# Robust Failure Diagnosis of Microservice System Through Multimodal Data

Shenglin Zhang, *Member, IEEE*, Pengxiang Jin, Zihan Lin, Yongqian Sun, *Member, IEEE*, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, *Member, IEEE*, Wa Jin, Dai Zhang, Zhenyu Zhu, and Dan Pei, *Senior Member, IEEE*

***Abstract*—Automatic failure diagnosis is crucial for large microservice systems. Currently, most failure diagnosis methods rely solely on single-modal data (i.e., using either metrics, logs, or traces). In this study, we conduct an empirical study using real-world failure cases to show that combining these sources of data (multimodal data) leads to a more accurate diagnosis. However, effectively representing these data and addressing imbalanced failures remain challenging. To tackle these issues, we propose *DiagFusion*, a robust failure diagnosis approach that uses multimodal data. It leverages embedding techniques and data augmentation to represent the multimodal data of service instances, combines deployment data and traces to build a dependency graph, and uses a graph neural network to localize the root cause instance and determine the failure type. Our evaluations using real-world datasets show that *DiagFusion* outperforms existing methods in terms of root cause instance localization (improving by 20.9% to 368%) and failure type determination (improving by 11.0% to 169%).**

***Index Terms*—Microservice systems, failure diagnosis, multimodal data, graph neural network.**

Manuscript received 19 February 2023; revised 22 May 2023; accepted 14 June 2023. Date of publication 27 June 2023; date of current version 13 December 2023. This work was supported in part by the Advanced Research Project of China under Grant 31511010501, in part by the National Natural Science Foundation of China under Grants 62272249 and 62072264, and in part by the Natural Science Foundation of Tianjin under Grant 21JCQNJC00180. Recommended for acceptance by T. Batista. *(Corresponding author: Yongqian Sun.)*

Shenglin Zhang is with the College of Software, Nankai University, Tianjin 300071, China, also with the Key Laboratory of Data and Intelligent System Security, Ministry of Education, Tianjin 300071, China, and also with the Haihe Laboratory of Information Technology Application Innovation (HL-IT), Tianjin 300350, China (e-mail: zhangsl@nankai.edu.cn).

Pengxiang Jin, Zihan Lin, Yongqian Sun, Sibo Xia, Zhengdan Li, Zhenyu Zhong, and Wa Jin are with the College of Software, Nankai University, Tianjin 300071, China (e-mail: jinpengxiang@mail.nankai.edu.cn; linzihan@mail.nankai.edu.cn; sunyongqian@nankai.edu.cn; xiasibo@mai.nankai.edu.cn; lzd@nankai.edu.cn; zyzhong@mail.nankai.edu.cn; 1913173@mail.nankai.edu.cn).

Bicheng Zhang is with the School of Computer Science, Fudan University, Shanghai 200437, China (e-mail: 22210240069@m.fudan.edu.cn).

Minghua Ma is with the Microsoft, Beijing 100080, China (e-mail: minghuama@microsoft.com).

Dai Zhang and Zhenyu Zhu are with the ZhejiangE-CommerceBank Co., Ltd., Hangzhou, Zhejiang 310013, China (e-mail: henry.zd@mybank.cn; michael.zzy@mybank.cn).

Dan Pei is with the Department of Computer Science, Tsinghua University, Beijing 100190, China, and also with the Beijing National Research Center for Information Science and Technology, Beijing 100084, China (e-mail: peidan@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TSC.2023.3290018

## I. Introduction

MICROSERVICES architecture is becoming increasingly popular for its reliability and scalability [1]. Typically, it is a large-scale distributed system with dozens to thousands of service instances running on various environments (e.g., physical machines, VMs, or containers) [2]. Due to the complex and dynamic nature of microservice systems, the failure of one service instance can propagate to other service instances, resulting in user dissatisfaction and financial losses for the service provider. For example, Amazon Web Service (AWS) suffered a failure in December 2021 that impacted the whole networking system and took nearly seven hours to diagnose and mitigate [3]. Therefore, it is crucial to timely and accurately diagnose failures in microservice systems.

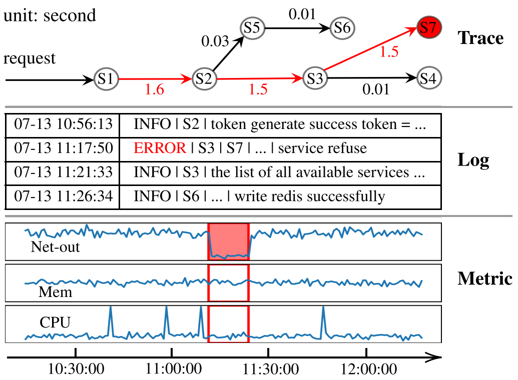
To effectively diagnose failures, microservice system operators typically collect three types of monitoring data: traces, logs, and metrics. Traces are tree-structured data that record the detailed invocation flow of user requests. Logs are semi-structured text that records hardware and software events of a service instance, including business events, state changes, hardware errors, etc. Metrics are time series indicating service status, including system metrics (e.g., CPU utilization, memory utilization) and user-perceived metrics (e.g., average response time, error rate). From now on, we use the term *modality* to describe a particular data type. Fig. 1 shows an example of the three modalities of a microservice system.

Automatic failure diagnosis of microservice systems has been a topic of great interest over the years, particularly when identifying the root cause instance and determining the failure type. Most approaches rely on *single-modal* data, such as traces [1], [4], [5], [6], logs [7], [8], [9], [10], or metrics [11], [12], [13], [14], to capture failure patterns. However, relying solely on single-modal data for diagnosing failures is not effective enough for two reasons. First, a failure can impact multiple aspects of a service instance, causing more than one modality to exhibit abnormal patterns. Using just one data source cannot fully capture these patterns and accurately distinguish between different types of failures. Second, some types of failures may not be reflected in certain modalities, making it difficult for methods relying on that modality to identify these failures.

Moreover, we conduct an empirical study on an open-source dataset to verify the necessity of combining *multimodal* data for robust failure diagnosis. As listed in Table I, the dataset

TABLE I
DETAILED INFORMATION OF THE FAILURES IN THE EMPIRICAL STUDY

| Failure Type | Root Cause | Metric | Log | Trace | # Failures |
|---|---|---|---|---|---|
| System stuck | High memory usage | memory_usage_pct ↑ | - | - | 505 |
| Process crash | Incorrect deallocation | memory_stats_active_anon ↓ | - | - | 16 |
| Login failure | Network interruption | - | ERROR ǀ 0.0.0.1 ǀ 172.17.0.5 ǀ M1 ǀ uuid: 78fef9f0 information has expired, mobile phone login is invalid | S1->S2: RT=11s | 527 |
| File missing | Code bug | - | ERROR ǀ 0.0.0.3 ǀ W2 ǀ get an error [Errno 2] No such file or directory: 'resources/source_file/source_file.csv' | S2->S3: RT=1.5s | 36 |
| Access denied | Misconfiguration | - | ERROR ǀ 0.0.0.2 ǀ B2 ǀ 2768e0e0037e ǀ service refuse | S2->S4: RT=1.1s | 15 |



```
07-13 10:56:13 | INFO ǀ S2 ǀ token generate success token = ...
07-13 11:17:50 | ERROR ǀ S3 ǀ S7 ǀ ... ǀ service refuse
07-13 11:21:33 | INFO ǀ S3 ǀ the list of all available services ...
07-13 11:26:34 | INFO ǀ S6 ǀ ... ǀ write redis successfully
```

Fig. 1. Multimodal data of a microservice system. S1–S7 are different microservices.

contains failures caused by various reasons: high memory usage, incorrect deallocation, code bug, misconfiguration, network interruption, etc. We examine hundreds of service instance failures and conclude that combining traces, logs, and metrics (*multimodal*) is crucial for accurate diagnosis. For example, the microservice shown in Fig. 1 is experiencing a failure due to missing files. It generated error messages in logs and a significant increase in status code 500 in related traces. Additionally, one of its metrics, network out bytes, dropped dramatically during this failure.

These observations highlight the importance of incorporating multimodal data for robust failure diagnosis. However, combining multimodal data for diagnosing failures in microservice systems faces two major challenges:

1) *Representation of multimodal data:* The formats of metrics, logs, and traces are significantly different from each other. Service instance metrics are often in the form of time series (the bottom of Fig. 1), while logs are usually semi-structured text (the middle of Fig. 1) and traces often take the form of tree structures with spans as nodes (the top of Fig. 1). It is challenging to find a unified representation of all this multimodal data that fully utilizes complementary information from each data type.
2) *Imbalanced failure types:* Fault tolerance mechanisms in microservice systems often result in a high ratio of normal data to failure-related data. Some types of failures are much rarer than others, leading to an imbalance in the ratio of different types of failures (Table I).

To tackle the above challenges, we present *DiagFusion*, an automated failure diagnosis approach that integrates trace, log, and metric data. To form a unified representation of the three modalities with different formats and natures, *DiagFusion* combines lightweight preprocessing and representation learning, which maps data from different modalities into the same vector space. Since the labeled failures are usually inadequate to train the representation model effectively, we propose a data augmentation mechanism, which helps *DiagFusion* to learn the correlation between the three modalities and failures effectively. To further enhance the accuracy of our diagnosis, *DiagFusion* uses historical failure patterns to train a Graph Neural Network (GNN), capturing both spatial features and possible failure propagation paths, which allows *DiagFusion* to conduct root cause instance localization and failure type determination.

Our contributions are summarized as follows:

- We propose *DiagFusion*, a multimodal data-based approach for failure diagnosis (Section IV). *DiagFusion* builds a dependency graph from trace and deployment data to capture possible failure propagation paths. Then it applies a GNN to achieve a two-fold failure diagnosis, i.e., root cause instance localization and failure type determination. To the best of our knowledge, we are among the first to learn a unified representation of the three modalities for the failure diagnosis of microservice systems (i.e., trace, log, and metric).
- We leverage data augmentation to improve the quality of the learned representation, which allows *DiagFusion* to work with limited labeled failures and imbalanced failure types.
- We conduct extensive experiments on two datasets, one from an open-source platform and another from a real-world microservice system (Section V). The results show that when *DiagFusion* is trained on 160 and 80 cases, it achieves Avg@5 of 0.75 and 0.76 on the two datasets, respectively, improving the accuracy of *root cause instance localization* by 20.9% to 368%. Moreover, *DiagFusion* achieves F1-score of 0.84 and 0.80, improving the accuracy of *failure type determination* by 11.0% to 169%.

Our implementation of *DiagFusion* is publicly available.[1]

The rest of the paper is organized as follows: Section II introduces the necessary background. Section III presents the results of an empirical study of failures in microservice systems. Section IV describes the overview and detailed implementation of *DiagFusion* in failure diagnosis. In Section V, we evaluate the performance and time efficiency of *DiagFusion* using two

[1] https://anonymous.4open.science/r/DiagFusion-378D

datasets. Section VI discusses the technical rationale, robustness, and threats to validity. Section VII presents the related work in failure diagnosis. Section VIII concludes the paper.

## II. Background

### A. Microservice Systems and Multimodal Data

Microservice systems allow developers to independently develop and deploy functional software units (microservice). For example, when a user tries to buy an item on an online shopping website, the user will experience item searching, item displaying, order generation, payment, etc. Each of these functions is served by a specific microservice. A failure at a specific service instance can propagate to other service instances in many ways, bringing cascading failures. However, diagnosing online failures in microservice systems is difficult due to these systems' highly complex orchestration and dynamic interaction. To accurately find the cause of a failure, operators must carefully monitor the system and record traces, logs, and metrics. These three modalities of monitoring data stand as the three pillars of the observability of microservice systems. The collection and storage of instances' monitoring data are not in the scope of this paper. The three modalities: trace, log, and metric, and their roles in failure diagnosis are described below.

*Trace:* Traces record the execution paths of users' requests. Fig. 1 shows an example of trace at the top. Google formally proposed the concept of traces at Dapper [15], in which it defined the whole lifecycle of a request as a *trace* and the invocation and answering of a component as a *span*. By examining traces, operators may identify microservices that have possibly gone wrong [4], [6], [16], [17], [18], [19], [20], [21]. Traces can be viewed as trees, with microservices as nodes and invocations as edges. Each subtree corresponds to a span. Typically, traces carry information about invocations, e.g., start time, caller, callee, response time, and status code.

*Log:* Logs record comprehensive events of a service instance. Some examples of logs are shown in the middle of Fig. 1. Logs are generated by developers using commands like *printf*, *logging.debug*, *logging.error*. They provide an internal picture of a service instance. By examining logs, operators may discover the actual cause of why an instance performs not well. Typically, logs consist of three fields: timestamp, verbosity level, and raw message [22]. Four commonly used verbosity levels, i.e., INFO, WARN, DEBUG, and ERROR, indicate the severity of a log message. The raw message of a log conveys detailed information about the event. To utilize logs more effectively, researchers have proposed various parsing techniques to extract templates and parameters, e.g., FT-Tree [23], Drain [22], POP [24], MoLFI [25], Spell [26], and Logram [27].

*Metric:* Various system-level metrics (e.g., CPU utilization, memory utilization) and user-perceived metrics (e.g., average response time) are configured for monitoring system instances. Each metric is collected at a predefined interval, forming a time series, as shown at the bottom of Fig. 1. These metrics track various aspects of performance issues. By examining metrics, operators can determine which physical resource is anomalous or is the bottleneck [28], [29], [30], [31], [32], [33].

In addition to trace, log, and metric, *deployment data* is also important to failure diagnosis. A microservice system comprises many hardware and software assets that form complicated interrelationships. Operators must carefully record these relationships (*a.k.a.* deployment data) to keep high maintainability of the system. Leveraging deployment data enables the understanding of failure propagation paths and characteristics.

### B. Preliminaries

*Representation Learning:* Representation learning has been widely used in natural language processing tasks, usually in the form of word embedding. Popular techniques of representation learning include static representation like word2vec [34], GloVe [35], fastText [36], and dynamic representation like ELMo [37], BERT [38], GPT [39]. With the similarities between logs and natural languages, representation learning can be applied to extract log features [40]. We employ fastText to learn a unified representation of events from multimodal data. Compared to word2vec and GloVe, fastText can utilize more information [36]. We employ fastText to learn a unified representation of the multimodal data.

In essence, fastText is a neural network model that processes words as input and takes the output from the hidden layer (a vector of real numbers) as its representation. It can be trained in both supervised and unsupervised modes, but the supervised mode generally yields more accurate results due to its incorporation of label information. In the supervised training mode, the neural network is optimized by predicting the class of the document. Once the training is completed, fastText can be used to provide vectorized representations (i.e., embeddings) for any given input.

*Graph Neural Network:* GNN can effectively model data from non-euclidean space, thereby being popular among fields with graph structures, e.g., social networks, biology, and recommendation systems. Popular GNN architecture includes Graph Convolution Network (GCN) [41], GraphSAGE [42], Graph Attention Network (GAT) [43], etc. GNNs apply graph convolutions, allowing nodes to utilize their information and learn from their neighbors through message passing. There are numerous components in microservice systems that interconnect with each other. Thus graph structure is suitable to model microservice systems, and we employ GNN to learn the propagation patterns of historical failure cases.

### C. Problem Statement

When a failure occurs, operators need to localize the root cause instance and determine what has happened to it to achieve timely failure mitigation. For large-scale microservice systems, the first task is a ranking problem: to rank the root cause instance higher than other instances. We use the term *root cause instance localization* to name this task (Task #1). The second task is a classification problem: to classify the failure into a predefined set of failure types. We use the term *failure type determination* to name this task (Task #2).

After each failure, operators will carefully conduct a post-failure analysis: labeling its root cause instance and its failure type. Additionally, chaos engineering can generate a large number of failure cases [44]. It can enlarge the number of failure cases and enrich the types of failures. We train *DiagFusion* based on these failure cases.

## III. Empirical Study

Most existing failure diagnosis methods are based on single-modal data. However, these methods cannot fully capture the patterns of failed instances, leading to ineffective failure diagnosis. We conduct an empirical study conducted on Generic AIOps Atlas (GAIA)[2] dataset to show the ineffectiveness of these methods. The dataset is collected from a simulation environment consisting of 10 microservices, two database services (MySQL and Redis), and five host machines. The system serves mobile users and PC users. Operators injected five types of failures, including system failures (System stuck and Process crash) and service failures (Login failure, File missing, and Access denied). The failure injection record is provided along with the data. Table I lists some typical symptoms of failures. We can see that no modality alone can distinguish the patterns of these five types of failures. It also shows that traces, logs, and metrics may display different anomalous patterns when a failure occurs. Mining the correlation between multimodal data can provide operators with a more comprehensive understanding of failures.

Besides, Table I shows that some failures occur much more frequently than others. For example, the total occurrences of *Process crash*, *File missing*, and *Access denied* (67) equals only 12% of the occurrences of *Login failure* (527).

To further understand the distribution of failure types in the production environment, we investigated $N$ failures in a microservice system of Microsoft. Due to the company policy, we have to hide some details of these failures. The failures of the studied system are recorded in the Incident Management System (IcM) of Microsoft, where a failure is centralized handled, including the detection, discussion, mitigation, and post-failure analysis of failures. The IcM data of failures are persistently stored in a database. We query the failure records from the database within the time range from 2021 August to 2022 August. We only keep the failures with the status of "completed", for their post-failure analyses have been reviewed. In the *root cause* field of post-failure analysis, operators categorize the failures into the following types: code, data, network, hardware, and external. We can see from Fig. 2 that different failure types are imbalanced regarding the number of failure cases. The imbalanced data poses a significant challenge because most machine learning methods perform poorly on failure types with fewer occurrences.

## IV. Approach

### A. Design Overview

In this article, we propose *DiagFusion*, which combines the modality of trace, log, and metric for accurate failure diagnosis. The training framework of *DiagFusion* is summarized in Fig. 3. First, *DiagFusion* extracts events from raw traces, logs, and metrics data and serializes them by their timestamps. Then, we train a neural network to learn the distributed representation of events by encoding events into vectors. The challenge of data imbalance is overcome through data augmentation during model training. We unify three modalities with different natures by turning unstructured raw data into structured events and vectors. Then we combine traces with deployment data to build a dependency graph (DG) of the microservice system. After that, the representations of events and DG are glued together by a GNN. We train GNN using historical failures to learn the propagation pattern of system failures.

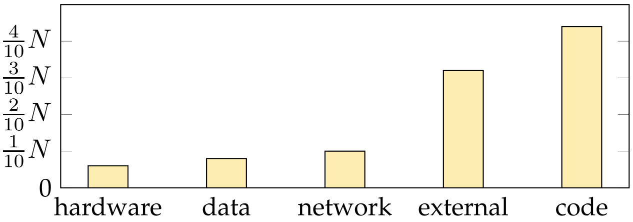


Fig. 2. The distribution of failure types at a large-scale real-world microservice system.

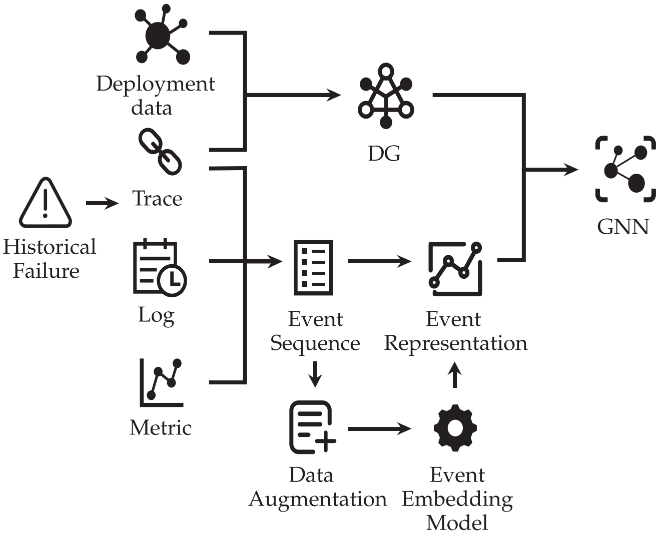


Fig. 3. The training framework of *DiagFusion*.

After the training stage, we save the event embedding model and the GNN. Fig. 5 depicts the real-time failure diagnosis framework of *DiagFusion*. The trigger of *DiagFusion* can be alerts generated through predefined rules. When a new failure is alerted, *DiagFusion* will perform a real-time diagnosis and give the results back to operators.

### B. Unified Event Representation

*DiagFusion* unifies the three modalities by extracting events from the raw data and encoding them into vectors. Specifically, it collects failure-indicative events by leveraging effective and lightweight methods, including anomaly detection techniques for metrics and traces and template parsing techniques for logs.

[2] https://github.com/CloudWise-OpenSource/GAIA-DataSet

Then, it trains a fastText [36] model on event sequences to generate embedding vectors of events.

First, we introduce the *instances* in a microservice system. Microservice systems have the advantage of dynamic deployment by utilizing the container technique. In this paper, we use the term *instance* to describe a running container and the term *service group* to describe the logical component that an instance belongs to. For example, *Billing* is a service group in a microservice system, and *Billing_cff19b* denotes an instance, where *cff19b* is the container id. Below we will describe the event extraction from different modalities.

*Trace Event Extraction:* Traces record calling relationships between services. We group trace data by its caller and callee services. *DiagFusion* will examine multiple fields inside a trace group. Under different implementations of trace recording, trace data can carry different fields, e.g., response time and status code, which reflect different aspects of operators' interests. We apply an anomaly detection algorithm (i.e., 3-sigma) for numerical fields like response time to detect anomalous behaviors. For categorical fields like status code, we count the number of occurrences of each value. If the count of some value increases dramatically, we determine that this field is anomalous. We determine that a group of caller and callee is anomalous if one of its fields becomes anomalous. The extracted trace events are in the form of tuple <*timestamp, caller-instance-id, callee-instance-id*>.

*Log Event Extraction:* Logs record detailed activities of an instance (service or machine). We perform log parsing for log event extraction using Drain [22], which has been proven to be effective in practice. Drain uses a fixed depth parse tree to distinguish the template part and the variable part of log messages. For example, in the log message "uuid: 8fef9f0 information has expired, mobile phone login is invalid", "uuid: ****** information has expired, mobile phone login is invalid" is the template part, and "8fef9f0" is the variable part. After we get the template part of a log message, we hash the string of the template part to obtain an event template id. The extracted log events are in the form of tuple <*timestamp, instance-id, event-template-id*>.

*Metric Event Extraction:* Metrics are also recorded at the instance level. We perform 3-sigma to detect anomalous metrics. When the value of a metric exceeds the upper bound of 3-sigma, the anomaly direction is *up*. Similarly, the anomaly direction is *down* if the value is below the lower bound. The extracted metric events are in the form of tuple <*timestamp, instance-id, metric-name, anomaly-direction*>.

The above extraction provides events from different modalities. Despite the differences in raw data, all extracted events share two fields, namely *timestamp* and *instance-id*. These are the keys to unifying different modalities. We group events by *instance-id* and serialize events in the same group by *timestamp*. Fig. 4 shows the event extraction and serialization process for one instance. The event sequence of instance $i$ is denoted by $E_i$.

After getting the event sequence of every instance, we further assign labels to every event sequence according to operators' post-failure analysis. Original failure labels are often in the form of tuple <*root cause instance-id, failure type*>. To fully utilize the label information, we relabel event sequences in an instance-wise manner. Specifically, the root cause instance's event sequence is labeled by the actual failure type, while other instances' event sequences are labeled as "non-root-cause". A microservice system with $p$ historical failures and $q$ instances results in $N = p \times q$ event sequences after relabeling. Then, we learn unified representations from these relabeled historical event sequences using the event embedding model.

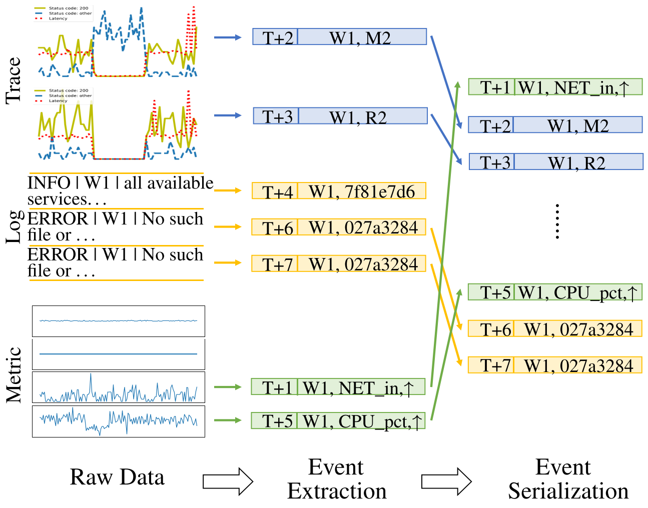


Fig. 4. The event extraction and serialization process using traces, logs, and metrics.

With event sequence and instance labeling, we can transform events into vectors. We use the term *event embedding* to describe the mapping of events to real number vectors. Specifically, we train a fastText model on the event sequences to obtain the vectorized representation for events from all three modalities. FastText is a neural network originally proposed for text classification. For a document with word sequences, fastText extracts $n$-grams from it and predicts its label. In our scenario, we replace word sequences with event sequences and replace document labels with failure types. The training of fastText minimizes the negative log-likelihood over classes:

$$\min_{f} -\frac{1}{N} \sum_{n=1}^{N} y_n \log \left( f \left( x_n \right) \right) \tag{1}$$

where $x_n$ is the normalized bag of features of the $n$-th event sequence, $y_n$ denotes the relabeled information, and $f$ is the neural network. We treat fastText's output as the vectorized representation of events. The training detail of the event embedding model is described in Section IV-D.

### C. Graph Neural Network

In the event representation process, *DiagFusion* captures the local features of instances. However, failures can propagate between instances, so we need to have a global picture of the system, i.e., how a failure will affect the system. To this end, we employ a GNN to learn the failure propagation between service instances and integrate all the information of the whole system.

To leverage a GNN, it is essential to consider both *nodes* and *edges* within a graph. The *nodes* in a GNN corresponds to the instances in a microservice system. An instance is characterized by its anomalous events in *DiagFusion*. We represent an instance $i$ by averaging all of its events:

$$h_i^{(0)} = \frac{1}{|E_i|} \sum_{\forall e \in E_i} \mathcal{V}_1(e) \tag{2}$$

where $E_i$ is the extracted event sequences, and $\mathcal{V}_1(e)$ is the vectorized representation of event $e$ learned by the event embedding model.

The *edges* in a GNN correspond to the dependency graph in a microservice system. There are two dominant ways of propagation failure between services: function calling or resource contention [45]. So we combine traces and deployment data to capture probable failure propagation paths. Specifically, we aggregate traces to get a call graph. Then we add two directed edges for each pair of caller and callee, with one pointing from the caller to the callee and the other in the reverse direction. From deployment data, we add edges between two instances if they are co-deployed, i.e., sharing resources.

After obtaining the dependency graph and instance representations, we employ GNN to learn the failure propagation pattern by its message-passing mechanism. At the $K$-th layer of GNN, we apply topology adaptive graph convolution [46] and update the internal data of instances according to:

$$H^K = \sum_{k=0}^{K} \left( D^{-1/2} A D^{-1/2} \right)^k X \Theta_k \tag{3}$$

where $A$ denotes the adjacency matrix, $D_{ii} = \sum_{j=0} A_{ij}$ is a diagonal degree matrix, $\Theta_k$ denotes the linear weights to sum the results of different hops together.

Finally, we add a MaxPooling layer as the readout layer to integrate the information of the whole microservice system. Following the MaxPooling layer, there is a fully connected layer where each neuron corresponds to either a service group with possible root cause instances for task #1 or a failure type for task #2.

### D. *Training of* DiagFusion

*DiagFusion* applies a two-phase training strategy to learn the failure pattern of a microservice system. First, it trained the event embedding model with data augmentation. Then it trains the GNN with a joint learning technique.

*1) Training of Event Embedding Model: DiagFusion* employs a data augmentation strategy to enrich the training dataset and reduce the model's bias towards the majority class. First, we train our event embedding model on the original data. The trained neural network, denoted by $f_0$, maps events to the vector space $\mathcal{V}_0$. To increase the number of failure cases, we add new event sequences for each failure type (including "non-root-cause") by randomly taking an event sequence of that type and replacing one of the events with its closest neighbor (determined by euclidean distance) in $\mathcal{V}_0$. After all failure types are expanded to a relatively large size, e.g., 1000, we can obtain a more balanced training set. Further details on the choice of the expanding size can be found at Section V-E. Then we train the event embedding model again ($f_1$) on the expanded data and regard the representations generated in this round ($\mathcal{V}_1$) as the final unified event representations.

*2) Training of Graph Neural Network:* We train the GNN in a joint learning fashion to fully utilize the shared information between tasks #1 and #2. Then we combine the trained GNN with a ranking strategy to better fit the nature of microservice systems.

*Ranking Strategy:* One of the advantages of microservice systems is that the architecture allows dynamic deployment of service instances. Thus, service instances are constantly being created and destroyed. However, when it comes to failure diagnosis, this kind of flexibility raises a challenge for learning-based methods. The failure diagnosis model will have to be retrained frequently if the output layer directly outputs the probability of being the root cause instance for each instance since many instances can be created or destroyed after the model training is finished. We add an extract step in *DiagFusion* to overcome this challenge. Instead of directly determining the root cause instance, *DiagFusion* is trained on service groups, the logical aggregation of service instances, for task #1. Then *DiagFusion* ranks the instances inside a candidate service group by the length of their event sequences. The instance with more anomaly events will be ranked higher and likely be the root cause instance.

*Joint Learning:* Intuitively, the two tasks of failure diagnosis, i.e., root cause instance localization and failure type determination, share some knowledge in common. For a given failure, the only difference between task #1 and task #2 lies in their labels. So *DiagFusion* integrates a joint learning mechanism to utilize the shared knowledge and reduce the training time. (Training two models separately requires twice the time otherwise.) Specifically, the joint loss function is:

$$-\frac{1}{F} \sum_{i=1}^{F} \left( \sum_{j=1}^{S} y(s)_{i,j} \log p(s)_{i,j} + \sum_{k=1}^{T} y(t)_{i,k} \log p(t)_{i,k} \right) \tag{4}$$

where $F$ is the number of historical failures, $S$ is the number of service groups, $T$ is the number of failure types, $y(s)$ is the root cause service group labeled by operators, $y(t)$ is the failure type, $p(s)$ is the predicted service group, and $p(t)$ is the predicted failure type.

### E. *Real-Time Failure Diagnosis*

After the training stage, we save the trained event embedding model and the GNN. When a new failure is alerted, *DiagFusion* performs a real-time diagnosis process as shown in Fig. 5.

*1) Running Example:* Fig. 6 shows how *DiagFusion* can be integrated with microservice systems. To better explain how *DiagFusion* diagnoses failure, we demonstrate the workflow of *DiagFusion* using one real-world failure from D1. At 10:46, service instance B1 encounters a failure of access denied. Fig. 7 shows the original data, event sequence, and the DG.

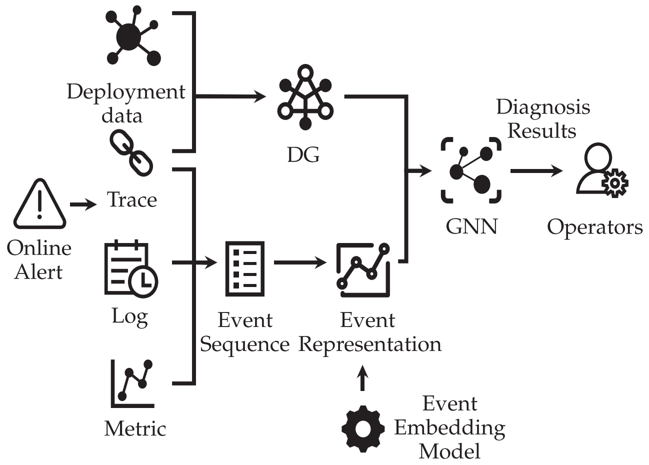


Fig. 5. Real-time failure diagnosis.

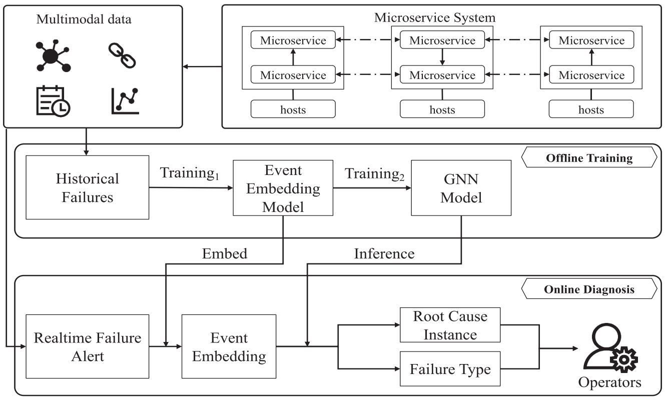


Fig. 6. Integration of *DiagFusion* with a microservice system.

From Fig. 7(a), we can see that failure-indicative events from different modalities are temporally intertwined. Then the GNN predicts service group “B” and failure type “access denied”. Further ranking within the service group “B” gives “B1” as the Top1 instance. The overall process takes less than 10 seconds. Thus, *DiagFusion* effectively addresses tasks #1 and #2.

## V. Evaluation

In this section, we evaluate the performance of *DiagFusion* using two real-world datasets. We aim to answer the following research questions (RQs):

*RQ1*: How effective is *DiagFusion* in failure diagnosis?

*RQ2*: Does each component of *DiagFusion* have significant contributions to *DiagFusion*’s performance?

*RQ3*: Is the computational efficiency of *DiagFusion* sufficient for failure diagnosis in the real world?

*RQ4*: What is the impact of different hyperparameters?

### A. Experimental Setup

*1) Dataset:* To evaluate the performance of *DiagFusion*, we conduct extensive experiments on two datasets collected from two microservice systems under different business backgrounds and architectures, D1 and D2. To prevent data leakage, we split the data of D1 and D2 into training and testing sets according to their start time, i.e., we use data from the earlier time as the training set and data from the later time as the test set. Detailed information is listed in Table II. The systems that produce D1 and D2 are as follows:

TABLE II
Detailed Information of Datasets

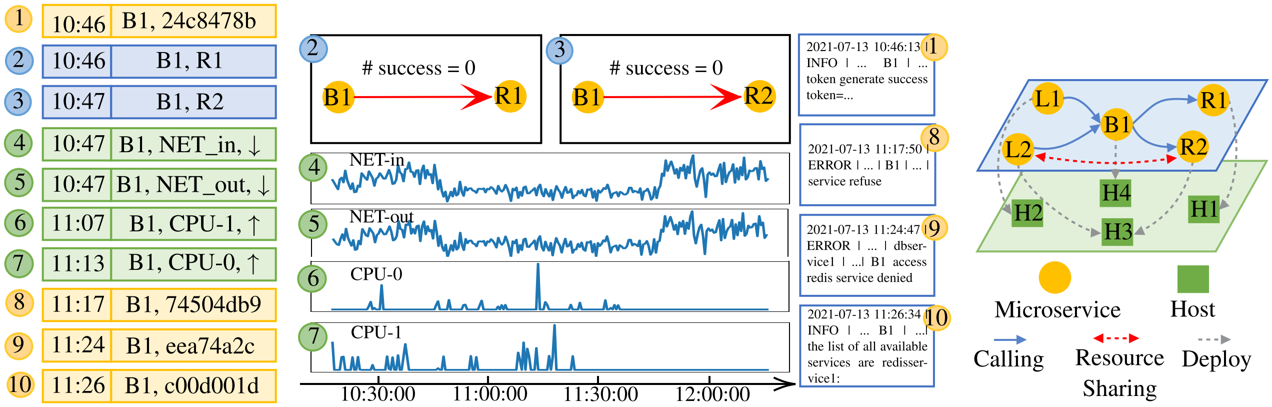
| Dataset | # Instances | # Training | # Test | # Records | |
|---|---|---|---|---|---|
| D1 | 17 | 160 | 939 | trace<br>log<br>metric | 2,321,280<br>87,974,577<br>56,684,196 |
| D2 | 18 | 80 | 79 | trace<br>log<br>metric | 1,123,200<br>21,356,923<br>8,228,010 |

1) D1. The details of D1 are elaborated in Section III.
2) D2. The second dataset is collected from the management system of a top-tier commercial bank. The studied system consists of 14 instances, including microservices, web servers, application servers, databases, and dockers. Due to the non-disclosure agreement, we cannot make this dataset publicly available. Two experienced operators examined the failure records from January 2021 to June 2021. They classified the failures into five types of failures, i.e., CPU-related failures, memory-related failures, JVM-CPU-related failures, JVM-memory-related failures, and IO-related failures. The classification was done separately, and they checked the labeling with each other to reach a consensus.

*2) Baseline Methods:* We select six advanced single-modal-based methods (two for trace (i.e., MicroHECL [5], MicroRank [6]), two for log (i.e., Cloud19 [8], LogCluster [7]), and two for metric (i.e., AutoMAP [13], MS-Rank [12])), and two multimodal-based methods (i.e., PDiagnose [47], CloudRCA [48]) as the baseline methods. More details can be found in Section VII. Among the baseline methods, Cloud19, LogCluster, and CloudRCA cannot address Task #1 (root cause instance localization), while MicroHECL, MicroRank, AutoMAP, MS-Rank, and PDiagnose cannot address Task #2 (failure type determination). Therefore, we divide the baseline methods into two groups to evaluate the performance of Task #1 and Task #2, respectively: MicroHECL, MicroRank, AutoMAP, MS-Rank, and PDiagnose for Task #1, Cloud19, LogCluster, and CloudRCA for Task #2.

We configure the parameters of all these methods according to their papers. Specifically, we use the same configuration for parameter settings explicitly mentioned in the papers and not limited to a particular dataset (e.g., significance level, feature dimension). For parameter settings that apply to a particular dataset (e.g., window length, period), we adapt them according to the range the papers provide or to our data.

*3) Evaluation Metrics:* As stated in Section II-C, *DiagFusion* aims to localize the root cause instance and determine the failure type. We carefully select different evaluation metrics for



| | | |
|---|---|---|
| 1 | 10:46 | B1, 24c8478b |
| 2 | 10:46 | B1, R1 |
| 3 | 10:47 | B1, R2 |
| 4 | 10:47 | B1, NET_in, ↓ |
| 5 | 10:47 | B1, NET_out, ↓ |
| 6 | 11:07 | B1, CPU-1, ↑ |
| 7 | 11:13 | B1, CPU-0, ↑ |
| 8 | 11:17 | B1, 74504db9 |
| 9 | 11:24 | B1, eea74a2c |
| 10 | 11:26 | B1, c00d001d |

```
2021-07-13 10:46:13 | INFO | ... B1 | ... token generate success token=...
```

```
2021-07-13 11:17:50 | ERROR | ... | B1 | ... | service refuse
```

```
2021-07-13 11:24:47 | ERROR | ... | dbservice1 | ...| B1 access redis service denied
```

```
2021-07-13 11:26:34 | INFO | ... B1 | ...| the list of all available services are redisservice1:
```

(a) (b) (c)

Fig. 7. A running example of *DiagFusion*. (a) the serialized multimodal event sequence of the root cause instance (B1); (b) the original data corresponding to the event sequence; (c) part of the dependency graph in this failure.

both tasks to better reflect the real-world performance of all selected methods.

For Task #1, we use *Top-k accuracy* (A@k) and *Top-5 average accuracy* (Avg@5) as the evaluation metrics. A@k is a well-adopted metric that quantifies the probability that top-k instances output by each method indeed contain the root cause instance [5]. Formally, given $|A|$ as the test set of failures, $RC_i$ as the ground truth root cause instance, $RC_s[k]$ as the top-k root cause instances set generated by a method, A@k is defined as:

$$A@k = \frac{1}{|A|}\sum_{a\in A}\begin{cases}1, & \text{if } RC_{ia} \in RC_{sa}\left[k\right]\\ 0, & \text{otherwise}\end{cases} \tag{5}$$

Avg@5 is another popular metric that evaluates a method's overall capability of localizing the root cause instance[49]. In practice, operators often examine the top 5 results. Avg@5 is calculated by:

$$Avg@5 = \frac{1}{5}\sum_{1\leq k\leq 5} A@k \tag{6}$$

For Task #2, which is a multi-class classification problem, we use the weighted average *precision*, *recall*, and *F1-score* to test the performances. These metrics have been selected based on a previous study [50] as a reliable way to assess the model's effectiveness in this specific context. With True Positives (TP), False Positives (FP), and False Negatives (FN), the calculation is given by F1-score $= 2 \times \frac{precision \times recall}{precision + recall}$, where $precision = \frac{\text{TP}}{\text{TP+FP}}$ and $recall = \frac{\text{TP}}{\text{TP+FN}}$.

*4) Implementation:* We implement *DiagFusion* and baselines with Python 3.7.13, PyTorch 1.10.0, scikit-learn 1.0.2, fastText 0.9.2, and DGL 0.9.0. We run the experiments on a server with 12 × Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz and 128 G RAM (without GPUs). We repeat every experiment five times and take the average result to reduce the effect of randomness.

### B. Overall Performance (RQ1)

To demonstrate the effectiveness of *DiagFusion*, we compare it with the baseline methods on Task #1 and Task #2.

TABLE III
Effectiveness of Failure Type Determination (Task #2)

| Method | D1 | | | D2 | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F1-score | Precision | Recall | F1-score |
| *DiagFusion* | **0.860** | **0.829** | **0.839** | **0.822** | **0.797** | **0.800** |
| Cloud19 | 0.774 | 0.774 | 0.756 | 0.526 | 0.278 | 0.297 |
| LogCluster | 0.615 | 0.477 | 0.336 | 0.521 | 0.722 | 0.605 |
| CloudRCA | 0.436 | 0.453 | 0.357 | 0.589 | 0.506 | 0.538 |

The comparison result of Task #1 is shown in Fig. 8. *DiagFusion* achieves the best performance. Specifically, the A@1 to A@5 of *DiagFusion* are almost the best on D1 and D2. More specifically, the Avg@5 of *DiagFusion* exceeds 0.75 on both D1 and D2, respectively. It is at least 0.13 higher on both datasets than baselines using single-modal data due to the advantage of using multimodal data. Compared with PDiagnose, which also uses multimodal data, the Avg@5 of *DiagFusion* is higher by at least 0.18. This indicates that learning from historical failures improves the accuracy of diagnosis significantly.

The result of Task #2 is shown in Table III. For this task, *DiagFusion* is better than almost all baselines. On D1, the precision, recall, and F1-score of *DiagFusion* are over 0.80. On D2, *DiagFusion* manages to maintain an F1-score of 0.80, which is at least 0.195 higher than the baselines. Considering both systems and tasks, DiagFusion consistently demonstrates superior performance, thereby substantiating its effectiveness.

### C. Ablation Study (RQ2)

To evaluate the effects of the three key technique contributions of *DiagFusion*: 1) data augmentation; 2) fastText embedding; 3) DG and GNN, we create five variants of *DiagFusion*. *C1:* Remove the data augmentation. *C2:* Use word2vec embedding instead of fastText. *C3:* Use GloVe embedding instead of fastText. *C4:* Replace the GNN output layer with a decision tree. *C5:* Replace the GNN output layer with a kNN model.

Table IV lists that *DiagFusion* outperforms all the variants on D1 and D2, demonstrating each component's significance. When removing the data augmentation (*C1*), the performance

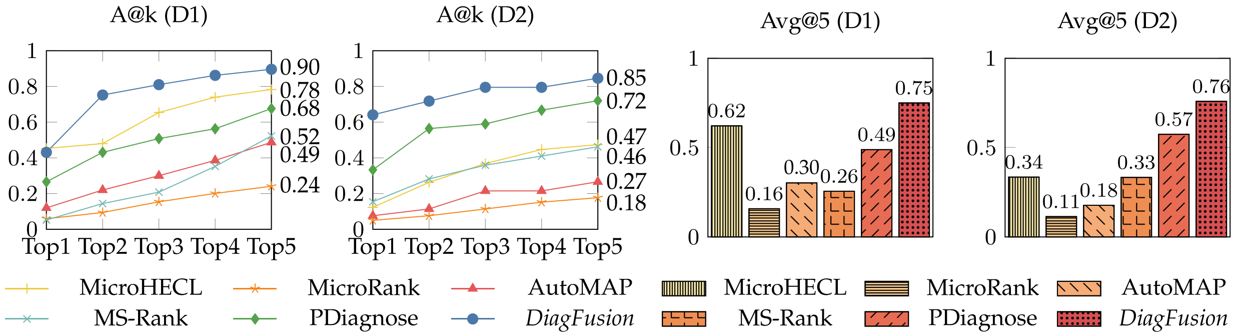


Fig. 8. Effectiveness of root cause instance localization (Task #1).

TABLE IV
CONTRIBUTIONS OF COMPONENTS

| | Method | Task #1 | | | | Task #2 | | |
|---|---|---|---|---|---|---|---|---|
| | | A@1 | A@3 | A@5 | Avg@5 | Precision | Recall | F1-score |
| D1 | *DiagFusion* | **0.419** | **0.813** | **0.914** | **0.750** | **0.860** | **0.829** | **0.839** |
| | C1 | 0.341 | 0.678 | 0.833 | 0.641 | 0.809 | 0.793 | 0.779 |
| | C2 | 0.306 | 0.639 | 0.780 | 0.594 | 0.780 | 0.765 | 0.768 |
| | C3 | 0.309 | 0.632 | 0.770 | 0.588 | 0.773 | 0.797 | 0.781 |
| | C4 | 0.359 | 0.657 | 0.760 | 0.616 | 0.351 | 0.102 | 0.104 |
| | C5 | 0.419 | 0.809 | 0.905 | 0.744 | 0.089 | 0.102 | 0.095 |
| D2 | *DiagFusion* | 0.646 | **0.848** | **0.873** | **0.790** | **0.822** | **0.797** | **0.800** |
| | C1 | 0.304 | 0.506 | 0.646 | 0.471 | 0.567 | 0.608 | 0.576 |
| | C2 | 0.646 | 0.823 | 0.861 | 0.780 | 0.793 | 0.734 | 0.753 |
| | C3 | **0.671** | 0.823 | 0.848 | 0.785 | 0.787 | 0.747 | 0.747 |
| | C4 | 0.494 | 0.620 | 0.646 | 0.587 | 0.780 | 0.595 | 0.639 |
| | C5 | 0.582 | 0.709 | 0.709 | 0.671 | 0.778 | 0.797 | 0.764 |

reduces across the board as models trained from imbalanced data are more likely to bias predictions toward classes with more samples. Data augmentation can alleviate this problem. The performance becomes worse when replacing fastText embedding strategy (*C2 & C3*). The reason is that fastText can learn from operators' failure labeling as well as co-occur relationships, while word2vec and GloVe can only learn from the co-occur relationships between events. Replacing the GNN output layer with classifiers such as decision trees and kNN (*C4 & C5*) degrades performance because the GNN can capture the interaction patterns and fault propagation among instances in microservice systems, but traditional classifiers cannot understand the graph structure information.

### D. Efficiency (RQ3)

We record the running time of all methods and compare them in Table V. The offline training time of *DiagFusion* is acceptable, particularly when considering its infrequent need for retraining. It shows that *DiagFusion* can diagnose one failure within 12 seconds on average online, which means it can achieve quasi-real-time diagnosis because the interval of data collection in D1 and D2 is at least 30 seconds. Although *DiagFusion* may not possess apparent advantages among the methods in Table V, *DiagFusion* can meet the needs of online diagnosis.

TABLE V
THE COMPARISON OF TRAINING TIME (OFFLINE) AND DIAGNOSIS TIME (ONLINE) PER CASE ("-" MEANS NO NEED TRAINING)

| Method | D1 | | D2 | |
|---|---|---|---|---|
| | Offline | Online | Offline | Online |
| *DiagFusion* | 11.02 | 10.95 | 3.59 | 3.26 |
| MicroHECL | - | 65.98 | - | 28.40 |
| MicroRank | 22.9 | 34.47 | 53.2 | 54.94 |
| Cloud19 | 0.41 | 0.03 | 0.03 | 0.03 |
| LogCluster | <0.1 | <0.01 | 0.2 | <0.01 |
| AutoMap | - | 0.299 | - | 0.511 |
| MS-Rank | - | 1.14 | - | 12.94 |
| PDiagnose | - | 42.51 | - | 68.74 |
| CloudRCA | 1.43 | 0.06 | 0.83 | 0.07 |

### E. Hyperparameter Sensitivity (RQ4)

We discuss the effect of four hyperparameters of *DiagFusion*. Fig. 9 shows how Avg@5 (Task #1), F1-score (Task #2) change with different hyperparameters.

*Embedding Dimension:* The performance of *DiagFusion* reacts differently on different datasets in terms of sensitivity to dimensionality (D1 remains stable while D2 fluctuates more), and the optimal dimensionality is inconsistent across datasets and tasks. We choose the 100 dimensions in our experiments because it has the best overall performance.

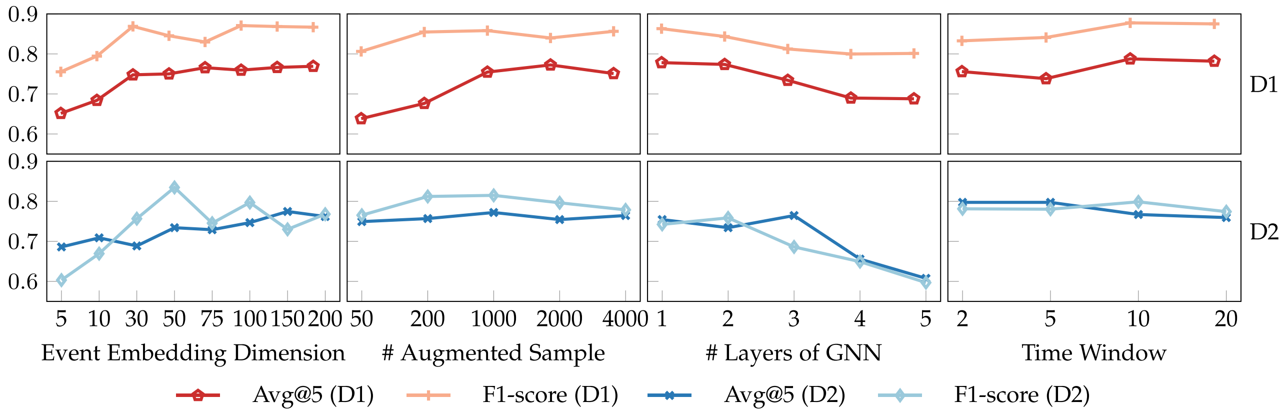


Fig. 9. The effectiveness of *DiagFusion* under different hyperparameters.

*The Number of Augmented Samples:* The experiments in Section V-B show that data augmentation has some improvement in the model's performance. However, when the number of samples increases to a certain amount, the information in the training set has already been fully utilized. Instead, the performance may be degraded due to the excessive introduction of noise. Generally speaking, *DiagFusion* does not need an excessive number of augmented samples as long as the samples are balanced.

*The Number of Layers in GNN:* As the layer number of GNN varies from 1 to 5, the performance of *DiagFusion* in three tasks shows a decreasing trend. The model performs best when the layer number is lower than 3. We do not recommend setting the layer number too large since training deep GNN requires extra training samples, which is hard to meet in real-world microservice systems.

*Time Window:* The length of the time window has little impact on performance because the moments when failures occur are sparse, and the anomaly events reported in a time window are only relevant to the current failure. With accurate anomaly detection, the performance of *DiagFusion* is stable.

## VI. Discussion

### A. Why Learning-Based Methods?

The *DiagFusion* approach incorporates several learning-based techniques, such as fastText in the Unified Event Representation (Section IV-B) and GNN (Section IV-C). By doing so, *DiagFusion* significantly outperforms baseline approaches. We chose to build *DiagFusion* using learning-based methods for the following reasons: 1) *Accuracy:* learning-based methods provide high accuracy (Section V) and are therefore ideal for diagnosing failures. 2) *Generalization ability:* failure cases used to train *DiagFusion* contain different patterns of failure propagation for different systems. A strong generalization ability allows *DiagFusion* to perform robust diagnosis for each system. 3) *Ability to handle complicated data:* as microservice systems become increasingly complex and monitoring data more high-dimensional, manually setting up rules for failure diagnosis becomes time-consuming and error-prone. Learning-based methods, on the other hand, take this data as input and learn their relationships, making them well-suited to handle complicated data.

*Why FastText?* FastText was chosen because trace, log, and metric data have very different formats. However, they all share timestamps, meaning they can be sequenced according to their temporal order. FastText provides superior performance over other static embeddings like word2vec and GloVe, which was demonstrated in Section V-C. Although deep dynamic embeddings like ELMo, BERT, and GPT are popular in Natural Language Processing, they are not suitable for microservice settings as the number of failure cases is insufficient to train these large models.

*Why GNN?* GNN was chosen because the structure of microservice systems involves many instances and their relationships, which form the structure of a graph. Various approaches incorporating Random Walk [12], [13] exist to accomplish failure diagnosis on such graph structures. However, their ability to generalize is limited since domain knowledge can vary greatly between different systems. The domain knowledge contained in graph data can be effectively learned by GNNs [51], giving them a stronger generalization ability than approaches based on Random Walk.

*Concerns About Learning-Based Methods:* While learning-based methods offer several advantages, they do require labeled samples for training. This can be addressed by 1) utilizing the well-established failure management system in microservice systems as a natural source of failure labeling, 2) *DiagFusion* not requiring too many training samples to achieve good performance (the sizes of the training set of D1 and D2 are 160 and 80, respectively), and 3) the increasing adoption of chaos engineering, which enables operators to quickly obtain sufficient failure cases. Several successful practices with the help of chaos engineering have been reported [2], [6], [16], [18].

### B. Robustness

In practice, some modalities can be absent, hindering a successful failure diagnosis system to some extent. The cause of missing modalities can be generally classified into three categories. The first category refers to missing modalities caused

TABLE VI
ROBUSTNESS COMPARED TO PDIAGNOSE (TASK #1)

| Modality | *DiagFusion* | | PDiagnose | |
|---|---|---|---|---|
| | A@1 | A@3 | A@1 | A@3 |
| Trace, Log, Metric | 0.419 | 0.813 | 0.272 | 0.554 |
| Trace, Log | 0.274 | 0.661 | 0 | 0.161 |

by data collection problems. Modern microservice systems are developing rapidly; the same truth applies to their monitoring agents. Therefore, it is hard to guarantee that all monitoring data are ideally collected and transmitted. As a result, missing data is inevitable, which can give rise to missing modalities when specific modalities of the monitoring data are having collection problems. The second category refers to missing modalities caused by data availability problems. In some large corporations, monitoring data is individually collected by many different divisions. Sometimes, specific modalities can be exclusively governed by a division that does not want to disclose its service maintenance data. Thus, these modalities are collected but not available to general operators. The third category stands for missing modalities caused by data retrieval problems. In practice, we often encounter situations where it is very inconvenient to retrieve monitoring data from the data pool. Multimodal failure diagnosis requires much more data to be collected than single-modal-based methods and may face missing modality problems. However, an excellent multimodal-based approach should perform well even when some modalities are missing. We discover that 62 failure cases of D1 lack metric data. *DiagFusion* is compared with PDiagnose in these cases. As PDiagnose cannot address Task #2, we only present the results of Task #1.

As shown in Table VI, the performance of PDiagnose drops dramatically in these cases, while *DiagFusion* presents salient robustness. Although *DiagFusion* also witnesses a performance degradation, it is still better than PDiagnose and other Task #1 baselines. *DiagFusion* has seen complete data modalities during training and learned a unified representation, allowing it to capture anomalous patterns' correlation to failures better than single-modal-based methods. On the other hand, PDiagnose treats each modality independently, making it ineffective when facing missing modalities. To sum up, *DiagFusion* demonstrates robustness since it achieves satisfactory performance even when working with data with incomplete modalities.

### C. Concerns About Deployment and Validity

There are some concerns about deploying *DiagFusion* to real-world microservice systems: 1) *DiagFusion* needs to adapt to the highly dynamic nature of microservice architecture. The stored model of *DiagFusion* can still be effective when service instances are created or destroyed, for *DiagFusion* utilizes the concept of service group as a middle layer. The only situation in which *DiagFusion* needs to be retrained is when new service groups are created. However, the creation of service groups is very rare in practice. 2) Some production systems do not monitor all three modalities at the same time. The workflow of *DiagFusion* is general because the event embedding model is trained on event sequences and does not rely on any specific modality. Besides, the GNN module deals with feature vectors rather than original monitor data. *DiagFusion* can work given that any two of the three modalities are available.

There are two main threats to the validity of the study. The first one lies in the limited sizes of the two datasets used in the study. D1 and D2 are relatively smaller than complex industrial microservice systems. The second one lies in the limitation of the failure cases used in the study. Some failure cases of D1 are simpler than industrial failures and represent only a limited part of different types of failures. However, according to our experiments, *DiagFusion* is effective and robust. It is very promising that *DiagFusion* can also be effectively applied to much larger industrial microservice systems and more complex failure cases.

## VII. RELATED WORK

*Metric-Based Failure Diagnosis Methods:* Monitoring metrics are one of the most important observable data in microservice systems. Many works try to build a dependency graph to depict the interaction between system components during failure, such as Microscope [11], MS-Rank [12], and AutoMAP [13]. However, the correctness of the above works heavily depends on the parameter settings, which degrades their applicability. Besides, many methods extract features from system failures, such as Graph-RCA [52] and iSQUAD [50]. Nonetheless, failure cases are few in microservice systems because operators try to run the system as robustly as possible, severely affecting the performance of these feature-based methods.

*Trace-Based Failure Diagnosis Methods:* Trace can be used to localize the culprit service, for example, TraceRCA [4], MEPFL [18], MicroHECL [5], and MicroRank [6]. However, these trace-based methods often focus on the global feature of the systems and do not deal with the local features of a service instance.

*Log-Based Failure Diagnosis Methods:* LogCluster [7] performs hierarchical clustering on log sequences and matches online log sequences to the most similar cluster. Cloud19 [8] applies word2vec to construct the vectorized representation of a log item and trains classifiers to identify the failure type. Onion [9] performs contrast analysis on agglomerated log cliques to find incident-indicating logs. DeepLog [10] and LogFlash [53] integrate anomaly detection and failure diagnosis. They calculate the deviation from normal status and suggest the root cause accordingly. Log-based methods often ignore the topological feature of microservice systems.

*Multimodal Data-Based Failure Diagnosis Methods:* Recently, combining multimodal data to conduct failure diagnosis has drawn increasing attention. CloudRCA [48] uses both metric and log. It uses the PC algorithm to learn the causal relationship between anomaly patterns of metrics, anomaly patterns of logs, and types of failure. Then it constructs a hierarchical Bayesian Network to infer the failure type. PDiagnose [47] combines metric, log, and trace. It uses lightweight anomaly detection of the three modalities to detect anomaly patterns. Then its vote-based strategy selects the most severe component as the root cause. However, these two methods ignore the topology feature of microservice systems. Groot [54] integrates metrics,

TABLE VII
COMPARISON OF *DIAGFUSION* AND EXISTING REPRESENTATIVE APPROACHES

| Modality | Representative Approach | Core Technique | Diagnosis Result |
|---|---|---|---|
| Metric | AutoMAP [13] | Causal inference & random walk | Root Cause Instance |
| Metric | MS-Rank [12] | Causal inference & random walk | Root Cause Instance |
| Log | LogCluster [7] | Word2vec & traditional classifier | Failure Type |
| Log | Cloud19 [8] | Clustering | Failure Type |
| Trace | MicroRank [6] | Spectrum analysis & PageRank | Root Cause Instance |
| Trace | MicroHECL [5] | Graph traverse & Pearson correlation | Root Cause Instance |
| Multimodal | CloudRCA [48] | Bayesian inference | Failure Type |
| Multimodal | PDiagnose [47] | Vote-based strategy | Root Cause Instance |
| Multimodal | *DiagFusion* (ours) | Event embedding & GNN | Root Cause Instance & Failure Type |

status logs, and developer activity. It needs numerous predefined rules to conduct accurate failure diagnosis, which degrades its applicability to most scenarios.

We compare *DiagFusion* and existing representative approaches in Table VII. In conclusion, compared to single-modal-based methods, *DiagFusion* takes the three important modalities into account. Compared to existing multimodal-based methods, *DiagFusion* is among the first to represent different modalities in a unified manner, thus performing more robustly and accurately.

## VIII. CONCLUSION

Failure diagnosis is of great importance for microservice systems. In this paper, we first conduct an empirical study to illustrate the importance of using multimodal data (i.e., trace, metric, log) for failure diagnosis of microservice systems. Then we propose *DiagFusion*, an automatic failure diagnosis method, which first extracts events from three modalities of data and applies fastText embedding to unify the event from different modalities. During training, *DiagFusion* leverages data augmentation to tackle the challenge of data imbalance. Then it constructs a dependency graph by combining trace and deployment data. Moreover, *DiagFusion* integrates event embedding and the dependency graph through GNN. Finally, the GNN reports the root cause instance and the failure type of online failure. We evaluate *DiagFusion* using two real-world datasets. The evaluation results confirm the effectiveness and efficiency of *DiagFusion*.

## REFERENCES

[1] X. Guo et al., “Graph-based trace analysis for microservice architecture understanding and problem diagnosis,” in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 1387–1397.

[2] X. Zhou et al., “Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study,” *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 243–260, Feb. 2021.

[3] AWS, “Summary of the AWS service event in the Northern Virginia (US-EAST-1) region,” 2021. [Online]. Available: https://aws.amazon.com/cn/message/12721/

[4] Z. Li et al., “Practical root cause localization for microservice systems via trace analysis,” in *Proc. IEEE/ACM 29th Int. Symp. Qual. Serv.*, 2021, pp. 1–10.

[5] M. Jin et al., “An anomaly detection algorithm for microservice architecture based on robust principal component analysis,” *IEEE Access*, vol. 8, pp. 226 397–226 408, 2020.

[6] G. Yu et al., “MicroRank: End-to-end latency issue localization with extended spectrum analysis in microservice environments,” in *Proc. Web Conf.*, 2021, pp. 3087–3098.

[7] Q. Lin et al., “Log clustering based problem identification for online service systems,” in *Proc. 38th Int. Conf. Softw. Eng. Companion*, 2016, pp. 102–111.

[8] Y. Yuan, W. Shi, B. Liang, and B. Qin, “An approach to cloud execution failure diagnosis based on exception logs in OpenStack,” in *Proc. IEEE 12th Int. Conf. Cloud Comput.*, 2019, pp. 124–131.

[9] X. Zhang et al., “Onion: Identifying incident-indicating logs for cloud systems,” in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1253–1263.

[10] M. Du et al., “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1285–1298.

[11] J. Lin, P. Chen, and Z. Zheng, “Microscope: Pinpoint performance issues with causal graphs in micro-service environments,” in *Proc. 16th Int. Conf. Serv.-Oriented Comput.*, Springer, Hangzhou, China, Nov. 12–15, 2018, pp. 3–20.

[12] M. Ma, W. Lin, D. Pan, and P. Wang, “Self-adaptive root cause diagnosis for large-scale microservice architecture,” *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1399–1410, May/Jun. 2022.

[13] M. Ma et al., “AutoMAP: Diagnose your microservice-based web applications automatically,” in *Proc. Web Conf.*, Y. Huang Eds. et al., Taipei, Taiwan, Apr. 20–24, 2020, pp. 246–258.

[14] Y. Pan et al., “Faster, deeper, easier: Crowdsourcing diagnosis of microservice kernel failure from user space,” in *Proc. 30th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2021, pp. 646–657.

[15] B. H. Sigelman et al., “Dapper, a large-scale distributed systems tracing infrastructure,” 2010. [Online]. Available: http://research.google.com/archive/papers/dapper-2010-1.pdf

[16] T. Yang et al., “AID: Efficient prediction of aggregated intensity of dependency in large-scale cloud systems,” in *Proc. IEEE/ACM 36th Int. Conf. Automated Softw. Eng.*, 2021, pp. 653–665.

[17] J. Kaldor et al., “Canopy: An end-to-end performance tracing and analysis system,” in *Proc. 26th Symp. Operating Syst. Princ.*, Shanghai, China, Oct. 28–31, 2017, pp. 34–50.

[18] X. Zhou et al., “Latent error prediction and fault localization for microservice applications by learning from system trace logs,” in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 683–694.

[19] C. Zhang et al., “DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning,” in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 623–634.

[20] B. Li et al., “Enjoy your observability: An industrial survey of microservice tracing and analysis,” *Empirical Softw. Eng.*, vol. 27, no. 1, 2022, Art. no. 25.

[21] P. Liu et al., “Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks,” in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng.*, 2020, pp. 48–58.

[22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in *Proc. IEEE Int. Conf. Web Serv.*, I. Altintas and S. Chen, Eds., Honolulu, HI, USA, Jun. 25–30, 2017, pp. 33–40.

[23] S. Zhang et al., “Syslog processing for switch failure diagnosis and prediction in datacenter networks,” in *Proc. 25th IEEE/ACM Int. Symp. Qual. Serv.*, Vilanova i la Geltrú, Spain, Jun. 14–16, 2017, pp. 1–10.

[24] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “Towards automated log parsing for large-scale log data analysis,” *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 931–944, Nov./Dec. 2018.

[25] S. Messaoudi et al., “A search-based approach for accurate identification of log message formats,” in *Proc. 26th Conf. Prog. Comprehension*, F. Khomh, C. K. Roy, and J. Siegmund, Eds., Gothenburg, Sweden, May 27/28, 2018, pp. 167–177.

[26] M. Du and F. Li, “Spell: Online streaming parsing of large unstructured system logs,” *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 11, pp. 2213–2227, Nov. 2019.

[27] H. Dai, H. Li, C. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using $n$-gram dictionaries," *IEEE Trans. Softw. Eng.*, vol. 48, no. 3, pp. 879–892, Mar. 2022.

[28] M. Sun et al., "CTF: Anomaly detection in high-dimensional time series with coarse-to-fine model transfer," in *Proc. IEEE 40th Conf. Comput. Commun.*, Vancouver, BC, Canada, May 10–13, 2021, pp. 1–10.

[29] Y. Su et al., "Detecting outlier machine instances through Gaussian mixture variational autoencoder with one dimensional CNN," *IEEE Trans. Comput.*, vol. 71, no. 4, pp. 892–905, Apr. 2022.

[30] L. Shen et al., "Time series anomaly detection with multiresolution ensemble decoding," in *Proc. 35th AAAI Conf. Artif. Intell. 33rd Conf. Innov. Appl. Artif. Intell. 11th Symp. Educ. Adv. Artif. Intell.*, 2021, pp. 9567–9575.

[31] M. Ma et al., "Jump-starting multivariate time series anomaly detection for online service systems," in *Proc. USENIX Annu. Tech. Conf.*, I. Calciu and G. Kuenning, Eds., USENIX Assoc., 2021, pp. 413–426.

[32] Z. Li et al., "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, F. Zhu, B. C. Ooi, and C. Miao, Eds., 2021, pp. 3220–3230.

[33] L. Dai et al., "SDFVAE: Static and dynamic factorized VAE for anomaly detection of multivariate CDN KPIs," in *Proc. Web Conf.*, J. Leskovec Eds. et al., 2021, pp. 3076–3086.

[34] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[35] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, A. Moschitti, B. Pang, and W. Daelemans, Eds., 2014, pp. 1532–1543.

[36] P. Bojanowski et al., "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017.

[37] M. E. Peters et al., "Deep contextualized word representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, M. A. Walker, H. Ji, and A. Stent, Eds., New Orleans, LA, USA, Jun. 1–6, 2018, pp. 2227–2237.

[38] J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[39] T. B. Brown et al., "Language models are few-shot learners," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, H. Larochelle Eds. et al., 2020, Art. no. 159.

[40] W. Meng et al., "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, S. Kraus, Ed., Macao, China, Aug. 10–16, 2019, pp. 4739–4745.

[41] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[42] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, I. Guyon Eds. et al., Long Beach, CA, USA, 2017, pp. 1024–1034.

[43] L. Zhou, Q. Zeng, and B. Li, "Hybrid anomaly detection via multihead dynamic graph attention networks for multivariate time series," *IEEE Access*, vol. 10, pp. 40 967–40 978, 2022.

[44] L. Zhang, B. Morin, P. Haller, B. Baudry, and M. Monperrus, "A Chaos engineering system for live analysis and falsification of exception-handling in the JVM," *IEEE Trans. Softw. Eng.*, vol. 47, no. 11, pp. 2534–2548, Nov. 2021.

[45] Y. Wang et al., "Fast outage analysis of large-scale production clouds with service correlation mining," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, Madrid, Spain, May 22–30, 2021, pp. 885–896.

[46] Y. Zhou et al., "Graph neural networks: Taxonomy, advances, and trends," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 1, pp. 15:1–15:54, 2022.

[47] C. Hou, T. Jia, Y. Wu, Y. Li, and J. Han, "Diagnosing performance issues in microservices with heterogeneous data source," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Social Comput. Netw.*, New York, NY, USA, 2021, pp. 493–500.

[48] Y. Zhang et al., "CloudRCA: A root cause analysis framework for cloud computing platforms," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage.*, G. Demartini Eds. et al., 2021, pp. 4373–4382.

[49] Y. Meng et al., "Localizing failure root causes in a microservice through causality inference," in *Proc. IEEE/ACM 28th Int. Symp. Qual. Serv.*, Hangzhou, China, Jun. 15–17, 2020, pp. 1–10. [Online]. Available: https://doi.org/10.1109/IWQoS49365.2020.9213058

[50] M. Ma et al., "Diagnosing root causes of intermittent slow queries in large-scale cloud databases," in *Proc. VLDB Endowment*, vol. 13, no. 8, pp. 1176–1189, 2020.

[51] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 249–270, Jan. 2022.

[52] Á. Brandón et al., "Graph-based root cause analysis for service-oriented and microservice architectures," *J. Syst. Softw.*, vol. 159, 2020, Art. no. 110432.

[53] T. Jia, Y. Wu, C. Hou, and Y. Li, "LogFlash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng.*, Z. Jin Eds. et al., Wuhan, China, Oct. 25–28, 2021, pp. 80–90.

[54] H. Wang et al., "Groot: An event-graph-based approach for root cause analysis in industrial settings," in *Proc. IEEE/ACM 36th Int. Conf. Automated Softw. Eng.*, Melbourne, Australia, Nov. 15–1, 2021, pp. 419–429.

**Shenglin Zhang** (Member, IEEE) received the BS degree in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2017. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His current research interests include failure detection, diagnosis, and prediction for service management.

**Pengxiang Jin** received the bachelor's degree in software engineering from Nankai University, Tianjin, China, in 2020. He is currently working toward the master degree with the College of Software, Nankai University. His research interests include anomaly detection and anomaly localization.

**Zihan Lin** received the bachelor's degree in software engineering from Nankai University, Tianjin, China, in 2021. He is currently working toward the master degree with the College of Software, Nankai University. His research interests include failure localization and anomaly detection.

**Yongqian Sun** (Member, IEEE) received the BS degree in statistical specialty from Northwestern Polytechnical University, Xi'an, China, in 2012, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2018. He is currently an assistant professor with the College of Software, Nankai University, Tianjin, China. His research focuses on anomaly detection, root cause analysis, and failure diagnosis in service management.

**Bicheng Zhang** received the bachelor's degree from Nankai University. He is currently working toward the master degree with Fudan University. His research interests include cloud native and AIOps.

**Sibo Xia** is currently working toward the master degree. His main research interests include knowledge graph, failure detection, and diagnosis.

**Zhengdan Li** is an experimentalist with Nankai University, Tianjin, China. Her research interests include artificial intelligence and software engineering.

**Zhenyu Zhong** received the BS degree in software engineering from Nankai University, Tianjin, China, in 2020. He is currently working toward the PhD degree with the College of Software, Nankai University, Tianjin, China. His current research interests include anomaly detection, deep learning, and NLP.

**Minghua Ma** (Member, IEEE) received the PhD degree from Tsinghua University, in 2021. He is a researcher with Microsoft. His current research interests include cloud intelligence/AIOps.

**Wa Jin** is currently working toward the bachelor degree. Her main research interests include anomaly detection and failure diagnosis.

**Dai Zhang** is employed with ZhejiangE-CommerceBank Co., Ltd., Launched by Ant Group. As a technical expert, he mainly focuses on financial basic technical architecture and cloud-native system stability.

**Zhenyu Zhu** is employed with ZhejiangE-CommerceBank Co., Ltd., Launched by Ant Group. As a technical expert, he mainly focuses on financial basic technical architecture and cloud-native system stability.

**Dan Pei** (Senior Member, IEEE) received the BE and MS degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, in 1997 and 2000, respectively, and the PhD degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA), in 2005. He is currently an associate professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include network and service management in general. He is asenior member of the ACM.