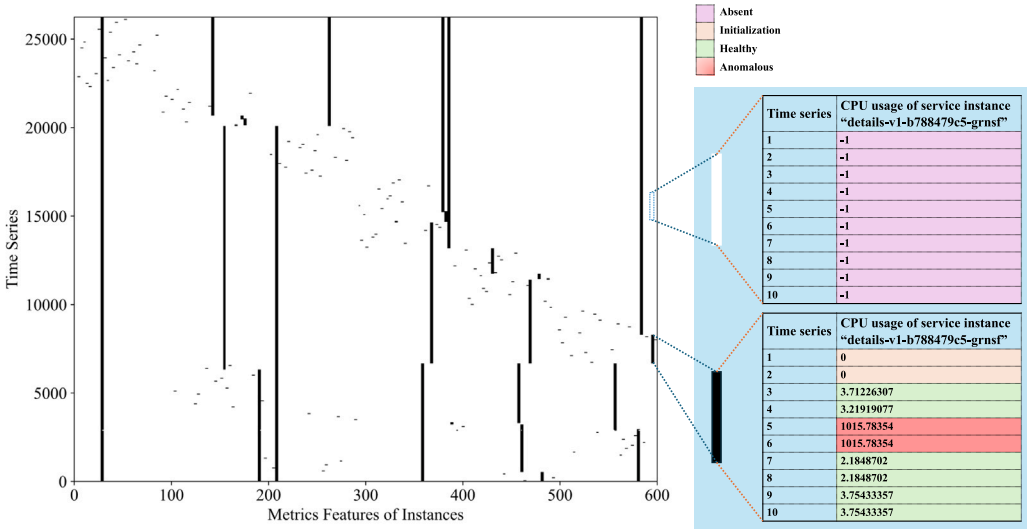


Fig. 8. Sparsity of time-series metrics features of instances.

instances that have existed in history. Meanwhile, most instances have a short period of existence. Using multi-instance pooling for multiple instances of the same service in the same time window can effectively transform sparse features into useful, dense features, while the smaller feature dimension improves the training efficiency.

Second, we find that masking the Absent type metrics features (to exclude the effect of Absent type features on the model parameters) has a boosting effect on the MetricSage model. MicroIRC-TopoChange without mask decreases 11.1%, 11.4%, and 4.7% on $PR@1$, $PR@3$, and $PR@5$, respectively, compared to MicroIRC-TopoChange. It shows that although the corresponding −1 value of Absent type features is usually dropped after multi-instance pooling with maximum operation, it still adversely affects the updating of model parameters during the calculation of gradient, and it is necessary to eliminate this adverse effect by masking.

Finally, we consider the case without the dynamic extension of the MetricSage model (MicroIRC-TopoChange without multi-instance pooling & mask, i.e., it remains the same as when the topology is stable). MicroIRC-TopoChange without pooling & mask reduces $PR@1$, $PR@3$, and $PR@5$ by 13.9%, 6.9%, and 7.0%, respectively, as compared to MicroIRC-TopoChange.

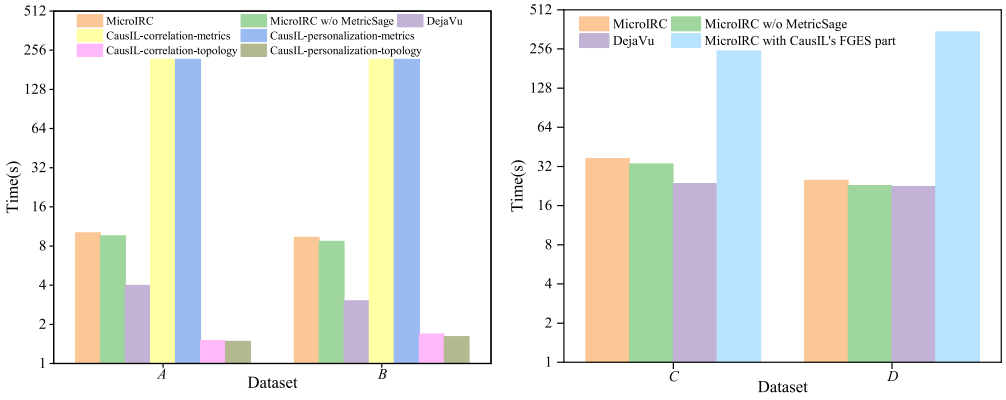
The p-value of the t-test is used to evaluate whether the observed improvements offered by the extension of dynamic topological changes occur by chance. Despite the MicroIRC-TopoChange without MetricSage, all p-values are less than 0.05, demonstrating that the extension is highly likely to improve root cause localization accuracy in the case of dynamic changes in topology, which are statistically significant. Although not statistically significant compared to the MicroIRC-TopoChange without MetricSage on all accuracy evaluation metrics, the use of the MetricSage model with the dynamic extension is able to produce a huge improvement on the $PR@1$ metrics.

We find that in the ablation experiments of each module of the extension of dynamic changes in topology, such as removing the multi-instance pooling layer, it leads to lower root cause localization accuracy compared to directly using MicroIRC for inference on models with static topology. Similar results are observed in $PR@3$ when the mask module or both the multi-instance pooling layer and mask module are removed. It shows that on dynamic changes in topology, if such dynamic cases are not taken as a key consideration, historical failure data not only does not help but may even have side effects, which we believe may all have a large impact on supervised methods. For example, methods like DejaVu, which rely on stable topology structures for model training and root cause localization, are entirely unsuitable for scenarios where the topology undergoes changes. This is because the number of nodes in the graph network corresponds one-to-one with the topology structure. From this perspective, similar modeling approaches are entirely different from MicroIRC.

5.9. Efficiency (RQ5)

We evaluate the efficiency of root cause localization across datasets A, B, C, and D, comparing MicroIRC with other methods, as depicted in Fig. 9. MicroIRC demonstrates a significant efficiency advantage in identifying root causes compared to CausIL using metrics type of graph, which relies on causal inference. A notable exception occurs in scenarios with very small microservice topologies, where CausIL may exhibit the highest efficiency. However, the accuracy achieved by CausIL in such cases is substantially lower than that of other methods. The efficiency of MicroIRC and DejaVu methods is similar in datasets A and B; as the size of the microservice topology grows in datasets C and D, the difference increases slightly. The reason is that the two approaches cost similar on the inference part, and the main efficiency bottleneck is in the personalized random walk algorithm running on the HWT graph, which is less efficient on the larger-scale topology. Despite this, given MicroIRC's pronounced strengths in addressing new types of failure and dynamic changes, we consider the slight efficiency loss to be a justified trade-off.

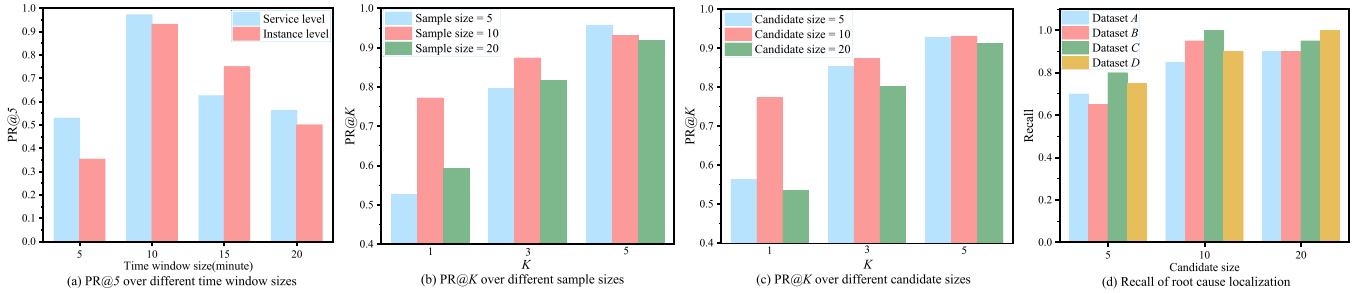
As shown in Table 8, we further analyze the efficiency of MicroIRC's components and find that the personalized random walk indeed accounts for more than 90% of the overall time, with a very subtle



(a) Localization efficiency for each failure on dataset *A* and *B*

(b) Localization efficiency for each failure on dataset *C* and *D*

Fig. 9. Localization efficiency for each failure.



(a) PR@5 over different time window sizes

(b) PR@*K* over different sample sizes

(c) PR@*K* over different candidate sizes

(d) Recall of root cause localization

Fig. 10. Impact of different parameters.

Table 8
Execution time of different components in the root cause localization process.

Datasets	Random walk stage (ms)	Subdivision stage (ms)
A	9614 (95.00%)	506 (5.00%)
B	8709 (93.15%)	640 (6.85%)
C	33,436 (91.20%)	3226 (8.80%)
D	22,806 (90.88%)	2289 (9.12%)

Table 9
Influence of the sampling size on model training.

Number of sampling	Average batch time (s)	Validation F1	Loss
5	0.563	0.728	0.711
10	0.604	0.829	0.513
20	0.913	0.896	0.39

decrease in that percentage as the service topology increases. This phenomenon suggests that while the personalized random walk is always a performance bottleneck for the method, the increase in subdivision time may be more pronounced. We suspect that this is due to the fact that the increase in service topology leads to a rise in metrics dimensions, which in turn increases the size of the model parameters. We will continue to optimize the method's efficiency for even larger service sizes in our future work.

5.10. Parameter analysis (RQ6)

5.10.1. Selection of the time window

When a failure is detected within a specific time interval, the root cause localization process selects a sliding time window near the anomaly time interval to collect metrics data during the start-up phase. The time window size directly impacts the effectiveness of data acquisition for root cause localization. If the time window is too large or takes a significant amount of time to collect, it may include normal metrics and noisy data, thus diminishing the accuracy. Conversely, an overly small time window may not reveal abnormal metrics data. Selecting a reasonable time window can effectively improve the accuracy of metrics data utilization, as shown in Fig. 10(a). The $PR@K$ metric shows a rising trend, followed by a decline as the sliding window increases, with the peak performance occurring at a time window of 10 min.

5.10.2. Impact of the sampling numbers on feature aggregation

The number of sampling used in each convolution process represents the number of neighboring nodes gathered by a single node. It is an important parameter that affects the performance of graph neural network model training and classification accuracy. A larger number of sampling can lead to higher accuracy but lower model training efficiency, and vice versa. Table 9 illustrates the impact of different sampling numbers (e.g., 5, 10, and 20) on model training time and final results, measured by the F1 score (defined as the summed average of precision and recall rates). A larger sampling number generally results in better classification of time-series metrics features and a more accurate classification effect. However, the influence of the sampling number on the final accuracy is not significant, as shown in Fig. 10(b). Because the input of the model consists of time series metrics features within the anomalous time window and the root cause candidate set. The feature weighting mechanism of the graph neural network on the candidate set involves weight multiplication. Therefore, a small decrease in the accuracy of fine classification hardly affects the ranking of weight sizes after multiplication. Considering both training time and final accuracy, a sampling frequency of five seconds, an anomalous time window of 10 min, and a sampling number of 10 are recommended for the datasets.

5.10.3. Selection of the candidate size in the random walk

In MicroIRC, the personalized random walk algorithm is used to generate a candidate set of root causes, which is then combined with

time-series metrics features from the anomaly time interval to be tested. This joint input is used to weigh the root cause features using a graph convolutional neural network, ultimately determining the final root cause. The size of the root cause candidate set can somewhat impact the final experimental results. It has been hypothesized that a larger candidate set will lead to slightly higher accuracy in root cause localization, and vice versa. However, as depicted in Fig. 10(c), the candidate set size has minimal effect on the accuracy of the final results. Nevertheless, it is important to note that an excessively small candidate set can significantly decrease the recall rate. The accuracy shows fluctuations within a 5% range as the candidate set size increases. Additionally, as illustrated in Fig. 10(d), the recall rate is notably lower when the candidate set is five compared to when it is 10 or 20.

6. Discussion

6.1. Guidelines for practitioners

These guidelines aim to equip software engineers and SREs with hands-on guidance on leveraging MicroIRC for efficient failure localization and resolution in microservice systems.

Environment construction and data preprocessing: After setting up the Service Mesh (e.g., Istio) and monitoring tools (e.g., Prometheus), it is vital to collect metrics data. Ensuring the quality of this data and performing necessary preprocessing steps, like metrics normalization, are essential for the accuracy of subsequent analyses.

Hyperparameter settings and model training: Identifying relevant hyperparameters is crucial for effective root-cause analysis. Practitioners should determine hyperparameters, including anomaly thresholds, sample sizes, time window sizes, and candidate sizes. This process should account for the fluctuation degree, fluctuation period, and the sampling frequency of metrics. Experimenting with various parameter settings and adjusting based on domain expertise and experimental outcomes is advisable. It is also important to train the model using a comprehensive set of historical data that represents various failure types and accounts for both stable and dynamic topologies.

Considerations and common issues: During data processing, maintaining the chronological order of metrics data is paramount to preserve the integrity of time-series features. Furthermore, when training the model, ensure that failure labels and collected metrics data correspond to the same time window to prevent discrepancies.

Appendix: We encourage further exploration of our publicly released code with datasets and materials of open-source tools (see Section 5.1).

6.2. Limitation

There are three main limitations to our proposed approach. The first limitation concerns the efficiency of MicroIRC, which uses the global metrics data dimension of the microservice system as the feature dimension and models the time-series root cause features through a graph neural network. When dealing with a large microservice system with high temporal sampling frequency, the training efficiency decreases as each temporal sample is trained separately in a graph neural network. This limitation highlights the need for further optimization to enhance the training efficiency in large-scale microservice systems. Moreover, supervised model-based methods inevitably require extra time for model training compared to unsupervised methods (Chakraborty et al., 2023; Wu et al., 2020b). In the future, we aim to design an encoder that effectively retains the metrics features while enhancing method efficiency amidst growing metrics data volumes.

Meanwhile, the accuracy of localization can be adversely affected by dynamic topology changes within the same anomalous time window, such as the frequent destruction and reconstruction of microservice instances. Supervised methods based on metrics face challenges when metrics dimensions alter due to topology adjustments. Entirely supervised model-based localization methods, like DejaVu, may fail in the face of topology changes, necessitating model retraining to accurately localize root causes once more. Our approach mitigates accuracy impacts by blending supervised models with unsupervised random walk techniques. Despite optimizations for handling dynamic topology changes, as demonstrated with dataset *E*, significant topology alterations still require retraining the MetricSage model. Enhancing the application of the supervised modeling component to new topologies remains a focus for future improvements to MicroIRC.

The third limitation is that the current MetricSage model is designed for scenarios with a single anomalous root cause and struggles with situations involving multiple simultaneous anomalous root causes. Future work will explore scenarios with multiple concurrent root causes, developing datasets to capture these complex failure modes for deeper analysis.

6.3. Threats to validity

The **internal** threat to validity concerns repeatability. To mitigate issues arising from bugs or errors during method implementation, we selected the most widely used microservice management and container orchestration framework (see Section 5.1). We also used the Service Mesh integrated with observability tools as a plug-in[14] to monitor systems and collect metrics data. This approach not only ensures minimal intrusiveness into the system but also facilitates the adaptation of our method to both new system constructions and modifications of existing systems without altering the source code. To bolster the reliability of our method, comprehensive testing was performed, and the procedures were repeated multiple times.

The **external** threat to validity primarily concerns the generalizability of our findings. Although our experiments with MicroIRC span five datasets, featuring a range of metrics and dimensions that demonstrate its capacity for generalization to a degree, we acknowledge that these datasets may not represent all systems or encapsulate every type of system failure. Nevertheless, based on current results, it is reasonable to believe MicroIRC has sufficient adaptability and can work well in microservice architecture-based systems. To investigate the applicability of MicroIRC to new failure types, we categorized failure types into recurring and new root causes in proportion, which yielded improved results. To validate MicroIRC's adaptability to dynamic topology changes, we incorporated optional extensions into the MetricSage model, achieving high accuracy on dataset *E*. Due to the smaller service granularity and the higher degree of modularity in the microservice architecture, almost all parts of a microservice system can run as microservices (e.g., gateways, middleware, and load balancers), allowing microservices to be deployed and scaled with the help of the containerization technology. At the same time, services and instances (containers) in a microservice system have access to a unified monitoring system that more accurately captures call relationships and operational metrics, allowing MicroIRC to locate failures down to the instance level. However, in other software architectures, such as multi-service systems, where the degree of service functional coupling is high, it is difficult to analyze service dependencies, and the concept of instances does not exist. MicroIRC might not work in this scenario.

In the future, we still need to explore the applicability of MicroIRC for more systems with additional failure types and metrics data types, not limited to microservice systems, ensuring a more comprehensive evaluation and better generalizability.

The **construct** threat to validity primarily arises from the selection of hyperparameters and evaluation metrics. The complexity and specificity of microservice systems necessitate that hyperparameter tuning incorporates system-specific domain knowledge. To mitigate this

[14] https://istio.io/latest/docs/concepts/observability/#metrics.

threat, it is recommended to identify the metrics fluctuation cycle and the degree of fluctuation of a particular microservice system together with SREs when practicing the actual methodology. This issue also pertains to the comparison methods, particularly regarding the adjustment of the anomalous time window size. During our experiments, we referred to several datasets and existing methods (Li et al., 2022b; Chakraborty et al., 2023), taking into account the timeliness of failure occurrence in the practice, and finally standardized on 15 min for dataset E, which features dynamic topology changes, and 10 min for the remaining datasets. This duration effectively captures the fluctuation period associated with failure metrics. For all other parameters, we adhered to specifications provided in the respective methodologies' publications or open-source code. Experimental results were determined by averaging the results obtained under optimal parameters, supported by statistical testing. Furthermore, we used widely-adopted evaluation metrics (see Section 5.4).

7. Conclusion and future work

The MicroIRC method proposed in this paper aims to address fine-grained root cause localization in microservices. By combining a personalized random walk approach with MetricSage, our approach achieves accurate results. The process involves gathering metrics data, correlating it with topology, extracting time-series features, training graph convolutional neural network models, and reducing root cause localization granularity to the instance level. Experiments conducted on five real-world datasets demonstrate that MicroIRC can accurately locate the root cause of microservices both at the instance and service levels. It outperforms state-of-the-art methods and exhibits robustness when handling dynamic topological changes and new failure types.

In the future, we will improve our research in the following aspects:

- We will consider scenarios where multiple instances of multiple types of anomalies occur simultaneously to locate root causes.
- After locating the root cause of a failure, predicting the degree of a failure and its impact on the entire system can help developers adjust the system and prevent future failures. We plan to leverage prediction techniques and log analysis to address the issue.

CRediT authorship contribution statement

Yuhan Zhu: Writing – original draft, Software, Methodology, Data curation, Conceptualization. **Jian Wang:** Writing – review & editing, Supervision, Methodology. **Bing Li:** Supervision, Methodology. **Yuqi Zhao:** Visualization, Validation. **Zekun Zhang:** Writing – review & editing, Validation. **Yiming Xiong:** Software, Methodology. **Shiping Chen:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data and source code can be downloaded from https://github.com/WHU-AISE/MicroIRC.git.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62032016), the Key Research and Development Program of Hubei Province (No. 2021BAA031), and the Foundation of Yunnan Key Lab of Service Computing (No. YNSC23102).

References

Akoglu, L., Tong, H., Koutra, D., 2015. Graph based anomaly detection and description: a survey. Data Min. Knowl. Discov. 29 (3), 626–688.

Aubet, F.-X., Pahl, M.-O., Liebald, S., Norouzian, M.R., 2018. Graph-based anomaly detection for iot microservices. Measurements 120 (140), 160.

Baarzi, A.F., Kesidis, G., 2021. SHOWAR: Right-sizing and efficient scheduling of microservices. In: SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021. ACM, pp. 427–441.

Brandón, Á., Solé, M., Huélamo, A., Solans, D., Pérez, M.S., Muntés-Mulero, V., 2020. Graph-based root cause analysis for service-oriented and microservice architectures. J. Syst. Softw. 159.

Chakraborty, S., Garg, S., Agarwal, S., Chauhan, A., Saini, S.K., 2023. CausIL: Causal graph for instance level microservice data. In: Proceedings of the ACM Web Conference 2023. ACM, pp. 2905–2915.

Chen, J., He, X., Lin, Q., Xu, Y., Zhang, H., Hao, D., Gao, F., Xu, Z., Dang, Y., Zhang, D., 2019a. An empirical investigation of incident triage for online service systems. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice. IEEE / ACM, pp. 111–120.

Chen, P., Qi, Y., Hou, D., 2019b. CauseInfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment. IEEE Trans. Serv. Comput. 214–230.

Chen, R., Ren, J., Wang, L., Pu, Y., Yang, K., Wu, W., 2022. MicroEGRCL: An edge-attention-based graph neural network approach for root cause localization in microservice systems. In: Service-Oriented Computing - 20th International Conference. Springer, pp. 264–272.

Chen, Y., Xu, D., Chen, N., Wu, X., 2023. FRL-MFPG: Propagation-aware fault root cause location for microservice intelligent operation and maintenance. Inf. Softw. Technol. 153, 107083.

Cheng, K., Zhang, S., Tu, C., Shi, X., Yin, Z., Lu, S., Liang, Y., Gu, Q., 2023. ProScale: Proactive autoscaling for microservice with time-varying workload at the edge. IEEE Trans. Parallel Distrib. Syst. 34 (4), 1294–1312.

Du, M., Li, F., Zheng, G., Srikumar, V., 2017a. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1285–1298.

Du, M., Li, F., Zheng, G., Srikumar, V., 2017b. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, pp. 1285–1298.

Du, Q., Xie, T., He, Y., 2018. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In: Algorithms and Architectures for Parallel Processing - 18th International Conference. Vol. 11337, Springer, pp. 560–572.

Fu, K., Zhang, W., Chen, Q., Zeng, D., Guo, M., 2022. Adaptive resource efficient microservice deployment in cloud-edge continuum. IEEE Trans. Parallel Distrib. Syst. 33 (8), 1825–1840.

Gan, Y., Liang, M., Dev, S., Lo, D., Delimitrou, C., 2021. Sage: practical and scalable ML-driven performance debugging in microservices. In: ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, pp. 135–151.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011. In: JMLR Proceedings, Vol. 15, JMLR.org, pp. 315–323.

Hamilton, W.L., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems. pp. 1024–1034.

He, Z., Chen, P., Luo, Y., Yan, Q., Chen, H., Yu, G., Li, F., 2022. Graph based incident extraction and diagnosis in large-scale online systems. In: 37th IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 48:1–48:13.

He, S., Lin, Q., Lou, J., Zhang, H., Lyu, M.R., Zhang, D., 2018a. Identifying impactful service system problems via log analysis. In: Proceedings of the 2018 Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 60–70.

He, S., Lin, Q., Lou, J.-G., Zhang, H., Lyu, M.R., Zhang, D., 2018b. 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. Association for Computing Machinery, pp. 60–70.

Huo, Y., Su, Y., Lee, C., Lyu, M.R., 2023. SemParser: A semantic parser for log analytics. In: Proceedings of the 45th International Conference on Software Engineering. IEEE Press, pp. 881–893.

Kandula, S., Mahajan, R., Verkaik, P., Agarwal, S., Padhye, J., Bahl, P., 2009. Detailed diagnosis in enterprise networks. In: Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM, pp. 243–254.

Kim, M., Sumbaly, R., Shah, S., 2013. Root cause detection in a service-oriented architecture. In: ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems. ACM, pp. 93–104.

Li, Z., Chen, J., Jiao, R., Zhao, N., Wang, Z., Zhang, S., Wu, Y., Jiang, L., Yan, L., Wang, Z., Chen, Z., Zhang, W., Nie, X., Sui, K., Pei, D., 2021. Practical root cause localization for microservice systems via trace analysis. In: 2021 IEEE/ACM 29th International Symposium on Quality of Service. IWQOS, pp. 1–10.

Li, Z., Tu, Y., Ma, Z., 2022a. Root cause analysis of anomalies based on graph convolutional neural network. Int. J. Softw. Eng. Knowl. Eng. 1155–1177.

Li, Z., Zhao, N., Li, M., Lu, X., Wang, L., Chang, D., Nie, X., Cao, L., Zhang, W., Sui, K., Wang, Y., Du, X., Duan, G., Pei, D., 2022b. 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. ACM, pp. 996–1008.

Lin, J., Chen, P., Zheng, Z., 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In: Service-Oriented Computing - 16th International Conference, ICSOC. Vol. 11236, Springer, pp. 3–20.

Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., Chen, X., 2016. Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering Companion. Association for Computing Machinery, pp. 102–111.

Liu, P., Chen, Y., Nie, X., Zhu, J., Zhang, S., Sui, K., Zhang, M., Pei, D., 2019. FluxRank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation. In: 30th IEEE International Symposium on Software Reliability Engineering. IEEE, pp. 35–46.

Liu, D., He, C., Peng, X., Lin, F., Zhang, C., Gong, S., Li, Z., Ou, J., Wu, Z., 2021. MicroHECL: High-efficient root cause localization in large-scale microservice systems. In: 43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice. IEEE, pp. 338–347.

Ma, M., Lin, W., Pan, D., Wang, P., 2019. MS-Rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications. In: 2019 IEEE International Conference on Web Services. IEEE, pp. 60–67.

Ma, M., Lin, W., Pan, D., Wang, P., 2022. Self-adaptive root cause diagnosis for large-scale microservice architecture. IEEE Trans. Serv. Comput. 1399–1410.

Ma, M., Xu, J., Wang, Y., Chen, P., Zhang, Z., Wang, P., 2020. AutoMAP: Diagnose your microservice-based web applications automatically. In: WWW '20: The Web Conference 2020. ACM / IW3C2, pp. 246–258.

Mariani, L., Pezzè, M., Riganelli, O., Xin, R., 2020. Predicting failures in multi-tier distributed systems. J. Syst. Softw..

Marwede, N., Rohr, M., van Hoorn, A., Hasselbring, W., 2009. Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation. In: 13th European Conference on Software Maintenance and Reengineering. IEEE Computer Society, pp. 47–58.

Meng, C., Song, S., Tong, H., Pan, M., Yu, Y., 2023. DeepScaler: Holistic autoscaling for microservices based on spatiotemporal GNN with adaptive graph learning. In: 38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023. IEEE, pp. 53–65.

Meng, Y., Zhang, S., Sun, Y., Zhang, R., Hu, Z., Zhang, Y., Jia, C., Wang, Z., Pei, D., 2020. Localizing failure root causes in a microservice through causality inference. In: 28th IEEE/ACM International Symposium on Quality of Service. IEEE, pp. 1–10.

Pan, Y., Ma, M., Jiang, X., Wang, P., 2023. DyCause: Crowdsourcing to diagnose microservice kernel failure. IEEE Trans. Dependable Secure Comput. 1–16.

Qiu, H., Banerjee, S.S., Jha, S., Kalbarczyk, Z.T., Iyer, R.K., 2020. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, pp. 805–825.

Sedghpour, M.R.S., Townend, P., 2022. Service mesh and eBPF-powered microservices: A survey and future directions. In: IEEE International Conference on Service-Oriented System Engineering, SOSE 2022, Newark, CA, USA, August 15-18, 2022. IEEE, pp. 176–184.

Soldani, J., Brogi, A., 2023. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. ACM Comput. Surv. 55 (3), 59:1–59:39.

Soldani, J., Forti, S., Brogi, A., 2022. Failure root cause analysis for microservices, explained. In: Distributed Applications and Interoperable Systems. Springer International Publishing, pp. 74–91.

Sun, C.-A., Zeng, T., Zuo, W., Liu, H., 2023. A trace-log-clusterings-based fault localization approach to microservice systems. In: 2023 IEEE International Conference on Web Services. ICWS, pp. 7–13.

Tong, G., Meng, C., Song, S., Pan, M., Yu, Y., 2023. GMA: Graph multi-agent microservice autoscaling algorithm in edge-cloud environment. In: IEEE International Conference on Web Services, ICWS 2023, Chicago, IL, USA, July 2-8, 2023. IEEE, pp. 393–404.

Usman, M., Ferlin, S., Brunstrom, A., Taheri, J., 2022. A survey on observability of distributed edge & container-based microservices. IEEE Access 10, 86904–86919.

Wang, L., Jiang, Y.X., Wang, Z., Huo, Q.E., Dai, J., Xie, S.L., Li, R., Feng, M.T., Xu, Y.S., Jiang, Z.P., The operation and maintenance governance of microservices architecture systems: A systematic literature review. J. Softw.: Evol. Process. e2433.

Wang, H., Nguyen, P., Li, J., Köprü, S., Zhang, G., Katariya, S., Ben-Romdhane, S., 2019. GRANO: Interactive graph-based root cause analysis for cloud-native distributed data platform. Proc. VLDB Endow. 12 (12), 1942–1945.

Wang, P., Xu, J., Ma, M., Lin, W., Pan, D., Wang, Y., Chen, P., 2018. CloudRanger: Root cause identification for cloud native systems. In: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society, pp. 492–502.

Weng, J., Wang, J.H., Yang, J., Yang, Y., 2018. Root cause analysis of anomalies of multitier services in public clouds. IEEE/ACM Trans. Netw. 26 (4), 1646–1659.

Wu, L., Bogatinovski, J., Nedelkoski, S., Tordsson, J., Kao, O., 2020a. Performance diagnosis in cloud microservices using deep learning. In: Service-Oriented Computing - ICSOC 2020 Workshops. Vol. 12632, Springer, pp. 85–96.

Wu, L., Tordsson, J., Bogatinovski, J., Elmroth, E., Kao, O., 2021a. MicroDiag: Fine-grained performance diagnosis for microservice systems. In: 2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence). pp. 31–36.

Wu, L., Tordsson, J., Elmroth, E., Kao, O., 2020b. MicroRCA: Root cause localization of performance issues in microservices. In: NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium. IEEE, pp. 1–9.

Wu, C., Zhao, N., Wang, L., Yang, X., Li, S., Zhang, M., Jin, X., Wen, X., Nie, X., Zhang, W., Sui, K., Pei, D., 2021b. Identifying root-cause metrics for incident diagnosis in online service systems. In: 32nd IEEE International Symposium on Software Reliability Engineering. IEEE, pp. 91–102.

Xie, S., Wang, J., Li, B., Zhang, Z., Li, D., Hung, P.C.K., 2023. PBScaler: A bottleneck-aware autoscaling framework for microservice-based applications. CoRR abs/2303.14620.

Yu, G., Chen, P., Chen, H., Guan, Z., Huang, Z., Jing, L., Weng, T., Sun, X., Li, X., 2021. MicroRank: End-to-End latency issue localization with extended spectrum analysis in microservice environments. In: WWW '21: The Web Conference 2021. ACM / IW3C2, pp. 3087–3098.

Yu, Y., Yang, J., Guo, C., Zheng, H., He, J., 2019. Joint optimization of service request routing and instance placement in the microservice system. J. Netw. Comput. Appl. 147.

Yuan, Y., Shi, W., Liang, B., Qin, B., 2019. An approach to cloud execution failure diagnosis based on exception logs in OpenStack. In: 12th IEEE International Conference on Cloud Computing. IEEE, pp. 124–131.

Zhang, Z., Li, B., Wang, J., Liu, Y., 2021. AAMR: Automated anomalous microservice ranking in cloud-native environment. In: The 33rd International Conference on Software Engineering and Knowledge Engineering, SEKE 2021, KSIR Virtual Conference Center, USA, July 1 - July 10, 2021. KSI Research Inc., pp. 86–91.

Zhang, L., Morin, B., Baudry, B., Monperrus, M., 2022. Maximizing error injection realism for chaos engineering with system calls. IEEE Trans. Dependable Secur. Comput. 19 (4), 2695–2708.

Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: An efficient data clustering method for very large databases. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996. ACM Press, pp. 103–114.

Zhang, L., Zhao, J., Zhang, M., 2020. Root cause analysis of concurrent alarms based on random walk over anomaly propagation graph. In: IEEE International Conference on Networking, Sensing and Control. IEEE, pp. 1–6.

Zhao, Y., Li, B., Wang, J., Jiang, D., Li, D., 2022. Integrating deep reinforcement learning with pointer networks for service request scheduling in edge computing. Knowl.-Based Syst. 258, 109983.

Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W., Ding, D., 2021. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. IEEE Trans. Softw. Eng. 47 (2), 243–260.

Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., Xiang, Q., He, C., 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 683–694.

Zhou, N., Zhou, H., Hoppe, D., 2023. Containerization for high performance computing systems: Survey and prospects. IEEE Trans. Softw. Eng. 49 (4), 2722–2740.

Yuhan Zhu received his B.S. degree from the School of Computer Science, Wuhan University, China, in 2022. He is currently pursuing a master's degree at Wuhan University. His current research interests include services computing, artificial intelligence for IT operations (AIOps), and edge computing.

Jian Wang received his Ph.D. degree in Computer Science from Wuhan University, China, in 2008. He is currently an Associate Professor in the School of Computer Science, Wuhan University, China. His current research interests include services computing and software engineering. He is now a member of the IEEE, a senior member of the China Computer Federation (CCF), and a member of the CCF Technical Committee on Services Computing (TCSC).

Bing Li received the Ph.D. degree from Huazhong University of Science and Technology, Wuhan, China, in 2003. He is currently a Professor at the School of Computer Science, Wuhan University. His main research areas are services computing, software engineering, artificial intelligence, and cloud computing. He is now a member of the IEEE and a distinguished member of the China Computer Federation (CCF).

Yuqi Zhao is currently a lecturer at the School of Computer Science, Central China Normal University. He graduated with a Ph.D. degree from the School of Computer Science, Wuhan University. He received both his Bachelor and Master degrees from Wuhan University, studying at the International School of Software and the School of Computer Science respectively. His current research interests include artificial intelligence, software engineering, and edge computing.

Zekun Zhang received his M.S. degree from the school of computer science, Wuhan University, China. He is currently working toward the Ph.D. degree at the school of computer science, Wuhan University. His research interests include cloud computing and fault diagnosis.

Yiming Xiong received the B.E. and the Master from Wuhan University, Wuhan, China, in 2018 and 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science. His current research interests include artificial intelligence and knowledge graph.

Shiping Chen is a principal research scientist at CSIRO Data61. He also holds an adjunct associate professor title with the University of Sydney through teaching and supervising PhD/Master students. His current research interests include secure data storage & sharing and secure multi-party collaboration. He is a senior member of the IEEE and a fellow of Institution of Engineering and Technology (IET).