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, Wa Jin, Dai Zhang, Zhenyu Zhu, 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 this data and addressing imbalanced failures remain a challenge. To tackle these issues, we introduce *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 and failure type determination.

Index Terms—Microservice systems, Failure diagnosis, Multimodal data, Graph neural network

1 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 record hardware and software events of a service instance, including business events, state changes, hardware errors, *etc*. Metrics are used to monitor service status and include system metrics (*e.g.*, CPU utilization, memory utilization) and user-perceived metrics (*e.g.*, average response time, error rate). Metrics are usually

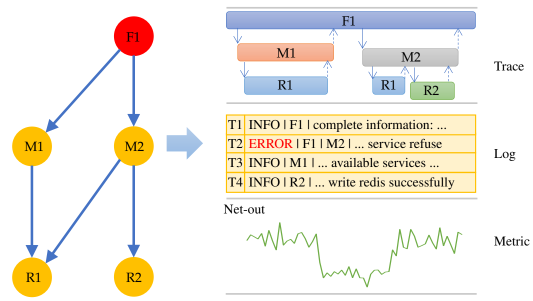


Fig. 1: Multimodal data of microservice systems. F1, M1, M2, R1, and R2 are service instances, and Tx are timestamps.

collected at a fixed interval (*e.g.*, once per minute) and thus form time series data. From now on, we use the term *modality* to describe a particular data type. Figure 1 shows an example of a microservice system and the three modalities.

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, logs, or metrics, to capture failure patterns. Trace-based methods, for example, use machine learning techniques to extract the features of service invocation and localize the root cause instance [1], [4]–[7]. Log-based methods, on the other hand, transform log items into vectors and use feature extraction to infer the failure type [8]–[11]. Finally, metric-based methods typically construct a dependency graph and determine the root cause instance based on the failure's propagation pattern in the graph [12]–[15].

However, relying solely on single-modal data for diagnosing failures in microservice systems is not effective enough. Our empirical study of an open-source dataset shows the limitations of these methods (as seen in Table 1).

- *Y. Sun is the corresponding author. S. Zhang, P. Jin, Z. Lin, Y. Sun, S. Xia, Z. Li, Z. Zhong, and W. Jin are with Nankai University, Tianjin, China. Email: {zhangsl, sunyongqian, lzd}@nankai.edu.cn, {jinpengxiang, linzihan, xiasibo, zyzhong, 1913173}@mail.nankai.edu.cn.*
- *B. Zhang is with Fudan University. Email: 22210240069@m.fudan.edu.cn.*
- *M. Ma is with Microsoft Research Asia, Beijing, China. Email: minghuama@microsoft.com.*
- *D. Zhang and Z. Zhu are with ZhejiangE-CommerceBank Co., Ltd. Email: {henry.zd, michael.zzy}@mybank.cn*
- *D. Pei is with Department of Computer Science, Tsinghua University, Beijing, China, and also with Beijing National Research Center for Information Science and Technology. Email: peidan@tsinghua.edu.cn.*

The reasons for this are twofold. 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. After examining hundreds of service instance failures, we conclude that combining traces, logs, and metrics (*multimodal*) is crucial for accurate diagnosis. For example, in Figure 1, the red-marked service experienced 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.

Recently, the combination of multimodal data has garnered much attention in other fields [16], and a popular approach is to use Graph Neural Networks (GNNs) [17], [18]. Given that the dependency relationships in microservice systems form a natural graph structure, we apply GNNs to learn failure patterns in these systems to pinpoint the root cause instance and determine failure types. However, there are two main challenges in using GNNs for diagnosing failures in microservice systems:
(1) **Representation of multimodal data.** Service instance metrics are often in the form of time series (the bottom right of Figure 1), while logs are usually semi-structured text (the middle right of Figure 1) and traces often take the form of tree structures with spans as nodes (the top right of Figure 1). It is challenging to find a unified representation of all this multimodal data that allows GNNs to utilize complementary information from each data type effectively.
(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 1).

In this paper, 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 nature, *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. This allows *DiagFusion* to conduct root cause instance localization and failure type determination.

The contributions of this paper are summarized as follows:
(1) We propose *DiagFusion*, a multimodal data-based approach for failure diagnosis. *DiagFusion* builds a dependency graph from trace and deployment data to capture possible failure propagation paths. Then it applies a GNN to achieve 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).
(2) 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.
(3) We conduct extensive experiments on two datasets, one from an open-source platform and another from a real-world microservice system. The results show that when *DiagFusion* is trained based 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* outperforms two state-of-the-art approaches in *failure type determination*.

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

2 Background

2.1 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. Figure 1 shows an example traces in the top-right corner. Google formally proposed the concept of traces at Dapper [19], 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], [5], [7], [20]–[24]. 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-right of Figure 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:

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

TABLE 1: Detailed information of the selected failures in the empirical study.

Failure Type	Metric	Log	Trace	# Failures
High memory usage	memory_usage_pct ↑	-	-	505
Free using memory	memory_stats_active_anon ↓	-	-	16
Login failure	-	ERROR ∣ 0.0.0.1 ∣ 172.17.0.5 ∣ M1 ∣ uuid: 78fef9f0 information has expired, mobile phone login is invalid	S1->S2: RT=11s	527
File not found	-	ERROR ∣ 0.0.0.3 ∣ W2 ∣ get an error [Errno 2] No such file or directory: 'resources/source_file/source_file.csv'	S2->S3: RT=1.5s	36
Access denied	-	ERROR ∣ 0.0.0.2 ∣ B2 ∣ 2768e0e0037e ∣ service refuse	S2->S4: RT=1.1s	15

timestamp, verbosity level, and raw message [25]. 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 [26], Drain [25], POP [27], MoLFI [28], Spell [29], and Logram [30].

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 in the bottom-right of Figure 1. These metrics track various aspects of performance issues. By examining metrics, operators can determine which physical resource is anomalous or is the bottleneck [31]–[36].

Deployment data. A microservice system comprises many hardware and software assets that form complicated inter-relationships. Operators must carefully record these relationships (*a.k.a.* deployment data) to keep high maintainability of the system. In addition, the deployment data are a valuable source for failure diagnosis and can be utilized to learn failures' propagation path and characteristics.

2.2 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 includes static representation like word2vec [37], GloVe [38], fastText [39], and dynamic representation like ELMo [40], BERT [41], GPT [42]. With the similarities between logs and natural languages, representation learning can be applied to extract log features [43]. We employ fastText to learn a unified representation of events from multimodal data. Compared to word2vec and GloVe, fastText can utilize more information [39].

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) [17], GraphSAGE [44], and Graph Attention Network (GAT) [45], *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.

2.3 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 [46]. It can enlarge the number of failure cases and enrich the types of failures. We train *DiagFusion* based on these failure cases.

3 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 physical resource failures (high memory usage and memory freed incorrectly) and software failures (login service error, access denied, and file not found). The failure injection record is provided along with the data. Table 1 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 1 shows that some failures occur much more frequently than others. For example, the total occurrences of *Free using memory*, *File not found*, 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 the microservice system of Microsoft. Due to the company policy, we have to hide some details of these failures. The

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

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 Figure 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.

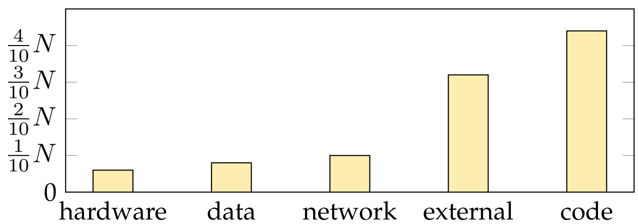


Fig. 2: The distribution of failure types at a large-scale real-world microservice system.

4 Approach

4.1 Design Overview

In this paper, we propose *DiagFusion*, which combines the modality of trace, log, and metric for accurate failure diagnosis. The training framework of *DiagFusion* is summarized in Figure 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 GNN. We train GNN using historical failures to learn the propagation pattern of system failures. After the training stage, we save the event embedding model and the GNN. Figure 6 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.

4.2 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 parsing techniques for logs. Then, it trains a fastText [39] model on event sequences to generate embedding vectors of events.

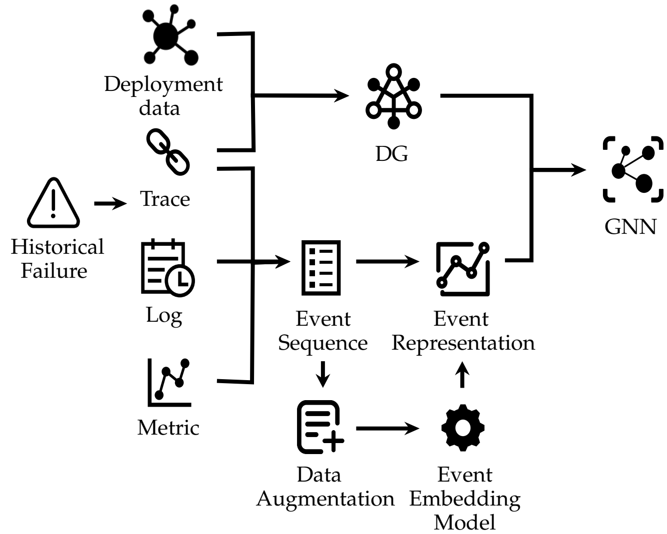


Fig. 3: The training framework of *DiagFusion*.

Instance and service group. 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.

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' interest. 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 [25], 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

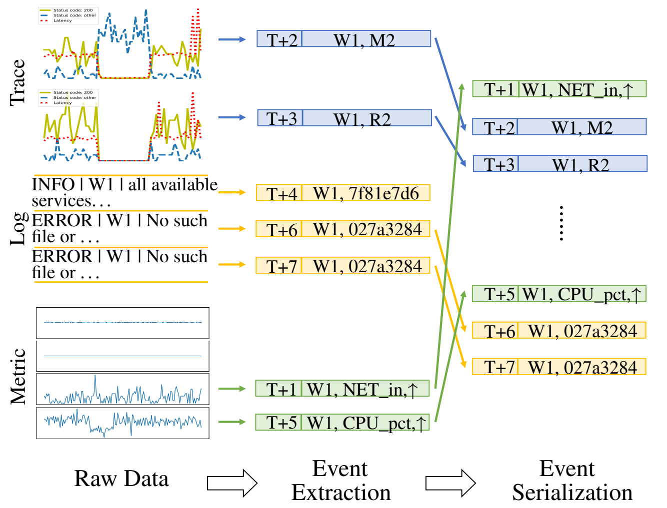


Fig. 4: The event extraction and serialization process using traces, logs, and metrics.

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*>.

Multimodal event grouping and serialization. Despite the differences in data modalities, 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*. Figure 4 shows the event extraction and serialization process for one instance. The event sequence of instance i is denoted by E_i.

Instance-wise relabeling. 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.

Event embedding model. Inspired by the success of log embedding in log analysis, we propose the concept of event embedding, which maps events into embedding 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 4.4.

4.3 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 GNN to learn the failure propagation between service instances and integrate all the information of the whole system.

Instance representation. 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.

Dependency graph building. There are two dominant ways of propagation failure between services: function calling or resource contention [47]. 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. We add edges between two instances if they are co-deployed for deployment data.

Message passing. After obtaining the dependency graph and instance representations, we train 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 [48] and update the internal data of instances according to:

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

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

Readout. We add a readout layer, *i.e.*, a MaxPooling layer, to the GNN to integrate the information of the whole microservice system. After the readout layer, there is a fully connected layer with output neurons. Each neuron corresponds to either a service group with possible root cause instances for task #1 or a failure type for task #2.

4.4 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.

4.4.1 *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

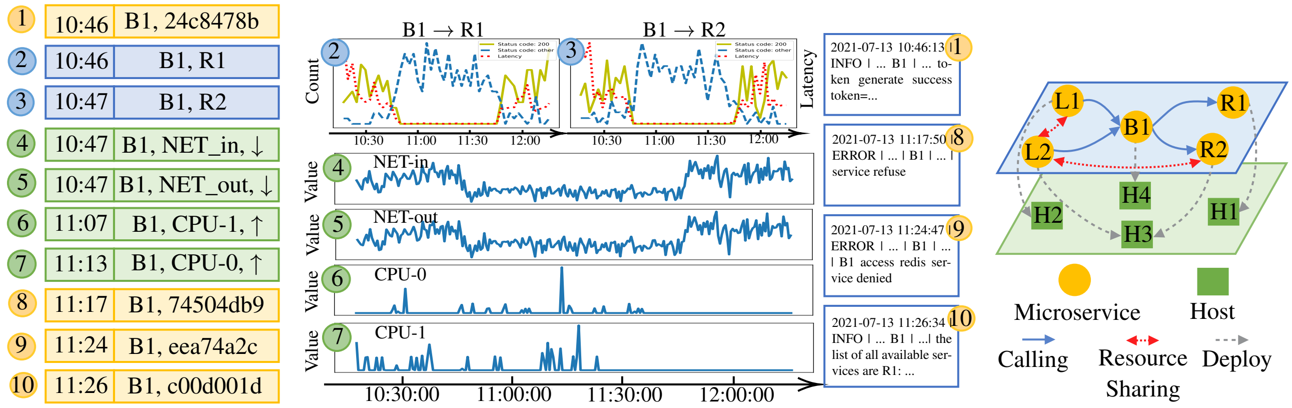


Fig. 5: A running example of *DiagFusion*. (a): the serialized multimodal event sequence of the root cause instance (B1); (b): the original data corresponding to the event sequence; (c): part of the dependency graph in this failure.

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 5.5. 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.

4.4.2 Training of Graph Neural Network

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 readout 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.) We assign the same weight for task #1 and task #2 to combine their loss functions. 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, $y\left(s\right)$ is the root cause service group labeled by operators, $y\left(t\right)$ is the failure type, $p\left(s\right)$ is the predicted service group, and $p\left(t\right)$ is the predicted failure type.

4.5 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 Figure 6.

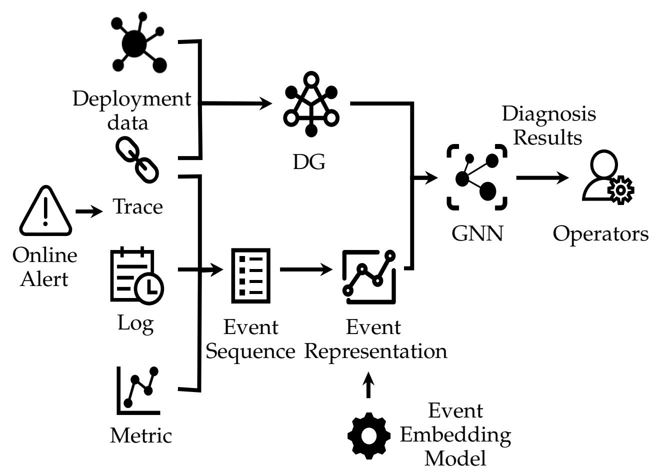


Fig. 6: Real-time failure diagnosis.

4.5.1 Running Example

To understand 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. Figure 5 shows the DG, event sequence, and the original data. From Figure 5(a), we can see that failure-indicative events from different modalities

TABLE 2: Detailed information of datasets.

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

are temporally intertwined. It takes *DiagFusion* about 10 seconds to perform diagnosis. Then it gives the result of B1 as the root cause instance and access denied as the failure type, effectively addressing tasks #1 and #2.

5 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 hyperparameter settings?

5.1 Experimental Setup

5.1.1 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 2. The systems that produce D1 and D2 are as follows:

1) D1. The detailed information on D1 is elaborated in Section 3.
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.

5.1.2 Baseline Methods

We select six advanced single-modal-based methods (two for trace (*i.e.*, MicroHECL [6], MicroRank [7]), two for log (*i.e.*, Cloud19 [9], LogCluster [8]), and two for metric (*i.e.*, AutoMAP [14], MS-Rank [13])), and two multimodal-based methods (*i.e.*, PDiagnose [49], CloudRCA [50]) as the baseline methods. More details can be found in Section 7. 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.

TABLE 3: 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

5.1.3 Evaluation Metrics

As stated in Section 2.3, *DiagFusion* aims to localize the root cause instance and diagnose the failure type. We carefully curated different evaluation metrics for different 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 quantifies the probability that top-k instances output by each method indeed contain the root cause instance. Formally, given $|A|$ as the test set of failures, RC_i as the ground truth root cause instance, $RC_s\left[k\right]$ 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 evaluates a method's overall capability of localizing the root cause instance. 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, 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 $recall = \frac{\text{TP}}{\text{TP+FN}}$.

5.1.4 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

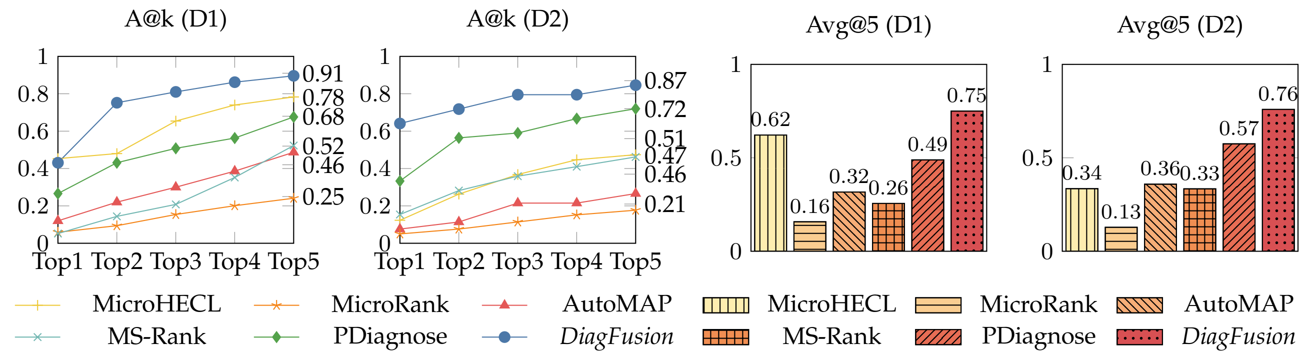


Fig. 7: Effectiveness of root cause instance localization (Task #1).

0.9.0. We run the experiments on a server with 12 × Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz and 128G RAM (without GPUs). We repeat every experiment five times and take the average result to reduce the effect of randomness.

5.2 Overall Performance (RQ1)

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

The comparison result of Task #1 is shown in Figure 7. *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 3. For this task, *DiagFusion* is better than almost all baselines. On D1, the precision, recall, and F1-Score of *DiagFusion* are over 0.80, being second only to Cloud19. All the methods in Table 3 do not perform as well as D1 on D2, which has more failure types and more complex failure patterns. Nevertheless, *DiagFusion* still manages to maintain an F1-Score of 0.80, which is at least 0.195 higher than the baselines. This indicates a greater advantage of using multimodal methods in complex systems.

5.3 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 4 lists that *DiagFusion* outperforms all the variants on D1 and D2, demonstrating each component's significance. When removing the data augmentation (**C1**), the performance 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 utilize supervised information while word2vec and GloVe are self-supervised. Replacing the GNN output layer with classifiers such as decision trees and kNN (**C4 & C5**) degrades performance because the GNN output layer can interpret representations containing graph structure information as prediction results, but these classifiers cannot understand the graph structure information.

5.4 Efficiency (RQ3)

We record the running time of all methods and compare them in Table 5. It shows that *DiagFusion* can diagnose one failure within 12 seconds on average online, and it can achieve quasi-real-time diagnosis because the interval of data collection in D1 and D2 is at least 30 seconds. This means that *DiagFusion* can meet the needs of online diagnosis, although it has no apparent advantages among the methods in Table 5. Offline time is not a sensitive factor because it does not need to be retrained frequently.

5.5 Hyperparameter Sensitivity (RQ4)

We discuss the effect of four hyperparameters of *DiagFusion*. Figure 8 shows how Avg@5 (Task #1), F1-Score (Task #2) change with different hyperparameters.

Embedding dimension. The performance of *DiagFusion* performs 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.

The number of augmented samples. The experiments in Section 5.2 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

TABLE 4: 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

TABLE 5: The comparison of training time (Offline) and diagnosis time (Online) per case. ("-" means this method does not 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

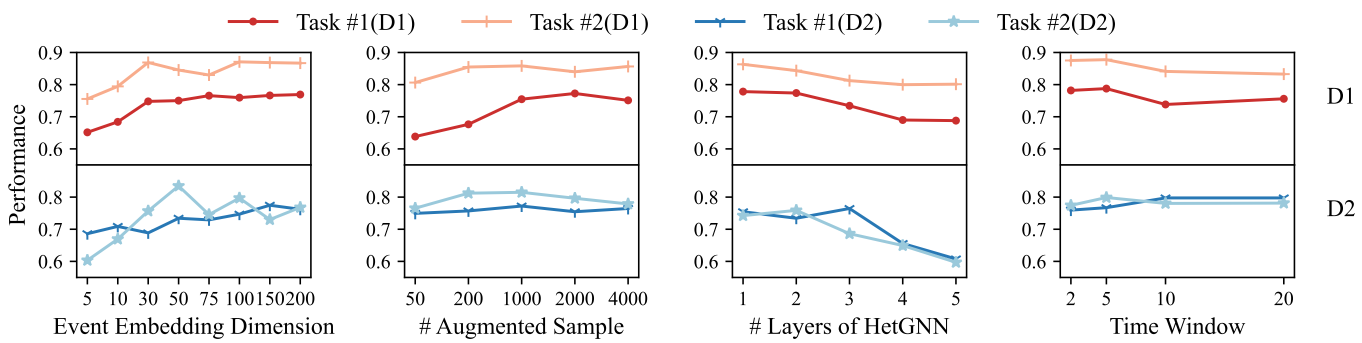


Fig. 8: The effectiveness of *DiagFusion* under different hyperparameters.

failures occur are sparse, and the anomaly events reported in a time window are only relevant to the current failure. With sufficient exception information and accurate anomaly detection, the performance of *DiagFusion* is stable.

6 Discussion

6.1 Why Learning-Based Methods?

The *DiagFusion* approach incorporates several learning-based techniques, such as fastText in the Unified Event Representation (Section 4.2) and GNN (Section 4.3). 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 5) 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 5.3. 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 [13], [14] 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 [18], 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 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 [4], [7], [20], [51].

6.2 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 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.

TABLE 6: 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

As shown in Table 6, 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. We claim that *DiagFusion* is robust because it can still achieve good performance using data with missing modalities in the online environment.

6.3 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 that *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.

7 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 [12], MS-Rank [13], and AutoMAP [14]. 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 [53]. 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 [5], MEPFL [4], MicroHECL [6], and MicroRank [7]. 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 [8] performs hierarchical clustering on log sequences and matches online log sequences to the most similar cluster. Cloud19 [9] applies word2vec to construct the vectorized representation of a log item and trains classifiers to identify the failure type. Onion [10] performs contrast analysis on agglomerated log cliques to find incident-indicating logs. DeepLog [11] and LogFlash [54] integrates 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 [50] 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 [49] 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 [55] integrates metrics, status logs, and developer activity. It needs numerous predefined rules to conduct accurate failure diagnosis, which degrades its applicability to most scenarios.

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.

8 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 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. Both the experiments and the case study show the superior effectiveness and efficiency of *DiagFusion*.

References

[1] X. Guo, X. Peng, H. Wang *et al.*, "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1387–1397.

[2] X. Zhou, X. Peng, T. Xie *et al.*, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Software Eng.*, vol. 47, no. 2, pp. 243–260, 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] X. Zhou, X. Peng, T. Xie *et al.*, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 683–694.

[5] Z. Li, J. Chen, R. Jiao *et al.*, "Practical root cause localization for microservice systems via trace analysis," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[6] M. Jin, A. Lv, Y. Zhu *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.

[7] G. Yu, P. Chen, H. Chen *et al.*, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, 2021, pp. 3087–3098.

[8] Q. Lin, H. Zhang, J.-G. Lou *et al.*, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 102–111.

[9] Y. Yuan, W. Shi, B. Liang *et al.*, "An approach to cloud execution failure diagnosis based on exception logs in openstack," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 124–131.

[10] X. Zhang, Y. Xu, S. Qin *et al.*, "Onion: identifying incident-indicating logs for cloud systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1253–1263.

[11] M. Du, F. Li, G. Zheng *et al.*, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

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

[13] M. Ma, W. Lin, D. Pan *et al.*, "Self-adaptive root cause diagnosis for large-scale microservice architecture," *IEEE Trans. Serv. Comput.*, vol. 15, no. 3, pp. 1399–1410, 2020.

[14] M. Ma, J. Xu, Y. Wang *et al.*, "Automap: Diagnose your microservice-based web applications automatically," in *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Y. Huang, I. King, T. Liu *et al.*, Eds. ACM / IW3C2, 2020, pp. 246–258.

[15] Y. Pan, M. Ma, X. Jiang *et al.*, "Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 646–657.

[16] T. Baltrusaitis, C. Ahuja, and L. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, 2018.

[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[18] 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, 2020.

[19] B. H. Sigelman, L. A. Barroso, M. Burrows *et al.*, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Inc., Tech. Rep., 2010.

[20] T. Yang, J. Shen, Y. Su *et al.*, "Aid: efficient prediction of aggregated intensity of dependency in large-scale cloud systems," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 653–665.

[21] J. Kaldor, J. Mace, M. Bejda *et al.*, "Canopy: An end-to-end performance tracing and analysis system," in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017, pp. 34–50.

[22] C. Zhang, X. Peng, C. Sha *et al.*, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 623–634.

[23] B. Li, X. Peng, Q. Xiang *et al.*, "Enjoy your observability: an industrial survey of microservice tracing and analysis," *Empir. Softw. Eng.*, vol. 27, no. 1, p. 25, 2022.

[24] P. Liu, H. Xu, Q. Ouyang *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 48–58.

[25] P. He, J. Zhu, Z. Zheng *et al.*, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, I. Altintas and S. Chen, Eds. IEEE, 2017, pp. 33–40.

[26] S. Zhang, W. Meng, J. Bu *et al.*, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *25th IEEE/ACM International Symposium on Quality of Service, IWQoS 2017, Vilanova i la Geltrú, Spain, June 14-16, 2017*. IEEE, 2017, pp. 1–10.

[27] P. He, J. Zhu, S. He *et al.*, "Towards automated log parsing for large-scale log data analysis," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 6, pp. 931–944, 2018.

[28] S. Messaoudi, A. Panichella, D. Bianculli *et al.*, "A search-based approach for accurate identification of log message formats," in *Proceedings of the 26th Conference on Program Comprehension, ICPC*

2018, Gothenburg, Sweden, May 27-28, 2018, F. Khomh, C. K. Roy, and J. Siegmund, Eds. ACM, 2018, pp. 167–177.

[29] 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, 2019.

[30] H. Dai, H. Li, C. Chen *et al.*, "Logram: Efficient log parsing using n-gram dictionaries," *IEEE Trans. Software Eng.*, vol. 48, no. 3, pp. 879–892, 2022.

[31] M. Sun, Y. Su, S. Zhang *et al.*, "CTF: anomaly detection in high-dimensional time series with coarse-to-fine model transfer," in *40th IEEE Conference on Computer Communications, INFOCOM 2021, Vancouver, BC, Canada, May 10-13, 2021*. IEEE, 2021, pp. 1–10.

[32] Y. Su, Y. Zhao, M. Sun *et al.*, "Detecting outlier machine instances through gaussian mixture variational autoencoder with one dimensional CNN," *IEEE Trans. Computers*, vol. 71, no. 4, pp. 892–905, 2022.

[33] L. Shen, Z. Yu, Q. Ma *et al.*, "Time series anomaly detection with multiresolution ensemble decoding," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 9567–9575.

[34] M. Ma, S. Zhang, J. Chen *et al.*, "Jump-starting multivariate time series anomaly detection for online service systems," in *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, I. Calciu and G. Kuenning, Eds. USENIX Association, 2021, pp. 413–426.

[35] Z. Li, Y. Zhao, J. Han *et al.*, "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, F. Zhu, B. C. Ooi, and C. Miao, Eds. ACM, 2021, pp. 3220–3230.

[36] L. Dai, T. Lin, C. Liu *et al.*, "SDFVAE: static and dynamic factorized VAE for anomaly detection of multivariate CDN kpis," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, J. Leskovec, M. Grobelnik, M. Najork *et al.*, Eds. ACM / IW3C2, 2021, pp. 3076–3086.

[37] T. Mikolov, K. Chen, G. Corrado *et al.*, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013.

[38] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1532–1543.

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

[40] M. E. Peters, M. Neumann, M. Iyyer *et al.*, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 2227–2237.

[41] J. Devlin, M.-W. Chang, K. Lee *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[42] T. B. Brown, B. Mann, N. Ryder *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell *et al.*, Eds., 2020.

[43] W. Meng, Y. Liu, Y. Zhu *et al.*, "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.

[44] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio *et al.*, Eds., 2017, pp. 1024–1034.

[45] 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.

[46] L. Zhang, B. Morin, P. Haller *et al.*, "A chaos engineering system for live analysis and falsification of exception-handling in the JVM," *IEEE Trans. Software Eng.*, vol. 47, no. 11, pp. 2534–2548, 2021.

[47] Y. Wang, G. Li, Z. Wang *et al.*, "Fast outage analysis of large-scale production clouds with service correlation mining," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 885–896.

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

[49] C. Hou, T. Jia, Y. Wu *et al.*, "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.

[50] Y. Zhang, Z. Guan, H. Qian *et al.*, "Cloudrca: A root cause analysis framework for cloud computing platforms," in *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, G. Demartini, G. Zuccon, J. S. Culpepper *et al.*, Eds. ACM, 2021, pp. 4373–4382.

[51] X. Zhou, X. Peng, T. Xie *et al.*, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Software Eng.*, vol. 47, no. 2, pp. 243–260, 2021.

[52] Á. Brandón, M. Solé, A. Huélamo *et al.*, "Graph-based root cause analysis for service-oriented and microservice architectures," *J. Syst. Softw.*, vol. 159, 2020.

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

[54] T. Jia, Y. Wu, C. Hou *et al.*, "Logflash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *32nd IEEE International Symposium on Software Reliability Engineering, ISSRE 2021, Wuhan, China, October 25-28, 2021*, Z. Jin, X. Li, J. Xiang *et al.*, Eds. IEEE, 2021, pp. 80–90.

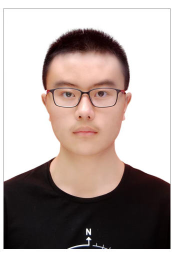
[55] H. Wang, Z. Wu, H. Jiang *et al.*, "Groot: An event-graph-based approach for root cause analysis in industrial settings," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 419–429.

Shenglin Zhang received B.S. in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012 and Ph.D. 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. He is an IEEE Member.

Pengxiang Jin received his bachelor degree in software engineering from Nankai University, Tianjin, China, in 2020. He is currently a master student in the College of Software at Nankai University. His research interests include anomaly detection and anomaly localization.

Zihan Lin received his bachelor degree in software engineering from Nankai University, Tianjin, China, in 2021. He is currently a master student in the College of Software at Nankai University. His research interests include failure localization and anomaly detection.

Yongqian Sun received the B.S. degree in statistical specialty from Northwestern Polytechnical University, Xi'an, China, in 2012, and Ph.D. 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 interests include anomaly detection and root cause localization in service management.



Bicheng Zhang is currently a first-year master student at Fudan University. He received his bachelor degree from Nankai University. His research interests include cloud native and AIOps.

Sibo Xia , born in 2000. Master candidate. His main research interests include knowledge graph, failure detection and diagnosis.

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

Zhenyu Zhong received his B.S. degree in software engineering from Nankai University, Tianjin, China, in 2020. Ph.D. candidate in the College of Software at Nankai University, Tianjin, China. His current research interests include anomaly detection, deep learning and NLP.

Minghua Ma is a researcher in Microsoft Research Lab Asia. Before that, he received his Ph.D. degree from Tsinghua University in 2021. His current research interests include Cloud Intelligence/AIOps.

Wa Jin , born in 2001. 1bachelor candidate. Her main research interests include anomaly detection and failure diagnosis.

Dai Zhang is employed at 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 at 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 received the B.E. and M.S. degree in computer science from the Department of Computer Science and Technology, Tsinghua University in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA) in 2005. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University. His research interests include network and service management in general. He is an IEEE senior member and an ACM senior member.