Hypergraph Neural Network-based Multi-Granular Root Cause Localization for Microservice Systems

Yaxiao Li*, Lu Wang*¶, Chenxi Zhang*¶, Qingshan Li*, Siming Rong*, Baiyang Wen*, Xuyang Li*, Kun Ma*, Quanwei Du*, KeYang Li*, Lingfeng Pan*, Xinyue Li†, Mingxuan Hui*
*Xidian University, Xi'an, China, †Peking University, Beijing, China
{yx_li, rsm, typemoon, whisper, mk2022, lfpan, keyangli, hmx}@stu.xidian.edu.cn
{wanglu, zhangchenxi}@xidian.edu.cn, qshli@mail.xidian.edu.cn
du_quanwei@163.com, xinyueli@stu.pku.edu.cn

***Abstract*—Modern enterprises are increasingly adopting microservice architectures to enhance system flexibility and scalability. However, in the face of ever-changing business requirements, the relationships between system components have become increasingly complex, resulting in significant challenges in maintaining system robustness. In recent years, multimodal data-driven approaches based on graph neural networks have emerged as a predominant solution for root cause localization in microservice systems. Our detailed analysis of architectural characteristics and existing research reveals two critical limitations. First, simple graph is insufficient to represent the one-to-many relationships inherent in microservice component interactions, such as deployment, subordinate, and dependency. Second, the current multimodal data-based method has difficulty in performing localization on faults occurring on hosts, services, and instances at the same time.**

To address these challenges, we propose HyperRCA, a novel multi-granular root cause analysis approach based on hypergraph neural networks. Our approach models system states during faults via a hypergraph with instances as graph nodes, explicitly capturing heterogeneous relationships through three innovative hyperedge designs: deployment hyperedges for infrastructure relationships, subordinate hyperedges for service hierarchies, and dependency hyperedges for inter-component interactions. We used hypergraph neural networks and multilayer perceptrons to train a root cause localization model based on hyperedge features to achieve multi-granularity root cause localization. Experimental evaluations demonstrate significant performance improvements over state-of-the-art approaches. HyperRCA achieves a maximum HR@5 improvement of 112.62% on single-granularity datasets and 466.43% in multi-granularity scenarios.

***Index Terms*—Microservice System, Root Cause Localization, Hypergraph Neural Network, Multi-granular**

I. Introduction

Software systems are widely used across various industries, serving as the backbone for digital transformation. In response to the increasing demand for agility and efficiency, more than 84% of companies are using or evaluating cloud-native technologies to build their software systems [1]. Among these technologies, microservice architecture is an important part of cloud native technology. Microservice architecture is adopted in development due to its flexibility, scalability, and decentralization. However, these characteristics lead to complex features of system faults propagating across granularities. According to Gartner's 2023 Cloud Service Reliability Report [2], if the mission-critical service of AWS us-east-1 is interrupted for 24 hours, it is expected to cause a direct revenue loss of US$3.4 billion. If the mission-critical components of the service are interrupted for 48 hours, it is expected to cause a direct revenue loss of US$7.8 billion. Therefore, an effective method for root cause localization is urgently required to ensure reliable operation of microservice systems.

The microservice system is deployed on a distributed cluster and comprises numerous services, each with multiple instances responsible for specific functionalities. Hosts, services, and instances are the fundamental elements of a microservice system. They are collectively referred to as system components. Faults occurring within these system components are referred to as resource-granular faults. To effectively model the complex relationships among components and achieve precise root cause localization, it is essential to fully exploit the system's multimodal data. Multimodal data includes log, metric, and trace. Existing root cause localization methods use multimodal data for root cause analysis through feature fusion, result fusion, model fusion, etc [3]. These three types of data together constitute a comprehensive view of the system status.

As the scale of the system business expands, the dependencies between its components become more complex and diverse. There are diverse relationships between components, such as deployment relationships, subordinate relationships, and dependency relationships. If it is limited to the local dependency analysis between homogeneous entities, it will be difficult to characterize the heterogeneous interaction patterns between multi-dimensional entities such as hosts, services, and instances, which will lead to significant deviations in the localization of fault root causes.

In recent years, many methods based on multimodal data have been proposed because they have significantly outperformed those based on unimodal data. These methods usually use graph-based methods to model the structure and state of the system. However, simple graph means that each edge in the graph can only connect two nodes [4], which makes it difficult to model high-order relationships in microservice systems. Cross-granularity modeling explicitly captures diverse

¶Lu Wang and Chenxi Zhang are the corresponding authors.

component relationships and reveals the propagation pathways of system faults.

In addition, most of these methods only support root cause localization at a single granularity, such as instance granularity [5]–[13] and service granularity [14], [15]. The multi-granularity root cause localization method focuses on multi-granular faults such as hosts, services, and instances at the same time, which can reduce the occurrence of false negatives and improve the robustness of the system.

To address the above challenges, we propose HyperRCA, a multi-granular root cause localization method based on hypergraph neural networks and multimodal data. Firstly, HyperRCA uses a serialization method to unify multimodal data into sequence features, solving the problem of data heterogeneity. Secondly, HyperRCA builds a hypergraph for each fault time window. A variety of hyperedges are designed to capture diverse relationships such as deployment, subordination, and dependency. In addition, hyperedges can also be expanded according to actual conditions. Finally, HyperRCA uses a two-stage convolution method in the spatial domain to obtain hyperedge features that represent multi-granularity faults and integrate complex relationships. Based on the diversified hyperedge training root cause localization model, multi-granularity root cause localization is achieved.

To evaluate the performance of HyperRCA, we conduct experiments on two datasets, one multi-granular root cause dataset and one single-granularity root cause dataset. Experimental results show that HyperRCA improves the HR@5 of single-granularity and multi-granularity root cause localization by up to 112.62% and 466.43%.

In summary, the main contributions of this paper are:

- We first propose a method based on the combination of hypergraph neural network and multimodal data, using multimodal data as hypergraph node features to comprehensively characterize the system state.
- We design diverse hyperedges to explicitly encode cross-granularity relationships, using the resulting hyperedge features as input for root cause localization model.
- We have made the HyperRCA code [16] public to facilitate researchers to reproduce our work.

II. Background and Motivation

A. Background

1) Multimodal Data: In a microservice system, a single user request typically involves multiple services. For example, a ticket purchase may pass through several service, such as account authentication, ticket search, ticket display, order generation, and payment. Each function is provided by a specific service. However, due to the complex associations and dynamic interactions among services, it is difficult to locate the root cause [3], [17], [18]. To support observability, log, trace, and metric are the three core data types, while deployment relationships also provide key contextual information (Fig.1).

Log is a timestamped text record, either structured or unstructured, with optional metadata. Most programming languages have built-in logging capabilities or well-known, widely used logging libraries [19]. Logs include timestamp, verbosity level, instance, and raw information. There are four commonly used verbosity levels, INFO, WARN, DEBUG, and ERROR, indicating the severity of the log [7]. Engineers can understand the specific circumstances of the event by analyzing the log. Logs are usually collected and stored through ELK [20] and Splunk [21].

Metric is a measurement of a service captured at runtime [19], collected at fixed time intervals (such as once per second) for real-time monitoring and trend analysis. Metrics are collected and stored through tools such as Prometheus [22] and Grafana [23].

Trace is used to record call path in distributed systems. A trace, identified by a trace ID, forms a tree with services as nodes and call relationships as edges. Trace contains multiple fields, such as timestamp, caller, callee, duration, call type (http, rpc, db, telemetry, etc.), operation_name, trace ID, span ID, parent_span, etc. Engineers can identify failed calls and services by analyzing traces. Trace collection follows the OpenTelemetry [19] standard and is usually collected and stored through Jaeger [24] and Zipkin [25].

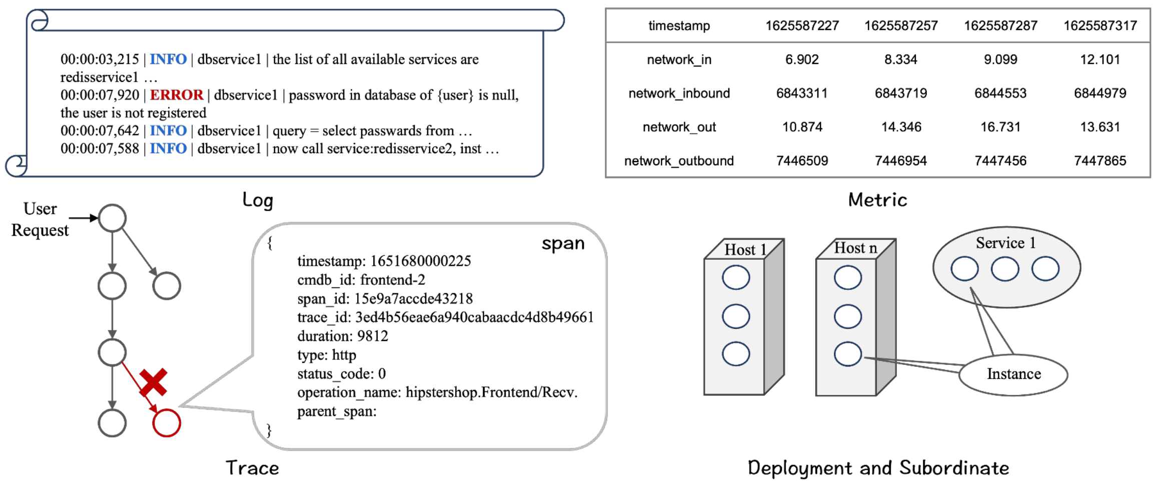
In addition to the three types of monitoring data, for microservice systems, deployment relationships and subordinate relationships also play an important role in root cause localization. Deployment relationships provide the physical relationship between instances and distributed hosts. Currently, many research works have considered deployment relationships [7] and achieved excellent performance.

2) Hypergraph: Hypergraph learning was first introduced in [26], which performs conduction learning and can be seen as a propagation process on a hypergraph structure. A hypergraph is a more general structure than a simple graph, which uses degree-free hyperedges to encode high-order data associations [4]. Each edge in a simple graph can only connect two vertices, which means that it can only express the relationship between two vertices. In the real world, there are a large number of one-to-many and many-to-many high-order relationships. In a hypergraph, each hyperedge can connect more than two vertices, making it more flexible, as shown in Fig.2.

The flexibility of hyperedge degrees gives hypergraphs powerful expressiveness, allowing them to accurately model a variety of group interactions that graphs lack. Hypergraphs are designed to handle complex high-order relational data. This inherent expressiveness of hypergraphs has led to their application in various fields, including recommender systems [27], computer vision [28], natural language processing [29], social network analysis [30], financial analysis [31], bioinformatics [32], and circuit design [33].

B. Motivation

In actual production environments, microservice systems are deployed on distributed clusters. WeChat's microservice system accommodates more than 3000 services running on over 20000 machines [34]. This results in complex cross-granularity relationships between the three granularities of



```
00:00:03,215 | INFO | dbservice1 | the list of all available services are
redisservice1 …
00:00:07,920 | ERROR | dbservice1 | password in database of {user} is null,
the user is not registered
00:00:07,642 | INFO | dbservice1 | query = select passwards from …
00:00:07,588 | INFO | dbservice1 | now call service:redisservice2, inst …
```

timestamp	1625587227	1625587257	1625587287	1625587317
network_in	6.902	8.334	9.099	12.101
network_inbound	6843311	6843719	6844553	6844979
network_out	10.874	14.346	16.731	13.631
network_outbound	7446509	7446954	7447456	7447865

```
{
  timestamp: 1651680000225
  cmdb_id: frontend-2
  span_id: 15e9a7accde43218
  trace_id: 3ed4b56eae6a940cabaacdc4d8b49661
  duration: 9812
  type: http
  status_code: 0
  operation_name: hipstershop.Frontend/Recv.
  parent_span:
}
```

Fig. 1: Examples of multimodal data and relationship

distributed hosts, services, and instances in the microservice system. As business systems grow and clusters become more complex, system functions and resource types increase, which in turn increases the probability of fault and the types of faults.

TABLE I: RCA granularity of multimodal data-based methods

Approach	Host	Service	Instance
TVdiag [15]			✓
DiagFusion [7]			✓
DeepHunt [8]			✓
ART [9]			✓
OCEAN [10]			✓
CHASE [11]			✓
MicroCBR [12]			✓
Nezha [5]			✓
MULAN [13]			✓
Eadro [14]		✓	
PDiagnose [15]		✓	
TrinityRCL [35]	✓	✓	✓
HyperRCA(Ours)	✓	✓	✓

In the past, most methods used single-modal data for root cause localization [36]–[38]. However, studies have shown that fault diagnosis using only single-modal data will produce false positives [7], [8], [39]. Using multimodal data is a solution. Multimodal data can monitor microservice systems from multiple perspectives and provide a comprehensive view of the system. In recent years, more than 10 methods based on multimodal data have emerged, and achieved excellent results.

After reviewing existing research work, as shown in the Table I, we found that the root cause localization granularity of current multimodal methods mostly stays at the instance level, which means that they are helpless for faults beyond the instance level. There are also a small number of methods that can locate faults at the service granularity. Fewer methods locate the root cause of faults at multiple granularities, which is exactly what engineers need.

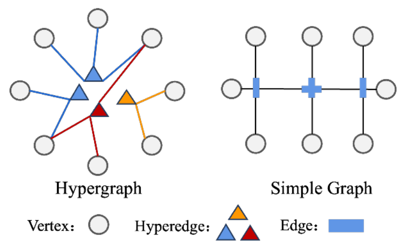


Fig. 2: Hypergraph and simple graph

As shown in Fig.2, simple graph can only model the relationship between two components, while there are higher-order relationships between components in a microservice system. The high-order correlations in a microservice system are mainly reflected in the following aspects, as shown in the Fig.3: 1) In a distributed cluster, multiple service instances are deployed on distributed hosts through pods, and there is a one-to-many relationship; 2) A single fault synchronously affects multiple replica instances through the service resource definition (one-to-many).

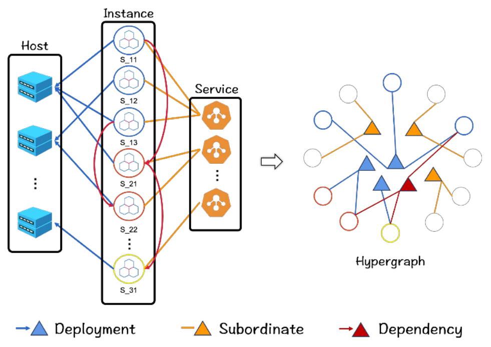


Fig. 3: One-to-many relationships in microservice systems

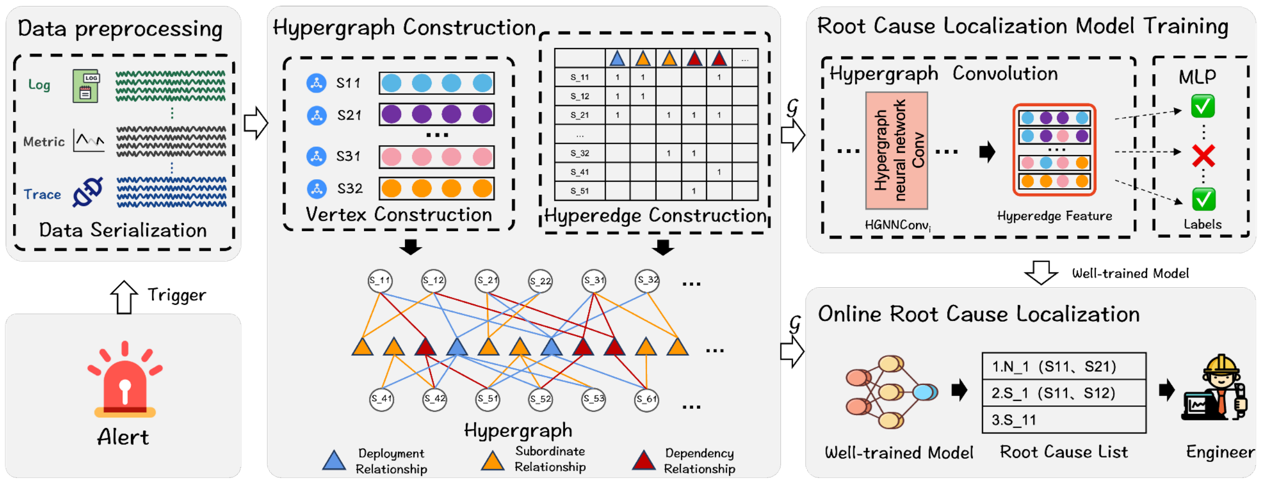


Fig. 4: Overview of HyperRCA

In the field of root cause localization of microservice systems, graph neural networks are widely used to model system and fault information [5], [7], [14]. However, traditional graph structures can form pairs (i.e., first-order relationships) between two topics, which is not powerful enough to handle high-order correlations, thus limiting the ability to model high-order correlations [4]. In addition, simple graphs are weak in modeling multimodal real data. Hypergraphs can explicitly model such multivariate asymmetric dependencies by allowing edges to connect any number of nodes, and their hyper-adjacency matrices and spectral domain features can effectively capture implicit fault propagation paths.

III. Approach

Fig.4 is an overview of HyperRCA. HyperRCA first serializes multimodal data, then builds a hypergraph within the fault time window, designs diverse hyperedges, and finally selects the hyperedge of the fault root. HyperRCA consists of four parts: Multimodal data preprocessing(§3.A), Hypergraph construction(§3.B), Root cause localization model training(§3.C), and Online root cause localization(§3.D).

A. Multimodal Data Preprocessing

Multimodal data is crucial for root cause localization in microservice systems, as single-modal data cannot reveal all types of faults [39]. Some faults appear only in a specific modality and would be missed if relying on others. By providing complementary perspectives, multimodal data reduces such false negatives. Yet, heterogeneity across modalities and structural complexity within each modality make effective fusion challenging. Recent studies address this by applying serialization methods, widely used in CV and NLP, to multimodal fault diagnosis in microservice systems [39], [40].

Serialization of log. Among the existing methods for preprocessing log data, the log parsing algorithm Drain is widely used. Inspired by the existing methods [40], HyperRCA uses Drain to extract log templates, count the frequency fluctuations of various templates in the fault window, and construct event weights. The pre-trained BERT model is used to generate the semantic embedding vector of the template, and the events in the weighted aggregation window are converted into a time series feature sequence. Finally, Transformer is used to capture the long-term dependencies, and global pooling is used to generate a compact representation of the log modality.

Serialization of metric. Metrics are in the form of serialized data, so only simple preprocessing is required. First, the metrics (CPU, memory, response latency) of each microservice instance and API are z-score standardized, and then the first-order difference is performed to eliminate baseline drift. Secondly, different indicators are regarded as independent channels, and the channel attention of SENet is used to enhance abnormally sensitive indicators. Finally, dilated causal convolution (DCC) is used to model the horizontal correlation and vertical dynamics between indicators.

Serialization of trace. Trace consists of the path of user requests, including inter-service dependency information and detailed information of service calls. HyperRCA extracts service call latency, request counts, and status codes from trace data. Indicators such as average latency per minute, total number of requests per minute, and frequency of various status codes per minute are calculated. Multi-granularities time series features are constructed in the same way as metric data. In addition, trace data contains structural information. HyperRCA uses frequent item mining technology to find service calls that frequently appear within the time window. Based on this, dependency hyperedges are constructed in §3.B.

B. Hypergraph Construction

To support multi-granular root cause localization, HyperRCA integrates multimodal data within the fault time window and builds a hypergraph to represent runtime components of a microservice system. Faults may propagate not only along service call paths but also across services on the same host

[35]. HyperRCA models this diversity using a hypergraph neural network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{H})$, where instances as nodes of the hypergraph. Because instances are the key to linking multi-granular resources. Instances connect downward to hosts (deployment), upward to services (subordinate), and laterally through frequent inter-instance calls (dependency). Their features are preprocessed in §3.A.

To capture these relationships, HyperRCA defines three types of hyperedges across hosts, services, and instances, corresponding to deployment, subordinate, and dependency relations. This hypergraph is encoded as a hypergraph matrix and serves as input to §3.C. The hyperedge is defined as:

Deployment hyperedge. The microservice system is deployed on multiple distributed hosts in a distributed manner, and multiple service instances are deployed on each distributed hosts. The purpose of the component deployment hyperedge is to model the physical deployment relationship between the service instance and the distributed hosts. For example, three instances are deployed on distributed host-A.

Subordinate hyperedge. The subordinate relationship between service and instance. The instance is the implementation of the service and handles the real business logic. For example, when a user accesses the IP of the Service, the Service selects the three created instances through the label, and the request will be evenly distributed to the three service instances.

Dependency hyperedge. Service call relationships are usually not simple point-to-point, but complex call chains such as one-to-many and many-to-many. In microservice systems, faults will propagate along the call chain. HyperRCA uses frequent item mining to mine frequently occurring instance call relationships from trace telemetry data within each fault time window, and selects a suitable number of frequent items as dependency hyperedges.

In addition, in order to meet the complexity and variability of microservice systems in actual use, HyperRCA supports scalable node types and hyperedge types. For example, according to actual conditions, engineers can dynamically associate instances with configuration centers and key management systems to capture systemic failures caused by configuration errors or key failures; model compatibility relationships such as service interface versions and serialization protocols to identify incompatibility issues caused by grayscale releases.

C. Root Cause Localization Model Training

After completing the hypergraph construction, we proposed a multi-granular root cause localization framework that integrates hypergraph neural networks and multi-layer perceptrons(MLP). The hyperedges in the hypergraph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{H})$ directly represent the root cause, and each hyperedge $e \in \mathcal{E}$ is assigned a fault label corresponding to its granularity. The spatial convolution method of HGNN+ [4] is used to capture the high-order dependencies between components, and the hyperedge representation is mapped to the category space through MLP to achieve localization.

Specifically, HyperRCA formalizes the system state in time window t as a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{H})$. The node set $\mathcal{V} = v_i$ contains entities such as service instances, and the hyperedge set $\mathcal{E} = e_j$ consists of three types of predefined relationships (§3.B). The association matrix $\mathbf{H} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ defines the node-hyperedge affiliation relationship. When $\mathbf{H}_{i,j} = 1$, it means that node v_i belongs to hyperedge e_j.

Gao et al. [4] propose a two-stage spatial convolution method to obtain hyperedge features and node features. In order to capture high-order relationships between components, we use a deep network to model the complex relationships of microservice systems through multi-layer stacking. First, input node features $\mathbf{X}_t$, with a dimension of $N \times C$, where N represents the number of nodes, and C represents the feature dimension; the hypergraph structure is represented by the association matrix $\mathbf{H}$, with a dimension of $N \times M$, where M is the number of hyperedges; the initial hyperedge feature $\mathbf{Y}_t$. Secondly, using the adaptive weight mechanism, the learnable hyperedge weight W is adopted to dynamically adjust the importance of different hyperedges. Finally, the hyperedge feature Y' output from the last convolution layer is used as the input of the MLP.

$$\mathbf{Y}_i = \sigma\left(\mathbf{D}_e^{-1}\mathbf{H}^\top \mathbf{X}_{i-1}\mathbf{W}\right) \tag{1}$$

$$\mathbf{X}_i = \sigma\left(\mathbf{D}_v^{-1/2}\mathbf{H}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}^\top \mathbf{D}_v^{-1/2}\mathbf{X}_{i-1}\mathbf{\Theta}\right) \tag{2}$$

$$\mathbf{Y}^{'} = Multi - ConV - Layer(\mathbf{X}_i, \mathbf{Y}_i) \tag{3}$$

Among them, $\mathbf{X}_i$ and $\mathbf{Y}_i$ refer to the node features and hyperedge features after the $i - th$ layer of convolution respectively. σ represents the nonlinear activation function. $\mathbf{D}_e$ represents the diagonal matrix of the hyperedge degree. $\mathbf{D}_v$ represents the diagonal matrix of the node degree. $\mathbf{\Theta}$ represents the learnable parameters of the hypergraph neural network. $Multi - ConV - Layer()$ refers to the hyperedge features obtained through multi-layer convolution.

HyperRCA uses a multi-layer perception (MLP) to implement multi-granular root cause localization, and the final hyperedge feature $\mathbf{Y}^{'}$ is used as the input of the root cause localization model. MLP maps features to the hidden layer, and then enhances the expression ability through a nonlinear activation function. The hidden layer output is mapped to the category space through a fully connected layer to generate a classification score matrix $\mathbf{Z}$. Rectified Linear Unit (ReLU) is a commonly used activation function that can introduce nonlinear characteristics to the neural network and enhance the model's expression ability.

$$\mathbf{Z} = \mathbf{W}_m\mathbf{Y}^{'} + b \tag{4}$$

$$\mathbf{h} = ReLU(\mathbf{Z}) \tag{5}$$

Where $\mathbf{W}_m$ is parameter learned in the MLP and b is bias term. Furthermore, $softmax$ is used for normalization after the fully connected layer to convert the classification scores into probability distributions. Among them, $\mathbf{P}$ is the category probability matrix of the hyperedge, with dimension $M \times K$, where K is the number of root cause categories. Each row P_i represents the probability that hyperedge i is the root cause.

$$\mathbf{P} = Softmax(\mathbf{Z}) \tag{6}$$

Next, HyperRCA uses the cross-entropy loss function to train the model. Assuming the true label of the hyperedge is $\mathbf{Y}_{true}$ (dimension is $M \times 1$, and each element is the category index of the hyperedge), the loss can be defined as:

$$loss = -\frac{1}{M}\sum_{i=1}^{M}\sum_{k=1}^{K} Y_{true}^{(i,k)} \log(P^{(i,k)}) \tag{7}$$

Among them, $Y_{true}^{(i,k)}$ is the one-hot encoding of the true label of hyperedge i, and $P^{(i,k)}$ is the predicted probability that hyperedge i belongs to category k.Finally, the classifier is trained through forward propagation, loss calculation, back-propagation, and parameter update.

D. Online Root Cause Localization

In online root cause localization, when the system detects an anomaly, HyperRCA is triggered to start root cause localization. First, the data $\mathbf{X}_t$ in the fault occurrence time window is serialized using the method proposed in §3.A, and the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{H})$ is constructed using the hypergraph construction method proposed in §3.B. Then, the hypergraph $\mathcal{G}$ is used as the input of the hypergraph convolution to generate the hyperedge feature $\mathbf{Y}^{'}$ that integrates the high-order relationships between components when the fault occurs.

$$\mathbf{Y}^{'} = Multi - ConV - Layer(\mathbf{X}_t, \mathbf{Y}_t) \tag{8}$$

Finally, the hyperedge feature $\mathbf{Y}^{'}$ is used as the input of the MLP, the hidden layer is mapped to the category space through fully connected layer, and root cause ranking is obtained after the Softmax normalization operation. $MLP()$ is a multi-layer perceptron model for root cause localization task.

$$\mathbf{P} = MLP(\mathbf{Y}^{'}) \tag{9}$$

HyperRCA ranks the suspected root causes from high to low according to the probability score. The higher the ranking, the more likely it is to be the root cause.

IV. Experiment

To evaluate HyperRCA, we conduct a series of experimental studies to investigate the following research questions:

- **RQ1**: How accurate is HyperRCA in root cause localization compared to baseline methods?
- **RQ2**: How efficient and scalable is HyperRCA in root cause localization compared to baseline methods?
- **RQ3**: How much do different types of hyperedges of HyperRCA contribute to the root cause localization?
- **RQ4**: What is the impact of the number of hypergraph convolutional layers on the root cause localization?

A. Experimental Setup

1) Dataset: To evaluate the fairness of our results, we conducted experimental verification on two datasets. GAIA is collected and maintained by Cloudwise [41]. Engineers constructed this dataset by simulating possible faults in real systems. GAIA has been widely used as evaluation dataset in various papers [6], [7], [11]. The fault granularity in GAIA is instance-granularity. Trainticket [42] is a medium-sized open source microservice benchmark architecture system. It has been used in numerous papers for fault injection and evaluation [5], [12]–[14], [37]. We used ChaosBlade [43], a chaos engineering tool widely used in industry and academia, for fault injection. We injected faults at three granularities: host, service, and instance. Table II provides detailed dataset.

TABLE II: Details of Datasets

Dataset	Granularity	Fault Type	Count
D1	Instance	System Stuck, Process Crash, Login Failure, File Missing, Access Denied	1099
D2	Host	CPU Overload	25
	Service	Network Delay, Network Loss, CPU Overload	90
	Instance	Instance is unavailable, CPU Overload	95

Dataset 1 (D1): This dataset is Generic AIOps Atlas (GAIA) dataset. The system is deployed on 5 distributed hosts, including 6 services and 11 instances in total. The dataset contains 1,099 faults of five types: system freeze (505), process crash (16), login failure (527), file loss (36), and access denied (15). The GAIA dataset collects multimodal data from the MicroSS system within two weeks, including 6,500 metrics, 700,000 log, and detailed trace within two weeks. The dataset contains 1099 fault cases with fault durations of 600s or 11s.

Dataset 2 (D2): To evaluate the performance of HyperRCA in multi-granularity root cause localization, we chose to inject faults at the Host, Service, and Instance granularities on Trainticket [42]. Trainticket contains 47 services. We deployed Trainticket on a Kubernetes cluster with 12 hosts. We referred to existing studies [5], [14], [37] based on Trainticket. We used ChaosBlade [43] to inject faults. We injected six types of faults at three granularities, as shown in the Table II. We inject faults into the same target at different times and may inject the same fault type. Each experiment involved injecting a single fault that lasted for 600s. For example, CPU overload faults were introduced at the host, service (all service instances belong to this service), and instance granularities, and marked the fault injection target as the root cause of each fault. In total, we collected a total of 210 faults, including 25 host granularity faults, 90 service granularity faults, and 95 instance granularity faults.

2) Baselines: In recent years, many works have proved that multimodal methods have significant advantages over single-modal methods. Therefore, we select six state-of-the-art methods based on multimodal data to compare with HyperRCA.

- **Nezha [5]**: The core idea is to integrate multimodal observable data, uniformly represent them as time-related homogeneous events, and build a global event graph. Nezha supports root cause localization at the instance and code area granularities.
- **DiagFusion [7]** extracts abnormal events from multimodal data and builds a service dependency graph to implement the root cause localization task. The root cause localization task of this method is only implemented in the instance granularity.
- **DeepHunt [8]** is a method for locating the root cause of microservice system failures based on graph autoencoders. The root cause localization task of this method is only implemented in the instance granularity.
- **TVdiag [6]** is a task-oriented and view-invariant multimodal fault diagnosis framework. This method aggregates multimodal alarm features with graph neural networks and jointly optimizes multi-task learning through dynamic weights. The granularity of the root cause localization task is only in the instance granularity.
- **Eadro [14]** is an end-to-end microservice fault diagnosis framework. This method integrates multimodal features through GRU and builds a service dependency graph to achieve root cause localization. The root cause localization task of this method is only implemented in the service granularity.
- **TrinityRCL [35]** is able to locate the root causes of anomalies at multiple granularities by constructing a causal graph that represents the complex, dynamic, and uncertain relationships between various entities related to the anomaly.

3) Evaluation Metrics: The root cause localization task generates a ranked list $\mathcal{R}$ containing root cause candidates. We use the widely used $HR@k$ (top-k hit rate) and mean reciprocal rank (MRR) to measure the performance.

HR@k represents the probability of the root cause being included in the top-k list (k=1, 3, 5 in this paper). The formula for $HR@k$ is as follows:

$$HR@k = \frac{1}{N}\sum_{i=1}^{N} r_i \in \mathcal{R}_i[1:k] \tag{10}$$

MRR is the multiplicative inverse of the rank of the root cause in the result list. If the root cause is not included in the top 10 result lists, the rank can be considered positive infinity [25]. Given a set of fault instances, where Rank_i is the rank of the root cause in the returned list of the fault instance, MRR is calculated as follows:

$$MRR = \frac{1}{|A|}\sum_{i=1}^{|A|} \frac{1}{\mathrm{Rank}_i} \tag{11}$$

4) Implementation and Settings: For the division of training and test sets in the dataset, we use the division method proposed by DiagFusion [7] for fair comparison. The training samples and test samples are divided according to the acquisition time to avoid data leakage. We used 60% of the data for training, 20% for verifying and 20% for testing.

We selected 10 hosts, 20 services, and 34 instances for fault injection. We chose multi-instance deployment for some services. Both the training and test sets include all six fault types, but the locations of the faults in the training and test sets differ. We injected multiple types of faults into the same target, and the same fault type was injected at different times. Fig. 5 shows the locations of fault injection in training and test sets. For example, at host-level, training set includes fault cases injected on 8 hosts, and test set includes fault cases injected on 4 hosts. Different fault cases injected on the same 2 hosts are present in both training and test sets. Although the locations of the cases overlap, the same fault case does not appear in both training and test sets.

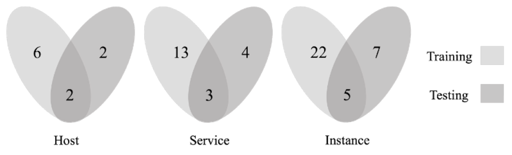


Fig. 5: Fault location on $D2$

During model training, we used the Adaptive Moment Estimation (Adam) optimizer with an initial learning rate of 3×10^{-4} and a weight decay of 5×10^{-4}. The implementation of HyperRCA is based on $python$ 3.8, $Pytorch$ 2.4.1, and DHG 0.9.4. All experiments are performed on a workstation with NVIDIA GeForce GTX 4090 GPU. We repeat each experiment five times and average the results to minimize the effect of randomness. We have published our code on github.

B. RQ1: Accuracy

Table III presents the comparative results of instance-granularity root cause localization on $D1$. HyperRCA achieves state-of-the-art performance with $HR@5$ of 0.910 and MRR of 0.770, demonstrating a performance improvement of up to 112.62% in $HR@5$ and 181.02% in MRR compared to baseline methods. Notably, DiagFusion ranks second in accuracy due to its multimodal event alignment mechanism, while DeepHunt exhibits the lower performance as an unsupervised approach susceptible to synthetic data artifacts. The Nezha method also extracts events from multimodal data, and designs an expected ranker and an actual ranker to output the root cause, so it also achieves good experimental results. TVdiag shows good performance in accuracy. It uses task-oriented learning to enhance the potential contribution of each modality to the corresponding task.

DeepHunt performed poorly because it is an unsupervised method, which has a lower accuracy than supervised methods. However, it can be expected that in the case of missing labels, unsupervised methods will be significantly higher than supervised methods. In addition, the data enhancement method used in DeepHunt will produce data that does not match the

TABLE III: Accuracy at Instance Granularity

DataSet	Method	HR@1	⇑HR@1	HR@3	⇑HR@3	HR@5	⇑HR@5	MRR	⇑MRR
D1	Nezha	0.577	18.37%	0.746	15.28%	0.831	9.51%	0.761	1.18%
	DiagFusion	0.420	62.62%	0.819	5.00%	0.907	0.33%	0.745	3.36%
	DeepHunt	0.109	526.61%	0.403	113.40%	0.648	40.43%	0.334	130.54%
	TVdiag	0.589	16.00%	0.766	12.27%	0.839	8.46%	0.734	4.91%
	Eadro	0.357	91.32%	0.576	49.31%	0.791	15.04%	0.680	13.24%
	TrinityRCL	0.112	509.82%	0.249	245.38%	0.428	112.62%	0.274	181.02%
	HyperRCA(Ours)	**0.683**	-	**0.860**	-	**0.910**	-	**0.770**	-

actual situation, which will greatly reduce the accuracy of the experimental results.

TrinityRCL locates the root cause by combining a causal graph with random walks. Firstly, the construction of the causal graph may contain redundant or erroneous causal edges. Secondly, random walks tend to prioritize nodes that are strongly correlated with anomalies but are not the root cause. This combination leads to insufficient localization accuracy.

For multi-granularity root cause localization, as shown in Table V, HyperRCA performs well at the host, service, and instance granularities, demonstrating that HyperRCA excels not only at a single granularity. Overall, HyperRCA achieves HR@5 of 0.810 and MRR of 0.696. Compared to baselines, these improvements reach 466.43% and 1060%, respectively.

We statistically analyze the results of HyperRCA and baseline methods at three granularities. Since Nezha, DiagFusion, DeepHunt, TVdiag, and Eadro only support root cause localization at the instance or service granularity, we only present their results at the instance or service level. As a result, these baseline methods perform poorly in the overall results.

While TrinityRCL supports multi-granularity root cause localization, the causal graph model it uses is prone to redundant or erroneous causal edges. Furthermore, TrinityRCL primarily constructs causal graphs based on call relationships, but fault propagation is not solely based on call relationships. These factors contribute to the poor experimental results.

In summary, compared with the baseline method, HyperRCA's hypergraph-based representation captures high-order component relationships that simple graph models fail to encode. This enables precise identification of cross-layer fault propagation paths. The serialization method (§3.A) demonstrates stronger temporal correlation capture than DiagFusion's event alignment, particularly evident in complex multi-granular scenarios. Baseline methods like Eadro (service-level) and TVdiag (instance-level) show constrained performance when handling mixed-granularity faults, while HyperRCA maintains consistent accuracy through its hyperedge design.

C. RQ2: Efficiency and Scalability

1) Efficiency: To evaluate the efficiency of HyperRCA and baseline methods, we compare their average execution time during testing. We exclude model training time from comparison since root cause localization models typically undergo offline training, with our focus being exclusively on inference efficiency.

TABLE IV: Efficiency

Dataset	Method	Per Test
D1	Nezha	3.168s
	DiagFusion	0.021s
	DeepHunt	0.003s
	TVdiag	0.013s
	Eadro	0.031s
	TrinityRCL	1.653s
	HyperRCA(Ours)	0.893s
D2	Nezha	5.106s
	DiagFusion	0.062s
	DeepHunt	0.017s
	TVdiag	0.048s
	Eadro	0.101s
	TrinityRCL	0.977s
	HyperRCA(Ours)	0.861s

As shown in Table IV, the average test execution times of HyperRCA and baseline methods range from 0.003 to 3.168 seconds on $D1$. Nezha demonstrates particularly high computational complexity due to its unsupervised pattern mining mechanism, which requires real-time construction of cross-service event graphs and pattern comparison, resulting in exponentially increasing computation overhead with system scale. TrinityRCL, like Nezha, does not train a root cause localization model. It constructs a causal graph and employs a random walk approach for root cause localization. This approach consumes more time than other baselines. Other baseline methods leverage pre-trained feature extractors to enable lightweight inference processes, though this computational advantage comes at the cost of compromised accuracy in complex scenarios.

On the multi-granularity dataset $D2$, the execution times range from 0.017 to 5.106 seconds. Baseline methods generally exhibit longer execution times compared to their performance on $D1$, attributable to the increased complexity of multi-granularity fault patterns. Notably, HyperRCA shows marginally reduced execution time on $D2$, suggesting enhanced adaptability when handling multi-granular failures.

While HyperRCA's hypergraph construction and hypergraph convolution operations introduce higher computational complexity than most baseline methods, resulting in moderately lower efficiency, we consider this trade-off acceptable given its statistically significant superiority in localization accuracy for both single-granularity and multi-granularity scenarios. The architectural sophistication of HyperRCA ultimately contributes to its robust performance across varying granularity levels

TABLE V: Accuracy at Multiple Granularities

Dataset	Method	Evaluation	Instance-level	⇑	Service-level	⇑	Host-level	⇑	Overall	⇑
D2	Nezha	HR@1	0.444	0%	-	-	-	-	0.190	188.42%
		HR@3	0.667	16.65%	-	-	-	-	0.286	174.83%
		HR@5	0.667	24.91%	-	-	-	-	0.286	183.22%
		MRR	0.556	17.27%	-	-	-	-	0.238	192.44%
	DiagFusion	HR@1	**0.500**	-11.20%	-	-	-	-	0.214	156.07%
		HR@3	0.667	16.65%	-	-	-	-	0.286	174.83%
		HR@5	0.722	15.38%	-	-	-	-	0.310	161.29%
		MRR	0.597	9.21%	-	-	-	-	0.256	171.88%
	DeepHunt	HR@1	0	-	-	-	-	-	0	-
		HR@3	0.222	250.45%	-	-	-	-	0.095	727.37%
		HR@5	0.333	150.15%	-	-	-	-	0.143	466.43%
		MRR	0.139	369.06%	-	-	-	-	0.060	1060%
	TVdiag	HR@1	0.222	100%	-	-	-	-	0.095	476.84%
		HR@3	0.556	39.93%	-	-	-	-	0.238	230.25%
		HR@5	0.722	15.38%	-	-	-	-	0.310	161.29%
		MRR	0.431	51.28%	-	-	-	-	0.185	276.22%
	Eadro	HR@1	-	-	0.353	83.29%	-	-	0.143	283.21%
		HR@3	-	-	0.706	8.36%	-	-	0.286	174.83%
		HR@5	-	-	**0.765**	0%	-	-	0.310	161.29%
		MRR	-	-	0.220	214.09%	-	-	0.289	140.83%
	TrinityRCL	HR@1	0.222	100%	0.235	175.32%	0.143	299.30%	0.214	156.07%
		HR@3	0.333	133.63%	0.353	85.68%	0.286	199.65%	0.286	174.83%
		HR@5	0.389	114.14%	0.412	85.68%	0.286	199.65%	0.310	161.29%
		MRR	0.292	123.09%	0.309	123.62%	0.201	272.64%	0.259	168.72%
	HyperRCA	HR@1	0.444	-	**0.647**	-	**0.571**	-	**0.548**	-
		HR@3	**0.778**	-	**0.765**	-	**0.857**	-	**0.786**	-
		HR@5	**0.833**	-	**0.765**	-	**0.857**	-	**0.810**	-
		MRR	**0.652**	-	**0.691**	-	**0.750**	-	**0.696**	-

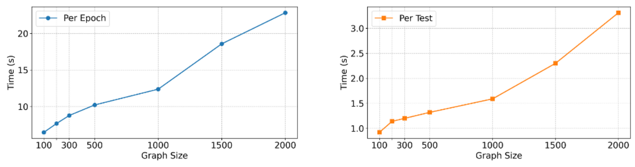
without substantial temporal overhead escalation.

2) Scalability: HyperRCA consumes the most time during the hypergraph construction, training, and testing phases. We analyzed the time complexity of the hypergraph construction. We constructed datasets with varying node sizes to analyze the time consumption trends during model training and testing.

Within an time window, constructing a hypergraph with n nodes and m hyperedges e_i, each containing $|e_i|$ nodes, primarily involves building the incidence matrix $H \in \mathbb{R}^{n\times m}$. The construction complexity is $\mathcal{O}(\sum_{i=1}^{m}|e_i|)$, which corresponds to the total number of memberships between nodes and hyperedges, denoted as E. In the worst case, when every hyperedge covers all nodes, the complexity reaches $\mathcal{O}(mn)$. Generally, if each hyperedge contains on average $\bar{d}$ nodes, the complexity is $\mathcal{O}(m\bar{d})$. Therefore, the construction of the hypergraph mainly depends on the number of hyperedges and their node coverage, and it does not exceed $\mathcal{O}(mn)$.

To evaluate the scalability of HyperRCA on systems of different sizes, we constructed seven datasets of varying sizes, containing 100, 200, 300, 500, 1000, 1500, and 2000 nodes, respectively. Each dataset contains 300 cases. Using the DHG package, simple graphs were converted into hypergraphs with randomly generated hyperedges to ensure generality. To eliminate the impact of feature dimensionality, all nodes were assigned features of the same dimension as those in $D1$ and $D2$, enabling a fair comparison across different graph sizes.

The training results show that the time per epoch increases



(a) Per Epoch at different scales (b) Per Test at different scales

Fig. 6: Performance comparison under different graph sizes

significantly with graph size. For example, as the number of nodes increases from 100 to 2000, the training time per epoch rises from approximately 6s to 22s. This trend highlights that in large-scale hypergraphs, the computational cost of the hypergraph convolution layers becomes the dominant factor and scales with both the number of nodes and the complexity of hyperedges. Nevertheless, the curve exhibits a smooth growth, suggesting that HyperRCA maintains stable scalability as the graph size expands.

The inference time for test also increases with graph size, but remains relatively low. Even for 2000 nodes, the test time does not exceed 4s. This indicates that HyperRCA incurs only minor overhead in online inference, which is critical for real-time root cause localization in large-scale systems.

Overall, HyperRCA has a tolerable training time and achieves efficient inference at large scale. HyperRCA leverages hypergraph modeling to capture complex dependencies and improve localization accuracy, while maintaining fast in-

TABLE VI: Ablation

Dataset	Method	HR@1	⇑HR@1	HR@3	⇑HR@3	HR@5	⇑HR@5	MRR	⇑MRR
$D1$	HyperRCA w/o DTH	0.579	17.96%	0.828	3.86%	0.869	4.72%	0.700	10.00%
	HyperRCA w/o SEH	0.634	7.73%	**0.887**	-3.04%	**0.919**	-0.98%	0.762	1.05%
	HyperRCA w/o DYH	0.656	4.12%	0.846	1.65%	0.878	3.64%	0.744	3.49%
	HyperRCA	**0.683**	-	0.860	-	0.910	-	**0.770**	-
$D2$	HyperRCA w/o DYH	0.450	11.11%	0.749	0.27%	0.800	6.25%	0.586	6.99%
	HyperRCA	**0.500**	-	**0.751**	-	**0.849**	-	**0.627**	-

ference online. Thus, it offers both effectiveness and scalability for practical use in microservice systems of varying scales.

D. RQ3: Ablation Study

To verify the impact of different types of hyperedges on the experimental results, we create three variants: deployment hyperedges, subordinate hyperedges, and dependency hyperedges. These variants include: 1) HyperRCA w/o DTH. To study the role of cross-distributed host and instance modeling in obtaining hyperedge features, we remove the deployment hyperedge from HyperRCA. 2) HyperRCA w/o SEH. To study the impact of cross-service and instance granularity modeling on the experimental results, we remove the subordinate hyperedge from HyperRCA. 3) HyperRCA w/o DYE. To study the impact of call relationships on the experimental results, we remove the dependency hyperedge from HyperRCA.

Table VI lists the specific performance and change range of each variant on single-granularity and multi-granularity datasets. For the single-granularity dataset $D1$, when the deployment hyperedge is removed, $HR@1$ decreases by 17.96% and MRR decreases by 10.00%. The results show that modeling the relationship between distributed hosts and instances helps to obtain more fault-related information. When the dependency hyperedges are removed, $HR@k(k = 1, 3, 5)$ and MRR are reduced to varying degrees. Dependencies model the relationship of frequent calls between instances and are also one of the ways for fault propagation.

When the subordinate hyperedges are removed, $HR@3$ and $HR@5$ are slightly improved, while $HR@1$ and MRR are reduced. This is because the main function of subordinate hyperedges is to associate all instances of a service. It can also help distinguish the status of different instances of the same service. However, $D1$ mainly has instance-granularity faults, but no service-granularity faults. Therefore, after removing the subordinate hyperedges, the hypergraph will pay more attention to instance-granularity features and faults.

On multi-granularity datasets $D2$, we only verify the impact of dependency hyperedges on root cause localization results. This is because the faults in multi-granularity datasets are on distributed hosts, services, and instances. If the deployment hyperedges are removed, the faults of distributed hosts will be ignored, and if the subordinate hyperedges are removed, the faults of services will be ignored. Therefore, we only verify the impact of dependency hyperedges on multi-dimensional root cause localization results.

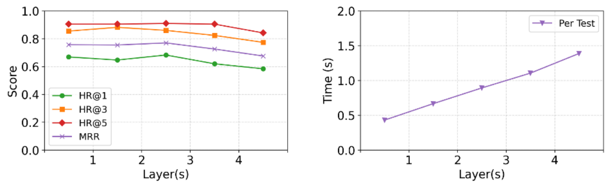
The experimental results show that dependency hyperedges will affect the accuracy, but the degree of influence is lower than that of single-granularity root cause localization. For multi-granularity root cause localization, deployment hyperedges and subordinate hyperedges play a greater role because these two types of hyperedges represent fault granularity.

In summary, all three types of hyperedges play essential roles in root cause localization. Deployment hyperedges capture host-instance relations, subordinate hyperedges enhance service-instance distinctions, and dependency hyperedges model call interactions. Their combined effect ensures accurate multi-granularity fault root cause localization.

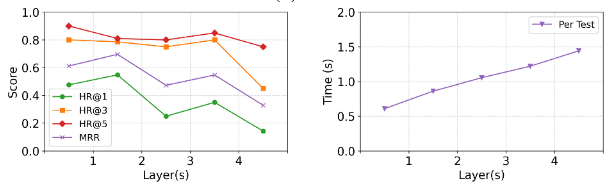
E. RQ4: Hyperparameters Sensitivity

We mainly discuss the impact of the number of hypergraph convolution layers on the performance of HyperRCA when constructing hyperedges. The $D1$ dataset only contains faults in the instance dimension, the system topology is small, and the service dependencies are relatively simple. When the number of hypergraph convolution layers increases from 0.5 to 2.5 layers, $HR@1$ increases from 0.670 to 0.683, and MRR increases from 0.757 to 0.770, indicating that increasing the number of layers can effectively capture fine-grained dependencies between instances. However, when the number of layers exceeds 3.5, $HR@1$ drops to 0.584 and MRR drops to 0.676, indicating that too deep a network will introduce redundant information, causing the model to overfit to simple topologies. In contrast, the $D2$ dataset contains multi-dimensional faults and the system complexity is significantly higher. When the number of layers increases from 0.5 to 1.5, $HR@1$ increases from 0.476 to 0.548, but when it further increases to 4.5 layers, $HR@1$ drops sharply to 0.143 and MRR drops from 0.696 to 0.329. This shows that multi-layer hypergraph convolution in complex systems may destroy the semantic consistency of cross-dimensional fault propagation paths, and deep networks have difficulty distinguishing noise from effective features.

For $D1$, the Per Test time increases from 0.431 s at 0.5 layers to 1.388 s at 4.5 layers, while $HR@1$ decreases by only 12.71%, suggesting that efficiency can be reasonably traded for accuracy in simple systems. In contrast, for $D2$, the Per Test time rises from 0.609 s at 0.5 layers to 1.443 s at 4.5 layers, accompanied by a drastic 66.7% decline in $HR@1$, indicating that in complex systems, the marginal benefits of deeper models diminish sharply.



(a) $D1$



(b) $D2$

Fig. 7: Performance of HyperRCA under different parameters

Experiments show that the impact of the number of hypergraph convolution layers on HyperRCA performance is highly dependent on the topological complexity and fault granularity characteristics of the dataset. In instance fault scenario ($D1$), the 2.5-layer model can balance feature extraction and computational efficiency. While in multi-granularity fault scenario ($D2$), the 1.5-layer model can effectively suppress noise interference. In summary, the more hypergraph convolution layers there are, the better the effect is not necessarily.

F. Threats to Validity

The internal validity threats primarily stem from potential inconsistencies in the baseline implementations and biases in parameter settings. For baseline methods with publicly available codes, we strictly follow their official code bases to ensure reproducibility. Since the original code of TrinityRCL is not open source, we reproduced the experiments based on the publicly available implementation. Although all baseline methods were tuned via grid search, the varying sensitivity of different methods to identical parameters (e.g., learning rates, batch sizes) might partially inflate or attenuate performance differences. In addition, while preprocessing procedures (e.g., normalization, missing value imputation) were standardized, method-specific assumptions about input distributions may still introduce latent biases.

The external validity threats mainly arise from dataset scale and fault coverage. The GAIA dataset (D1), though collected from a real-world microservice system, contains only 5 hosts and 11 instances. The Trainticket dataset (D2) expands to host, service, and instance levels but remains simplified, with 47 services in a static 12-host Kubernetes cluster, lacking cloud-native dynamics such as scaling or rolling updates. These limitations may restrict the ecological validity of our findings in large-scale production environments.

V. Related Work

Root cause localization in microservice systems has attracted increasing attention. Existing approaches can be divided into unimodal and multimodal methods according to the type of data used. Unimodal methods rely on a single modality, such as logs [38], [44], [45], metrics [36], [46], or traces [37], [47]–[50]. Multimodal methods combine at least two modalities, and numerous studies have shown they achieve higher accuracy and recall.

Recent years have seen a surge of multimodal approaches. MULAN [13] applies a customized language model for logs, metric-aware attention for reliability estimation, and causal graph fusion. Nezha [5] constructs event graphs and mines comparative patterns for fine-grained localization. DiagFusion [7], ART [9], and DeepHunt [8] serialize multi-source data into graph nodes, while Eadro [14] leverages call dependencies and GATs for end-to-end fault diagnosis.

Most existing works adopt graph neural networks. However, traditional GNNs capture only pairwise relations and struggle with the complex interactions of microservices. Hypergraphs have been introduced. CHASE [11] models causal propagation across multiple nodes with hyperedges, but it considers only trace data. HyperRCA extends this idea by constructing multiple types of hyperedges to capture diverse relationships.

Furthermore, as shown in Table I, most methods [6]–[12] focus on instance- or service-level localization, rarely supporting multi-granularity. This limitation can cause false negatives under diverse fault scenarios. For example, TraceContrast [37], though unimodal, demonstrates the benefit of modeling multi-dimensional performance patterns via critical path mining. Motivated by this, HyperRCA aims to achieve multi-granular root cause localization.

VI. Conclusion

In this work, we present HyperRCA, a hypergraph neural network-based method for multi-granular root cause localization. By modeling instances as nodes and designing deployment, subordinate, and dependency hyperedges, HyperRCA captures cross-granularity relationships and leverages hypergraph convolution to extract high-order correlations. Experiments on single-granularity and multi-granularity datasets show that HyperRCA consistently outperforms baselines in $HR@k$ ($k = 1, 3, 5$) and MRR, highlighting the advantages of hypergraph modeling for complex system dependencies.

VII. Acknowledgmet

This work is supported in part by National Natural Science Foundation of China under Grant U21B2015, Grant 62372351, the Fundamental Research Funds for the Central Universities and the Innovation Fund of Xidian University under Grant YJSJ25012, the Young Talent Fund of Association for Science and Technology in Shaanxi under Grant 20220113, the Proof of Concept Foundation of Xidian University Hangzhou Institute of Technology under Grant XJ2023230039, the Natural Science Foundation of Jiangsu Province under Grant BK20232028, the Natural Science Basic Research Program of Shaanxi under Grant 2025JC-YBQN-851.

References

[1] (2023) Cloud native 2023: The undisputed infrastructure of global technology. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2023/

[2] P. Insurance, "Cloud outage and the fortune 500: An impact analysis," Parametrix Solutions Inc., Tech. Rep., 2023. [Online]. Available: https://www.parametrixinsurance.com

[3] S. Zhang, S. Xia, W. Fan, B. Shi, X. Xiong, Z. Zhong, M. Ma, Y. Sun, and D. Pei, "Failure diagnosis in microservice systems: A comprehensive survey and analysis," *ACM Trans. Softw. Eng. Methodol.*, Jan. 2025, just Accepted. [Online]. Available: https://doi.org/10.1145/3715005

[4] Y. Gao, Y. Feng, S. Ji, and R. Ji, "Hgnn$^+$: General hypergraph neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3181–3199, 2023. [Online]. Available: https://doi.org/10.1109/TPAMI.2022.3182052

[5] G. Yu, P. Chen, Y. Li, H. Chen, X. Li, and Z. Zheng, "Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 553–565. [Online]. Available: https://doi.org/10.1145/3611643.3616249

[6] S. Xie, J. Wang, H. He, Z. Wang, Y. Zhao, N. Zhang, and B. Li, "Tvdiag: A task-oriented and view-invariant failure diagnosis framework with multimodal data," 2024. [Online]. Available: https://arxiv.org/abs/2407.19711

[7] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin, D. Zhang, Z. Zhu, and D. Pei, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 3851–3864, 2023. [Online]. Available: https://doi.org/10.1109/TSC.2023.3290018

[8] Y. Sun, Z. Lin, B. Shi, S. Zhang, S. Ma, P. Jin, Z. Zhong, L. Pan, Y. Guo, and D. Pei, "Interpretable failure localization for microservice systems based on graph autoencoder," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 2, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3695999

[9] Y. Sun, B. Shi, M. Mao, M. Ma, S. Xia, S. Zhang, and D. Pei, "Art: A unified unsupervised framework for incident management in microservice systems," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1183–1194. [Online]. Available: https://doi.org/10.1145/3691620.3695495

[10] L. Zheng, Z. Chen, H. Chen, and J. He, "Online multi-modal root cause analysis," 2024. [Online]. Available: https://arxiv.org/abs/2410.10021

[11] Z. Zhao, T. Zhang, Z. Shen, H. Dong, X. Ma, X. Liu, and Y. Yang, "Chase: A causal heterogeneous graph based framework for root cause analysis in multimodal microservice systems," 2024. [Online]. Available: https://arxiv.org/abs/2406.19711

[12] F. Liu, Y. Wang, Z. Li, R. Ren, H. Guan, X. Yu, X. Chen, and G. Xie, "Microcbr: Case-based reasoning onnbsp;spatio-temporal fault knowledge graph fornbsp;microservices troubleshooting," in *Case-Based Reasoning Research and Development: 30th International Conference, ICCBR 2022, Nancy, France, September 12–15, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 224–239. [Online]. Available: https://doi.org/10.1007/978-3-031-14923-8_15

[13] L. Zheng, Z. Chen, J. He, and H. Chen, "Mulan: Multi-modal causal structure learning and root cause analysis for microservice systems," in *Proceedings of the ACM Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 4107–4116. [Online]. Available: https://doi.org/10.1145/3589334.3645442

[14] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "Eadro: An end-to-end troubleshooting framework for microservices on multi-source data," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 1750–1762. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00150

[15] C. Hou, T. Jia, Y. Wu, Y. Li, and J. Han, "Diagnosing performance issues in microservices with heterogeneous data source," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, September 30 - Oct. 3, 2021*. IEEE, 2021, pp. 493–500. [Online]. Available: https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00074

[16] (2025). [Online]. Available: https://github.com/Apex-ISET/HyperRCA

[17] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, "Microhecl: high-efficient root cause localization in large-scale microservice systems," in *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '21. IEEE Press, 2021, p. 338–347. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP52600.2021.00043

[18] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, and D. Pei, "Practical root cause localization for microservice systems via trace analysis," in *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*. IEEE, 2021, pp. 1–10. [Online]. Available: https://doi.org/10.1109/IWQOS52092.2021.9521340

[19] (2025) What is opentelemetry? [Online]. Available: https://opentelemetry.io/docs/concepts/signals/

[20] (2025) Meet the search platform that helps you search, solve, and succeed. [Online]. Available: https://www.elastic.co/elastic-stack

[21] (2025) Meet ai assistant in observability cloud. [Online]. Available: https://www.splunk.com/

[22] (2025) From metrics to insight. [Online]. Available: https://prometheus.io/

[23] (2025) Making your observability stack faster and easier. [Online]. Available: https://grafana.com/

[24] (2025) Jaeger: open source, distributed tracing platform. [Online]. Available: https://www.jaegertracing.io/

[25] (2025) Zipkin is a distributed tracing system. [Online]. Available: https://zipkin.io/

[26] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, B. Schölkopf, J. C. Platt, and T. Hofmann, Eds. MIT Press, 2006, pp. 1601–1608. [Online]. Available: https://proceedings.neurips.cc/paper/2006/hash/dff8e9c2ac33381546d96deea9922999-Abstract.html

[27] X. Xia, H. Yin, J. Yu, Q. Wang, L. Cui, and X. Zhang, "Self-supervised hypergraph convolutional networks for session-based recommendation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, pp. 4503–4511, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16578

[28] X. Liao, Y. Xu, and H. Ling, "Hypergraph neural networks for hypergraph matching," in *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 1246–1255. [Online]. Available: https://doi.org/10.1109/ICCV48922.2021.00130

[29] K. Ding, J. Wang, J. Li, D. Li, and H. Liu, "Be more with less: Hypergraph attention networks for inductive text classification," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 4927–4936. [Online]. Available: https://aclanthology.org/2020.emnlp-main.399/

[30] J. Zhu, J. Zhu, S. Ghosh, W. Wu, and J. Yuan, "Social influence maximization in hypergraph in social networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 6, no. 4, pp. 801–811, 2019. [Online]. Available: https://doi.org/10.1109/TNSE.2018.2873759

[31] S. Yi and D.-S. Lee, "Structure of international trade hypergraphs," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2022, no. 10, p. 103402, oct 2022. [Online]. Available: https://dx.doi.org/10.1088/1742-5468/ac946f

[32] S. Feng, E. Heath, B. A. Jefferson, C. A. Joslyn, H. Kvinge, H. D. Mitchell, B. Praggastis, A. J. Eisfeld, A. C. Sims, L. B. Thackray, S. Fan, K. B. Walters, P. J. Halfmann, D. Westhoff-Smith, Q. Tan, V. D. Menachery, T. P. Sheahan, A. S. Cockrell, J. F. Kocher, K. G. Stratton, N. C. Heller, L. M. Bramer, M. S. Diamond, R. S. Baric, K. M. Waters, Y. Kawaoka, J. E. McDermott, and E. Purvine, "Hypergraph models of biological networks to identify genes critical to pathogenic viral response," *BMC Bioinform.*, vol. 22, no. 1, p. 287, 2021. [Online]. Available: https://doi.org/10.1186/s12859-021-04197-2

[33] K. Grzesiak-Kope, P. Oramus, and M. Ogorzaek, "Hypergraphs and extremal optimization in 3d integrated circuit design automation,"

Adv. Eng. Inform., vol. 33, no. C, p. 491–501, Aug. 2017. [Online]. Available: https://doi.org/10.1016/j.aei.2017.06.004

[34] H. Zhou, M. Chen, Q. Lin, Y. Wang, X. She, S. Liu, R. Gu, B. C. Ooi, and J. Yang, "Overload control for scaling wechat microservices," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 149–161. [Online]. Available: https://doi.org/10.1145/3267809.3267823

[35] S. Gu, G. Rong, T. Ren, H. Zhang, H. Shen, Y. Yu, X. Li, J. Ouyang, and C. Chen, "Trinityrcl: Multi-granular and code-level root cause localization using multiple types of telemetry data in microservice systems," *IEEE Trans. Softw. Eng.*, vol. 49, no. 5, p. 3071–3088, May 2023. [Online]. Available: https://doi.org/10.1109/TSE.2023.3241299

[36] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 246–258. [Online]. Available: https://doi.org/10.1145/3366423.3380111

[37] C. Zhang, Z. Dong, X. Peng, B. Zhang, and M. Chen, "Trace-based multi-dimensional root cause localization of performance issues in microservice systems," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 110:1–110:12. [Online]. Available: https://doi.org/10.1145/3597503.3639088

[38] C. Zhang, T. Jia, G. Shen, P. Zhu, and Y. Li, "Metalog: Generalizable cross-system anomaly detection from logs with meta-learning," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 154:1–154:12. [Online]. Available: https://doi.org/10.1145/3597503.3639205

[39] C. Zhao, M. Ma, Z. Zhong, S. Zhang, Z. Tan, X. Xiong, L. Yu, J. Feng, Y. Sun, Y. Zhang, D. Pei, Q. Lin, and D. Zhang, "Robust multimodal failure detection for microservice systems," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 5639–5649. [Online]. Available: https://doi.org/10.1145/3580305.3599902

[40] L. Tao, S. Zhang, Z. Jia, J. Sun, M. Ma, Z. Li, Y. Sun, C. Yang, Y. Zhang, and D. Pei, "Giving every modality a voice in microservice failure diagnosis via multimodal adaptive optimization," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1107–1119. [Online]. Available: https://doi.org/10.1145/3691620.3695489

[41] (2025). [Online]. Available: https://docs.aiops.cloudwise.com/zh/gaia/

[42] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, p. 243–260, Feb. 2021. [Online]. Available: https://doi.org/10.1109/TSE.2018.2887384

[43] (2025). [Online]. Available: https://chaosblade.io/

[44] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 4739–4745. [Online]. Available: https://doi.org/10.24963/ijcai.2019/658

[45] Y. Sui, Y. Zhang, J. Sun, T. Xu, S. Zhang, Z. Li, Y. Sun, F. Guo, J. Shen, Y. Zhang, D. Pei, X. Yang, and L. Yu, "Logkg: Log failure diagnosis through knowledge graph," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3493–3507, 2023. [Online]. Available: https://doi.org/10.1109/TSC.2023.3293890

[46] C.-M. Lin, C. Chang, W.-Y. Wang, K.-D. Wang, and W.-C. Peng, "Root cause analysis in microservice using neural granger causal discovery," in *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. [Online]. Available: https://doi.org/10.1609/aaai.v38i1.27772

[47] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, M. Vieira, H. Madeira, N. Antunes, and Z. Zheng, Eds. IEEE, 2020, pp. 48–58. [Online]. Available: https://doi.org/10.1109/ISSRE5003.2020.00014

[48] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3087–3098. [Online]. Available: https://doi.org/10.1145/3442381.3449905

[49] R. Ding, C. Zhang, L. Wang, Y. Xu, M. Ma, X. Wu, M. Zhang, Q. Chen, X. Gao, X. Gao, H. Fan, S. Rajmohan, Q. Lin, and D. Zhang, "Tracediag: Adaptive, interpretable, and efficient root cause analysis on large-scale microservice systems," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1762–1773. [Online]. Available: https://doi.org/10.1145/3611643.3613864

[50] T. Zhou, C. Zhang, X. Peng, Z. Yan, P. Li, J. Liang, H. Zheng, W. Zheng, and Y. Deng, "Tracestream: Anomalous service localization based on trace stream clustering with online feedback," in *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023, Florence, Italy, October 9-12, 2023*. IEEE, 2023, pp. 601–611. [Online]. Available: https://doi.org/10.1109/ISSRE59848.2023.00033