No More Data Silos: Unified Microservice Failure Diagnosis with Temporal Knowledge Graph

Shenglin Zhang, *Member, IEEE,* Yongxin Zhao, Sibo Xia, Shirui Wei, Yongqian Sun, *Member, IEEE,* Chenyu Zhao, Shiyu Ma, Junhua Kuang, Bolin Zhu, Lemeng Pan, Yicheng Guo, Dan Pei, *Senior Member, IEEE*

***Abstract*—Microservices improve the scalability and flexibility of monolithic architectures to accommodate the evolution of software systems, but the complexity and dynamics of microservices challenge system reliability. Ensuring microservice quality requires efficient failure diagnosis, including detection and triage. Failure detection involves identifying anomalous behavior within the system, while triage entails classifying the failure type and directing it to the engineering team for resolution. Unfortunately, current approaches reliant on single-modal monitoring data, such as metrics, logs, or traces, cannot capture all failures and neglect interconnections among multimodal data, leading to erroneous diagnoses. Recent multimodal data fusion studies struggle to achieve deep integration, limiting diagnostic accuracy due to insufficiently captured interdependencies. Therefore, we propose *UniDiag*, which leverages temporal knowledge graphs to fuse multimodal data for effective failure diagnosis. *UniDiag* applies a simple yet effective stream-based anomaly detection method to reduce computational cost and a novel microservice-oriented graph embedding method to represent the state of systems comprehensively. To assess the performance of *UniDiag*, we conduct extensive evaluation experiments using datasets from two benchmark microservice systems, demonstrating its superiority over existing methods and affirming the efficacy of multimodal data fusion. Additionally, we have publicly made the code and data available to facilitate further research.**

***Index Terms*—Microservice, failure diagnosis, multimodal data, knowledge graph.**

I. Introduction

MICROSERVICES have gained significant attention in the last few years in both industry and academia [1]. However, the complexity and dynamism of microservice systems pose unique challenges for maintenance [2]. When a microservice system fails, the failure will propagate along the interaction network between microservices, posing significant challenges and pressures for operators. Failures will negatively impact user experience and cause enormous losses for service providers if they are not quickly diagnosed and mitigated [3], [4]. To ensure high availability and minimize service downtime, prompt diagnosis, encompassing both detection and triage, is crucial [5], [6]. This allows operators to respond efficiently to failures, mitigating their impact and restoring service functionality as quickly as possible. Proactive failure detection can alert operators timely, and accurate failure triage can not only provide operators with suggestions for mitigation measures (Table I) but also minimize the cost associated with reassignment, which refers to the process of re-allocating the initially assigned mitigation measures to the potentially more appropriate measures [5]. Therefore, rapid and accurate failure diagnosis is indispensable for effectively maintaining microservice systems.

A microservice system is an architecture composed of multiple microservices, with each microservice having multiple instances. To monitor the status of microservice systems in real-time and achieve rapid failure diagnosis, operators often continuously collect three modalities of monitoring data (*i.e.*, metrics, logs, traces) for each microservice instance [7]. Fig. 1 shows the three modalities of data in a microservice system. Metrics include system-level metrics (*e.g.*, CPU utilization, memory utilization, and network throughput) and user-perceived metrics (*e.g.*, average response time, error rate, and page view count), presented as time series. Logs are semi-structured text that records various events and system runtime status. Traces are in the tree structure recording information about user requests like instances and service calls, and each call forms a span. For example, as shown in Fig. 1, after a failure occurs, the "system.cpu.pct_utilization" metric sharply rises, and simultaneously microservice instances S_1 and S_2 generate anomalous log messages, with a significant increase in response time in related traces.

Over the years, many single-modal data-based failure diagnosis approaches for microservice systems have been proposed (*e.g.*, metrics [8]–[10], logs [11]–[13], or traces [14]). However, these approaches usually lead to a large number of misdiagnoses due to the inherent limitations of the characteristics of the data used:

Limitations of Metrics. Metrics can react rapidly to changes in system performance [15], reflecting various failure modes via higher or lower values. However, the oversensitivity of the metrics may produce false alarms [15], [16]. For example, in scenarios where memory is effectively released, notwithstanding a potentially decreased memory uti-

Y. Sun is the corresponding author. Y. Sun is with the College of Software, Nankai University, Tianjin, China, and also with the Tianjin Key Laboratory of Software Experience and Human Computer Interaction (TKL-SEHCI), Tianjin, China. Email: sunyongqian@nankai.edu.cn

S. Zhang is with the College of Software, Nankai University, Tianjin, China, and also with the Haihe Laboratory of Information Technology Application Innovation (HL-IT), Tianjin, China. Email: zhangsl@nankai.edu.cn

Y. Zhao, S. Xia, S. Ma, and J. Kuang are with Nankai University, Tianjin, China. Email: {zyx_nkcs, xiath, mashiyu, 2013157}@mail.nankai.edu.cn.

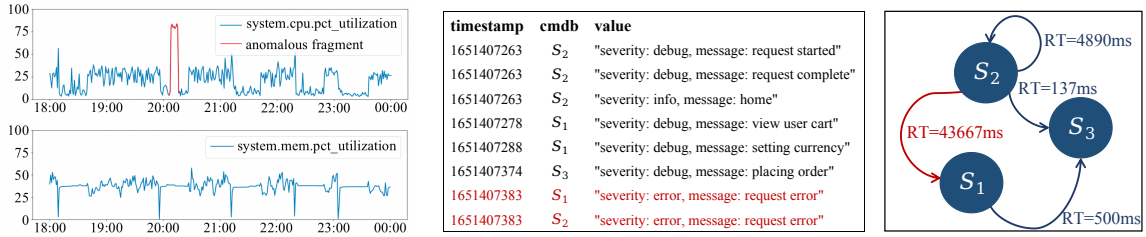
S. Wei is with University of the Chinese Academy of Sciences, Beijing, China. Email: weishirui23@mails.ucas.ac.cn.

C. Zhao is with Alibaba Group, Beijing, China. Email: zhaochenyu@mail.nankai.edu.cn.

B. Zhu is with Nanjing University, Nanjing, China. Email: bolinzhu@smail.nju.edu.cn

L. Pan and Y. Guo are with Huawei Technologies Co., Ltd., Shenzhen, China. Email: {panlemeng, guoyicheng3}@huawei.com.

D. Pei is with Department of Computer Science, Tsinghua University, Beijing, China. Email: peidan@tsinghua.edu.cn.



(a) (b) (c)

Fig. 1: An example of the multimodal data of a microservice system during a failure. Fig. 1(a) shows two metrics: the blue line in the figure represents the original value of the metric, and the red line represents an anomalous fragment. Fig. 1(b) shows eight logs; the red ones are anomalous. Fig. 1(c) shows a trace where a node represents a microservice instance, an edge from a caller to a callee denotes a span, and the red span is anomalous because of its excessively high response time.

TABLE I: Anomalous manifestations of multimodal data are derived from an extensive empirical analysis of failure cases, and mitigation measures for each failure type are based on their underlying root causes [21].

Failure Type	Metric	Log	Trace	Mitigation Measures
CPU resource underprovisioning	✓	-	-	Identify and optimize high CPU-consuming processes
Memory resource underprovisioning	✓	-	-	Identify and optimize high memory-consuming processes
Disk resource underprovisioning	✓	-	-	Identify and optimize high disk-consuming processes
System component damage	✓	-	-	Restart the system or replace the damaged component
Filesystem misconfigurations	-	✓	✓	Check the file path and file permissions, and run filesystem recovery programs
Access control errors	-	✓	✓	Check certificate or credential updates that have only been partially distributed
Container resource quota violations	✓	-	-	Increase the quota and restart the container
Network device breakdown	✓	-	✓	Test network devices and optimize network transport routes allocation
Incorrect data exchange	✓	-	✓	Check error detection and correction mechanisms
Incorrect network configuration	✓	-	-	Check the configuration of the network transport protocols

lization may be lower than normal level, it does not indicate a system failure.

Limitations of Logs. Since logs are semi-structured text generated by logging statements in software source code [17], they rely significantly on developers' domain knowledge. Consequently, certain nuanced performance issues, such as anomalous memory utilization, may not be adequately reflected in the logs, rendering it impossible to diagnose "Resource underprovisioning" failures (Table I).

Limitations of Traces. The information reflecting calls between microservice instances does not provide information about what is happening inside the instance [18]. This lack of information makes it unfeasible to diagnose "System component damage" failures through tracing alone (Table I).

Due to the above limitations, recent studies have explored multimodal data fusion for enhanced failure diagnosis [19], [20]. However, these approaches often fall short of achieving sufficiently deep integration of heterogeneous data sources. This limitation in effectively capturing the complex interdependencies between different modalities consequently hinders diagnostic accuracy. Furthermore, the predominant reliance on supervised learning frameworks in these existing methods introduces a significant dependency on labeled data, restricting their flexibility and generalization capabilities, particularly when confronted with novel failure patterns or evolving system architectures.

Much research has demonstrated that knowledge graph (KG) is a potent technique for combining multimodal information [22], [23]. It can capture entity characteristics for subsequent failure detection and triage. Therefore, we try to apply KG to fuse multimodal data for failure diagnosis (*i.e.*, failure detection and triage). However, applying KG to achieve failure diagnosis through multimodal data faces the following challenges:

1) The immense volume of monitoring data generated by microservices systems [2], makes it challenging to integrate all the data into the KG in the form of entities and relations. Furthermore, as aforementioned, the multimodal data is heterogeneous. The fusion of these heterogeneous data in a KG poses significant challenges.
2) The relations between entities in the KG manifest as a heterogeneous composition, capturing not only the states of distinct events within a given microservice instance but also the intricate interactions and resource-sharing across different instances. Moreover, the temporal variations in instance states and the dynamic fluctuations in the number of instances [24], [25] present notable obstacles in effectively acquiring temporal and heterogeneous structural information from a KG. Furthermore, deriving graph representations from dynamic node embeddings adds to the complexity of the challenge.

In this paper, we propose *UniDiag*, a novel approach for integrating the multimodal data in microservice systems and conducting failure diagnosis. It first fuses multimodal data using a temporal knowledge graph (TKG), which is constructed by temporally stacking KG snapshots to capture system dynamics [26]. This method enables the effective fusion of multimodal data. Subsequently, a novel microservice-oriented graph embedding (MOGE) method is employed to generate graph representations of the TKG, encoding its rich

structural and temporal information. The main contributions of this paper are as follows:

1) *UniDiag* considers the heterogeneity and correlation of three modalities of data (*i.e.*, metrics, logs, and traces) and combines these data with TKG. To the best of our knowledge, we are among the first to apply the KG-based method to fuse the three modalities of data for failure diagnosis.
2) To decrease the number of entities present in KG and fuse the heterogeneous data, we propose a simple yet effective stream-based anomaly detection method, addressing the first challenge.
3) To obtain the comprehensive state representation of a microservice system, we propose a novel MOGE method, integrating the Relation-aware Graph Convolutional Network (R-GCN) [27], Gate Recurrent Unit (GRU) [28] and second-order pooling (SOPOOL) [29], which addresses the second challenge. R-GCN and GRU effectively capture the heterogeneous structural and temporal information of entities and relations in the TKG, respectively. Furthermore, we utilize SOPOOL to learn a comprehensive graph representation of dynamically changing KGs by leveraging the node embeddings.
4) We assess the efficacy and efficiency of *UniDiag* by conducting experiments on two benchmark microservice-generated datasets. Our findings reveal that *UniDiag* achieves online diagnosis within 0.6 seconds and exhibits robust generalization across diverse microservice systems. Furthermore, we observe that *UniDiag* outperforms the baseline methods, as evidenced by an average F_1-score of 0.869 and 0.723 for failure diagnosis on the respective datasets, representing a significant improvement of 0.117 and 0.04 compared to the best-performing baseline methods, respectively. Beyond achieving superior performance on known failure types, *UniDiag* demonstrates promising capabilities for diagnosing previously unseen failure types, highlighting its potential for real-world deployment scenarios. To ensure better reproducibility, we have made our code and data publicly available [30].

II. Related Work

Various methodologies exist for diagnosing failures through metric analysis. For instance, iSQUAD [8] labels root causes and features for different failure clusters, while PatternMatcher [9] employs coarse-grained anomaly detection and a pattern classifier to identify anomalous patterns. DéjàVu [10] trains a model using historical failures and system dependencies.

Log analysis is also pivotal for diagnosing microservice system failures. LogCluster [11] clusters log sequences to determine failure types. Yuan et al. [12] use supervised methods to extract features from anomalous logs and correlate them with failure behaviors, requiring extensive manual labeling.

MEPFL [14] predicts errors and failure types for trace instances based on system trace data. However, methods like those in [31]–[33] using traces for anomaly detection or root cause localization do not specify failure types.

Several studies integrate multimodal data for anomaly detection or root cause localization [18]–[20], [34]–[37]. CloudRCA [20] combines metrics and logs with a Knowledge-informed Hierarchical Bayesian Network for root cause analysis. MicroCBR [34] embeds anomaly event sequences into a failure knowledge graph for case retrieval and failure triage. DiagFusion [19] uses embedding techniques and a Graph Neural Network to determine failure types. Eadro [18] models intra-service dynamics and inter-service dependencies using traces, logs, and KPIs for anomaly detection and root cause localization. TAD [35] employs a Transformer encoder model to identify failures and utilizes tracing chains to pinpoint faulty services. HeMiRCA [36] utilizes Spearman correlation to localize hierarchical root causes. MULAN [37] localizes root causes by co-learning a causal graph from metrics and logs. However, none of these approaches, including Eadro, TAD, HeMiRCA, and MULAN offer comprehensive triage of failure types in microservice systems.

III. Motivation and Problem Statement

A. Motivation

Our goal is to recommend to the operators the occurrence and type of failure based on the latest multimodal data to help operators quickly take the correct mitigation measures. We refer to this as failure diagnosis, which includes failure detection and triage, which facilitates a more comprehensive identification and comprehension of diverse failure types [5], [6], [38]. Table I lists typical failures encountered in microservice systems, showcasing the corresponding manifestations of these failures on multimodal data and the corresponding mitigation measures. The failure types are designated based on their underlying root causes [21]. Nevertheless, the primary objective of our research in this study centers on failure diagnosis rather than the subsequent remediation of failures.

Our examination of 164 microservice system failure cases collected from benchmark microservice systems (see §VI-A for more details) determined that the fusion of multimodal data holds crucial significance in detecting and triaging failures. During failures, anomalies can be observed in metrics [8]–[10], logs [11], [12], and traces [31]. Moreover, different types of failures exhibit distinct anomalous patterns across these multimodal data [18].

1) The Role of Metric in Failure Diagnosis: As shown in Fig. 1(a), the metric data is defined as $\mathbf{x} = \{x_1, x_2, \ldots, x_T\}$, where T is the observation window length and $x_t \in \mathbb{R}$ represents the observation at time t. Metrics are crucial for gauging proper system functioning. When a microservice instance experiences failures, the associated metrics (*e.g.*, memory utilization, disk I/O, CPU utilization, and network throughput) frequently exhibit abnormal patterns. Therefore, operators must remain vigilant in continuously monitoring the metrics.

2) The Role of Log in Failure Diagnosis: As shown in Fig. 1(b), a log message often contains multiple fields, including timestamp, node ID, and detailed information, etc. The detailed information field is unstructured and usually contains constants and variables. The constants are fixed texts the developers set to describe system event entities and relations.

In contrast, the variables carry dynamic runtime information (*e.g.*, IP address, file name, status code, etc.) [17]. Various log parsing algorithms, such as Drain [39] and FT-tree [40], can automatically distinguish the constants and the variables, and generate log templates (constants) and parameters (variables). System states and significant events at various critical points are the main function of logs [41]. Diagnosing microservice system failures can be effectively facilitated by analyzing the diverse anomalous behaviors manifested in logs.

3) The Role of Trace in Failure Diagnosis: The microservice systems encompass a significant volume of asynchronous interactions between instances of microservices, which involve intricate invocation chains [2]. As shown in Fig. 1(c), the mutual invocations between instances can be represented as trace trees, where one invocation request corresponds to one trace. Each span corresponds to one service invocation, having a span ID and a parent ID to reconstruct the calling relations between individual spans [42]. The trace data, which contains response time, status codes, and request frequency, etc., can be crucial indicators for failures [14].

4) Multimodal Data Fusion: For instance, consider a "Network device breakdown" failure may result in increased network packet errors for affected services. While metrics alone might indicate anomalies, identifying the failure type remains challenging without additional contextual information. By integrating metrics with traces, which expose disrupted communication paths between services, operators can more precisely diagnose the underlying network device failure.

In light of the above observations, a novel approach for the fusion of multimodal data to enable effective failure diagnosis has been conceptualized. This has culminated in the development of *UniDiag*.

It is worth noting that in our study, data collection is conducted with distinct granularity levels to capture comprehensive insights. Metrics and logs are collected at the microservice instance level, gaining a detailed understanding of the performance and the events within individual microservice instances, respectively. Meanwhile, traces are collected at the microservices system level, revealing intricate interactions and dependencies among the microservices and facilitating a holistic understanding of the whole system. This multi-faceted analysis, encompassing various levels of data granularity, enables *UniDiag* to deliver a comprehensive assessment of the microservices system and empowers accurate failure diagnosis.

B. Problem Statement

The fine-grained decomposition of microservices makes the system susceptible to performance issues due to highly complex orchestration and dynamic interactions. Failure diagnosis is the core of microservice system maintenance, which enables operators to resolve failures effectively.

Our goal is to recommend to the operators the occurrence and type of failure based on the latest multimodal system monitoring data to help operators quickly take the correct recovery measures. We refer to this as failure diagnosis, which includes failure detection and triage. The detection and triage of failures facilitate a more comprehensive identification and comprehension of diverse failure types, thereby enabling the implementation of suitable remedial actions [5], [6], [38]. For example, metrics, logs, and traces are the most direct signals that characterize underlying failures [43]. Table I lists typical failures encountered in microservice systems, showcasing the corresponding manifestations of these failures on multimodal data and the corresponding remedial measures. The failure types are designated based on their underlying root causes [21]. Nevertheless, the primary objective of our research in this study centers on failure diagnosis rather than the subsequent remediation of failures.

IV. Preliminaries

A. Temporal Knowledge Graph

A TKG incorporates temporal information into a static KG by extending the triple *(h, r, t)* into a quadruple *(h, r, t, τ)*, where *h* represents the head entity, *t* represents the tail entity, *r* represents the relation, τ provides additional temporal information about the fact of the triple, indicating that the fact is valid at time τ. KG can effectively combine multimodal data, and since metrics, logs, and traces change dynamically over time, using TKG allows for the accurate representation of the microservice system's state.

B. R-GCN

To capture the multiple relations in the graph, R-GCN [27] models different relations for the central node separately while adding self-looping features for feature fusion and continuously updating the representation of the central node. In KGs, different edge types convey distinct information, and the direction of edges plays a crucial role in determining the underlying facts. Therefore, in our work, we employ R-GCN to capture not only the states of individual events within a specific microservice instance but also the intricate relations among diverse instances.

C. GRU

Recurrent Neural Network (RNN) [44] is a powerful model for processing sequential data [45]. However, it cannot solve problems such as long-term memory and gradients in back-propagation. Long-Short Term Memory (LSTM) [46] and GRU [28] have been proposed to address this issue. GRU and LSTM are similar in functional performance in many cases [47], but compared to LSTM, GRU has fewer parameters and is more computationally efficient. Therefore, we choose GRU in *UniDiag*.

V. Approach

A. Overview

As shown in Fig. 2, *UniDiag* has two parts: offline training and online diagnosis. In the offline training stage, *UniDiag* consists of three main steps:

1) **Multimodal Temporal Knowledge Graph Construction (§V-B).** To address the first challenge of KG, we introduce a stream-based anomaly detection module. It

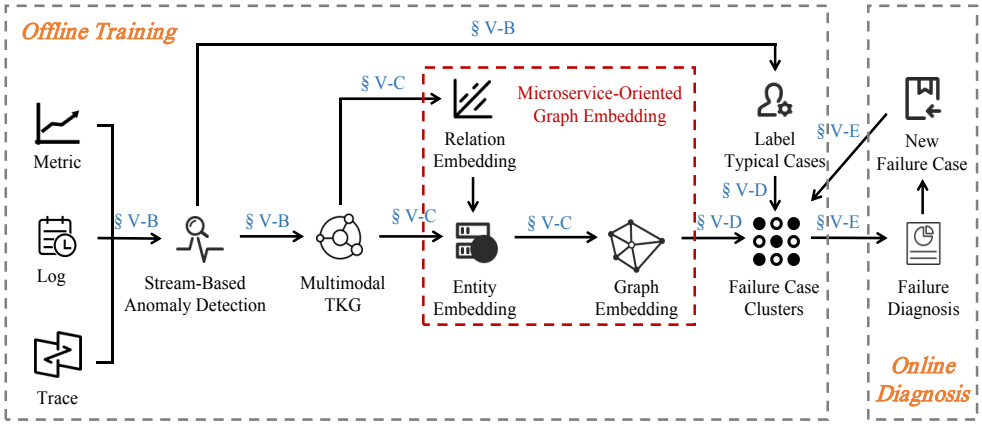


Fig. 2: The framework of *UniDiag*.

first serializes the multimodal data into streams based on their characteristics, and then conducts anomaly detection for each stream. To deal with the dynamically changing data, *UniDiag* builds a TKG based on the topological relations of microservice systems.

2) **MOGE (§V-C).** To address the second challenge, *UniDiag* combines R-GCN, GRU, and SOPOOL. R-GCN leverages its ability to handle multi-relational graphs to learn the structural information within the TKG. GRU, on the other hand, captures long sequence dependencies to learn the temporal information within the TKG. Lastly, SOPOOL pools the TKG to capture high-level features within the graph.
3) **Failure Diagnosis (§V-D).** A hierarchical clustering approach is used to perform failure diagnosis on the learned graph embeddings to obtain a number of clusters. Then, *UniDiag* presents these clusters to operators who investigate and assign corresponding normal or failure types to each cluster.

In the online diagnosis (§V-E) stage, *UniDiag* converts the data into a graph representation vector and diagnoses the failure type based on the trained model (§ V-D).

B. Multimodal Temporal Knowledge Graph Construction

KG representations, characterized by their structured nodes and edges, elucidate the relationships among diverse data sources within microservice systems. This structured approach enhances the comprehension of complex dependencies and facilitates the analysis of failure patterns. Temporal Knowledge Graphs (TKGs) offer significant flexibility and scalability, adeptly capturing the dynamic nature of microservices. Their ability to accommodate updates ensures real-time accuracy, which is essential for effective monitoring in rapidly evolving environments. *UniDiag* applies TKGs to enhance the fusion of heterogeneous multimodal data. The TKGs can be viewed as a series of sub-KGs, *i.e.*, $G = (G_{\tau-k+1}, ..., G_{\tau-1}, G_\tau)$, where G_τ is the KG at time τ. We try to utilize all instances, metrics, and log templates in the microservice system as nodes in G_τ.

Serializing Multimodal Data. As shown on the left side of Fig. 3, we need to serialize multimodal data of each microservice instance to obtain multimodal information at each time point.

1) **Metric Serialization.** Metrics appear in the form of time series with a serialized structure. Therefore, *UniDiag* adopts standard preprocessing steps such as normalization.
2) **Log Serialization.** Logs, as semi-structured text, encapsulate crucial information regarding system events [17]. *UniDiag* employs the Drain algorithm [39] to extract fixed text segments, referred to as log templates, which preserve the core semantic content essential for effective analysis [48]. Utilizing the sliding window technique, *UniDiag* segments the logs and quantifies the occurrence frequency of each log template per minute. These frequencies are subsequently transformed into time series data, aligning with metrics for comprehensive analysis.
3) **Trace Serialization.** *UniDiag* concentrates on three pivotal metrics for traces: response time (RT), error count (EC), and query count (QC) [32]. These metrics are essential for identifying anomalies related to performance, reliability, and traffic [32]. Specifically, to ensure a comprehensive evaluation, we divide the data into one-minute time windows and obtain the values of the three critical metrics within each window. Accordingly, we generate three time series for each tuple <caller, callee>, each reflecting the average response time, query count, and error count, respectively.

Capturing Microservice Dependencies. Understanding the intricate dependencies between microservices is crucial for effective failure diagnosing [19]. These dependencies primarily consist of function calling and resource contention [49]. To accurately model these interdependencies, we integrate traces and deployment data. Specifically, we aggregate trace data to represent the invocation relations between microservices, adding a directed edge in the KG from each caller to its corresponding callee. Additionally, when two instances are co-deployed according to the deployment data, we introduce a directed edge in the KG from the instance to the physical host associated with the deployment data.

Transforming Time Series Data into Events. The intuitive

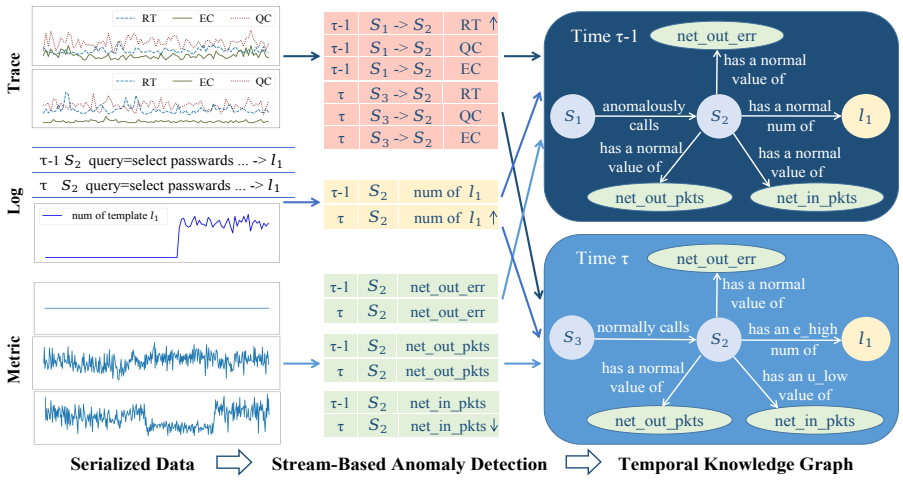


Fig. 3: Multimodal data serialization, stream-based anomaly detection, and construction of TKGs (u_low: unexpected low, e_high: excessively high).

idea is to regard the invocation and deployment relations between instances, the value of metrics data, the number of log templates, and the value of three vital trace-related metrics data as edges. However, this will bring excessively high calculation and storage costs for subsequent tasks, making real-time failure diagnosis impractical. Therefore, *UniDiag* applies a stream-based anomaly detection approach to transform these time series into events and capture the underlying characteristics of the multimodal data. As shown in the middle part of Fig. 3, we adopt the 3-sigma rule to classify the time series data as normal, excessively high, and unexpectedly low every minute, then the type of the edge is set accordingly. Specifically, the edge settings in G_τ are as follows:

The edge of an instance-type object represents whether it is a normal or anomalous call, and this classification depends on whether the current values of the three key trace-related metrics are normal, excessively high, or unexpectedly low. Moreover, the edge of a metric-type (log-template-type) object represents whether the current metric (log) data is normal, excessively high, or unexpectedly low. *UniDiag* stores this information on the edges of the KG.

Constructing the TKG. At time τ, *UniDiag* utilizes the output obtained from stream-based anomaly detection to derive the quadruples required for constructing a TKG. For example, the representation of a quadruple between microservice instance S_1 and metric m_1 is indicated as (S_1, has a normal value of, m_1, τ), whereas the representation of a quadruple between microservice instance S_1 and log template l_1 is denoted as (S_1, has an excessively high number of, l_1, τ). Concerning trace data, we utilize "anomalously calls" and "normally calls" to identify whether the trace-related metrics are anomalous or not. For instance, the quadruple representation for calls between two microservice instances S_1 and S_2 is expressed as (S_1, anomalously calls, S_2, τ). Moreover, the relation between microservice instance S_1 and host H_1 is presented as (S_1, deploys at, H_1, τ). Finally, *UniDiag* builds a comprehensive KG (τ) based on the serialized data and the topological relation between microservice instances. *UniDiag* stacks the KGs of each moment together to form a graph stream (*i.e.*, a TKG).

C. Microservice-Oriented Graph Embedding

To learn the nodes and edges information in TKG, *UniDiag* uses entity and relation embedding. We apply R-GCN and GRU to take into account the structural and temporal information of entities and relations, respectively.

Capture Structural Dependency. Each graph in the TKG contains rich facts, and the interconnection between entities demonstrates a structural dependency. Exploiting the structural dependency enables the central entity to aggregate information from its surrounding neighbors and enhance the learning of entity representations. *UniDiag* uses L-layer relation-aware GCN as an evolutionary unit to model the concurrent facts of structural dependency information. Each entity $e_{t,\tau}$ obtains information from itself, its connected neighboring entities $e_{h,\tau}$, and the relation r at time τ, and passes down to the next layer:

$$\vec{e}^{\,k+1}_{t,\tau} = f\Big(\frac{1}{\delta_t} \sum_{(h,r),\exists(h,r,t)\in\vartheta_\tau} \gamma_1^k(\vec{e}^{\,k}_{h,\tau} + \vec{r}_\tau) + \gamma_2^k \vec{e}^{\,k}_{t,\tau}\Big) \quad (1)$$

where ϑ_τ is the set of facts at timestamp τ, $k \in [0, L-1]$, $\vec{e}^{\,k}_{h,\tau}$, $\vec{r}_\tau$, $\vec{e}^{\,k}_{t,\tau}$ denote the k^{th} layer embeddings of entities h, relation r, and entities t at time τ, respectively. γ_1^k and γ_2^k are the parameters for aggregating features and self-loop in the k^{th} layer. $\vec{e}^{\,k}_{h,\tau} + \vec{r}_t$ represents the relational translation between the head entity and the tail entity under relation r. δ_t is a normalization constant, equal to the in-degree of entity t. $f(\cdot)$ is the RReLU activation function.

Capture Temporal Information. A TKG contains information about the dynamic evolution of KGs over time. *UniDiag* utilizes two temporal gating components to model the sequential information of entities and relations, respectively.

The embedding of all entities at time τ is determined by the final layer output of the relation-aware GCN at time τ and the entity embedding representation at time $\tau - 1$. Formally,

$$E_\tau = U_\tau \circ E_\tau^L + (1 - U_\tau) \circ E_{\tau-1} \quad (2)$$

where $E \in \mathbb{R}^{|\mathcal{E}|*d}$ denotes the entity embedding matrix, $\mathcal{E}$ is the set of entities, and d is the dimension of the embeddings. $\circ$ denotes the Hadamard product. The temporal gate $U_\tau \in \mathbb{R}^{|\mathcal{E}|\times d}$ conducts nonlinear transformation.

The embedding of a relation at time τ is influenced by $E_{\tau-1}$ and $e_{r,\tau}$, which can be formulated as:

$$\vec{r}'_\tau = [pooling(E_{\tau-1}, e_{r,\tau}); \vec{r}] \tag{3}$$

where $\vec{r}$ is the embedding of relation r, $e_{r,\tau}$ is the set of r-related entities, $pooling$ denotes the mean pooling operation, and $[;]$ denotes the vector concatenation operation.

UniDiag uses the GRU component to obtain the embedding of all relations:

$$R_\tau = GRU(R_{\tau-1}, R'_\tau) \tag{4}$$

where $R \in \mathbb{R}^{|\mathcal{R}|*d}$ denotes the relation embedding matrix, and $\mathcal{R}$ is the set of relations.

Our goal is to diagnose failures in the entire microservice system, so we need to learn the graph embeddings of the KG at each moment to perform downstream diagnostic tasks. Because microservice instances are dynamically created and deleted in the microservice system, the deployment and invocation relation also change dynamically [24], [25]. That is, the KGs in the constructed TKG can be different, which brings challenges for subsequent failure detection and triage. Moreover, there is no inherent ordering relation among the nodes in the graph, and we get the same output by using any order of node representation as input.

SOPOOL [29] collects second-order statistics from the information of all nodes. It captures the correlations among features and topology information in graph representation learning [29]. In this way, it can help *UniDiag* address the above challenge. Therefore, *UniDiag* uses SOPOOL to obtain a graph representation based on node representation. *UniDiag* transforms the entity embedding matrix E using a linear mapping to obtain the embedding of the graph:

$$SOPOOL(E) = E^{\mathrm{T}} E\mu \tag{5}$$

where $\mu \in \mathbb{R}^d$ is a trainable vector.

These graph embeddings, encapsulating the rich informational content of three distinct monitoring data (metrics, logs, and traces) gathered during microservice system operation, effectively capture the nuanced patterns embedded within each modality. This comprehensive representation, derived from the fusion of heterogeneous data sources, enables a more discriminating differentiation between various failure types, thereby facilitating increased accuracy in downstream failure diagnosis tasks.

We train the MOGE model with the KG inference task of the TKG. Specifically, the inference is performed on the KG of the next moment based on the KG of the current moment, and a cross-entropy loss is computed according to the result of the inferred and the actual KG of the next moment.

D. Failure Diagnosis

UniDiag trains failure diagnosis models using various failure features. Initially, the number of clusters n is set based on operators' prior knowledge. *UniDiag* then clusters the embedded representations of multimodal TKGs according to n. The farthest distance between cluster centers defines the distance threshold d.

We employ hierarchical agglomerative clustering (HAC) for the following reasons: (1) it uses a "bottom-up" approach to identify hierarchical relations; (2) it is easy to define and flexible with distance similarities and rules; (3) it handles clusters of different sizes; and (4) it is widely used in failure diagnosis [13], [50], [51]. Operators label each cluster with a specific failure type using a typical case, thus reducing annotation effort by labeling only the typical case closest to each cluster center.

E. Online Diagnosis and Update

For a new window's multimodal data, *UniDiag* calculates the distance between the failure's embedding and the existing cluster centers, mapping the failure to the nearest cluster. If the distance is less than d, the failure is classified according to the nearest cluster center. Otherwise, a new cluster is created and labeled by operators, who then update *UniDiag* accordingly. Discrepancies between *UniDiag*'s diagnosis and operators' verification result in cluster set updates, enhancing performance.

UniDiag is initially deployed in a limited number of scenarios and gradually expanded as it stabilizes. Failures identified in one scenario can inform diagnoses in other scenarios, allowing operators to build a comprehensive knowledge base of failure types. This enables quick labeling of new failures, thereby improving diagnosis efficiency and mitigation.

F. Computational Complexity Analysis

To evaluate the computational efficiency of *UniDiag*, we conduct a complexity analysis of two critical components: Multimodal Temporal Knowledge Graph Construction and Microservice-Oriented Graph Embedding. In the Multimodal Temporal Knowledge Graph Construction phase, the time complexity for serializing multimodal data is determined by the number of data entries at each timestamp. Constructing the knowledge graph entails the insertion of nodes and edges. Assuming the number of nodes is $|\mathcal{E}_{all}|$ and the number of edges is $|\mathcal{R}_{all}|$, the complexity is $O(|\mathcal{E}_{all}| + |\mathcal{R}_{all}|)$. In the Microservice-Oriented Graph Embedding phase, the time complexity for capturing structural dependencies is $O(|\mathcal{F}|L)$, where $|\mathcal{F}|$ denotes the maximum number of concurrent facts within the knowledge graph. Additionally, the time complexity for capturing temporal information is $O(|\mathcal{E}_{sub}||\mathcal{R}|)$, where $|\mathcal{E}_{sub}|$ is the maximum number of entities involved in the relationships, and $|\mathcal{R}|$ is the size of the relation set.

VI. Evaluation

We aim to answer the following research questions (RQs):
RQ1: How effective does *UniDiag* perform in failure diagnosis?
RQ2: Does each component contribute to *UniDiag*?
RQ3: Does each data modality contribute to *UniDiag*?
RQ4: Is *UniDiag* computationally efficient?
RQ5: What is the impact of different hyperparameter settings?

TABLE II: Dataset information

Dataset	#Instances	#Failures	#Records	
D1	17	126	metric	3,677,426
			log	1,569,917
			trace	520,674
D2	46	252	metric	349,648
			log	3,048,094
			trace	5,288,134

A. Experimental Setup

Dataset. To evaluate *UniDiag*, we conduct extensive experiments using two datasets, D1 and D2, sourced from distinct microservice systems with varying business contexts and architectures. Table II details the datasets. Both datasets are split into training (the initial 70% of failure cases) and testing (the remaining 30%) sets based on their start time.

Dataset 1 (D1), derived from CloudWise's Generic AIOps Atlas (GAIA)[1], serves as a benchmark for microservice failure diagnosis [19], [52]–[54]. GAIA data, primarily from CloudWise's MicroSS simulation system, encompasses comprehensive monitoring data and simulated failures. We test four types of failures: "Memory resource underprovisioning", "System component damage", "Filesystem misconfigurations", and "Access control errors".

Dataset 2 (D2) originates from a simulated e-commerce application utilizing microservices. It includes a variety of real failure injections to replicate e-commerce operational challenges. This dataset encompasses dynamic service topologies, real-time traces, metrics, and logs. We test five types of failures: "CPU resource underprovisioning", "Memory resource underprovisioning", "Disk resource underprovisioning", "System component damage", and "Container resource quota violations".

Implementation. *UniDiag* is implemented in PyTorch, and all of the experiments are conducted on a Linux Server with 64 16C32T Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, two NVIDIA(R) Tesla(R) V100S, and 187.5 GB RAM. As for the hyperparameters, the number of layers and the hidden size in R-GCN are 2 and 50, respectively. The dimension of graph embedding is 100.

Baselines. We compare *UniDiag* with the following baseline methods: iSQUAD [8], DéjàVu [10], LogCluster [11], Cloud19 [12], MEPFL [14], CloudRCA [20], MicroCBR [34], and DiagFusion [19], which are based on either single-modal or multimodal data. We implement these methods referring to the codes provided by the original papers. For the papers without open-source codes, we set the hyperparameters according to the descriptions provided in the respective papers.

Evaluation Metrics. In the task of failure diagnosis, distinct labels are assigned to each type of failure. Hence, to assess multi-label tasks, we utilize the extensively employed evaluation metrics known as Weighted Average Precision, Weighted Average Recall, and Weighted Average F_1-score [55]. For each label, we perform calculations to determine the respective precision $= TP/(TP+FP)$, recall $= TP/(TP+FN)$, and F_1-score $= 2 \times precision \times recall/(precision + recall)$, with True Positives ($TP$), False Positives ($FP$), and False Negatives ($FN$). Subsequently, these metrics are averaged by applying weighting based on the support, representing the number of true cases associated with each label.

Furthermore, in the ablation experiments, we employ two widely acknowledged metrics, namely Normalized Mutual Information (NMI) [56] and Clustering Accuracy (ACC) [57], to evaluate the performance of *UniDiag*. NMI is a commonly used evaluation metric for clustering results, which measures the consistency between clustering results and ground-truth class labels. Formally,

$$\mathrm{NMI} = -\frac{2 \times \sum_i \sum_j \left(|\mathcal{T}_i \cap \mathcal{Q}_j| \times \log\left(\frac{C \times |\mathcal{T}_i \cap \mathcal{Q}_j|}{|\mathcal{T}_i| \times |\mathcal{Q}_j|}\right)\right)}{\sum_i \left(|\mathcal{T}_i| \times \log\left(\frac{|\mathcal{T}_i|}{C}\right)\right) + \sum_j \left(|\mathcal{Q}_j| \times \log\left(\frac{|\mathcal{Q}_j|}{C}\right)\right)} \tag{6}$$

where C is the total number of cases, $\mathcal{T}_i$ and $\mathcal{Q}_j$ denote the i-th class of the ground truth and the j-th cluster generated by the clustering algorithm, respectively.

ACC elucidates the correspondence between clustering results and ground-truth class labels by quantifying the degree to which each cluster encompasses data points belonging to the respective ground-truth class. ACC can be calculated as follows:

$$\mathrm{ACC}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{C} \sum_{c=1}^{C} \delta\left(x_c = \hat{x}_c\right) \tag{7}$$

where x and $\hat{x}$ denote the actual class labels of the results and the labels that match the ground truth best, respectively. δ denotes the Kronecker delta.

B. Overall Performance (RQ1)

Table III presents the number of labeled samples (failures needing labeling) and the overall performance comparison on the two datasets. The quantity of labeled data for each baseline is tailored to its specific requirements. We empirically determine the optimal number of labeled samples for methods without explicit labeling needs by conducting multiple experiments with varying labeled sample numbers, selecting the best configuration. Notably, the labeled samples are only a subset of the training data, while larger datasets are used for training and testing to ensure comprehensive learning and robust evaluation. A key distinction is that MEPFL [14] labels anomalous traces, unlike other methods that label failure cases. Table III details the number of traces needing labeling for MEPFL. *UniDiag* outperforms all baselines on both datasets, achieving an impressive Weighted Average F_1-score of 0.869 and 0.723, respectively.

iSQUAD [8] diagnoses failures using ground truths from similar failures but lacks historical failure integration, risking errors from noise and fluctuations. DéjàVu [10] trains a model with historical failures and Failure Dependency Graphs (FDGs), requiring many labeled cases. Both methods rely on metric data, lacking comprehensive features, resulting in poor performance. LogCluster [11] struggles to parse log

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

TABLE III: Effectiveness and efficiency of failure diagnosis

Method	D1						D2					
	Labeled Samples	Precision	Recall	F_1-score	Offline Time	Online Time	Labeled Samples	Precision	Recall	F_1-score	Offline Time	Online Time
UniDiag	15	**0.884**	**0.868**	**0.869**	3720.33s	0.32s	20	0.73	**0.724**	**0.723**	4098.57s	0.59s
iSQUAD [8]	**4**	0.642	0.658	0.645	25.09s	<0.01s	**5**	0.702	0.684	0.65	61.81s	<0.01s
DéjàVu [10]	88	0.678	0.684	0.674	118.92s	<0.01s	176	0.647	0.553	0.571	2590.32s	<0.01s
LogCluster [11]	11	0.518	0.579	0.534	0.12s	<0.01s	6	0.603	0.684	0.635	0.33s	<0.01s
Cloud19 (RF) [12]	88	0.74	0.7	0.713	189.00s	0.03s	176	0.601	0.632	0.613	99.00s	0.03s
MEPFL (MLP) [14]	47,022	0.619	0.600	0.599	452.30s	<0.01s	167,468	0.558	0.556	0.417	824.65s	<0.01s
CloudRCA [20]	88	0.676	0.605	0.581	25.88s	0.04s	176	0.855	0.6	0.683	44.56s	0.06s
MicroCBR [34]	20	0.74	0.7	0.713	-	0.10s	45	**0.924**	0.489	0.443	-	0.28s
DiagFusion [19]	88	0.743	0.77	0.752	0.02s	<0.01s	176	0.7	0.663	0.676	0.06s	<0.01s

TABLE IV: Effectiveness of the models dealing with new types of failures

Method	D1			D2		
	Precision	Recall	F_1-score	Precision	Recall	F_1-score
UniDiag	**0.798**	**0.726**	**0.73**	**0.666**	**0.692**	**0.669**
iSQUAD [8]	0.376	0.387	0.345	0.349	0.442	0.366
LogCluster [11]	0.449	0.533	0.462	0.487	0.566	0.509

entries into meaningful events, missing semantic information. Cloud19 [12] fails to address polysemy and uses a limited context window, missing broader contextual information and leading to suboptimal performance.

MEPFL [14] uses a supervised learning model through traces, relying on labeling all anomalous trace patterns.

CloudRCA [20] combines logs and metrics but ignores the microservice system's topology, missing crucial features. It also requires many labeled historical failure cases, and insufficient training samples harm its performance. In contrast, *UniDiag* excels in failure diagnosis, achieving a significantly higher Weighted Average F_1-score than existing methods.

UniDiag demonstrates superior performance in failure diagnosis compared to MicroCBR [34] and DiagFusion [19] by integrating both normal and failure data, thereby capturing a holistic view of the system state and mitigating information loss. Unlike MicroCBR, which is constrained by its reliance on failure data alone, and DiagFusion, which depends heavily on labeled historical cases, *UniDiag* employs TKG and MOGE to provide a comprehensive temporal representation of the microservice system's state, effectively modeling complex dependencies and interactions.

RT and invocation paths reflect anomalous patterns in trace data, while EC and QC are key for detecting reliability and traffic failures. MicroCBR uses only RT for anomaly detection, and DiagFusion uses RT and EC, potentially leading to inaccurate diagnoses. In contrast, *UniDiag* uses invocation paths, RT, EC, and QC to construct a knowledge graph, incorporating rich information for more accurate failure diagnosis.

Frequent failure scenarios in historical data can be swiftly and accurately identified through observation and practice. However, as the microservice architecture evolves and develops, new types of failures, that is, undiscovered types of failures that have never occurred or occurred less frequently in the past, often manifest within the microservice systems.

To evaluate *UniDiag*'s robustness in diagnosing new failure types, we select distinct failure types from diverse datasets and excluded associated samples from the training set. Given that supervised methods cannot diagnose new failure types and MicroCBR can only match new cases with existing types, we used iSQUAD and LogCluster as baselines. As listed in Table IV, by averaging the results of each new failure type diagnosis on the same dataset, we calculate the precision, recall, and F_1-score. *UniDiag* significantly outperforms the other methods. iSQUAD and LogCluster, limited to single-modal data, fail to detect some failures, resulting in numerous false positives and negatives. In contrast, *UniDiag*'s fusion of three modalities of monitoring data captures unique features from each modality, leading to comprehensive representation learning for each failure type. This demonstrates that multi-modal data fusion enhances *UniDiag*'s performance.

C. Contribution of Key Components (RQ2)

To show the effectiveness of *UniDiag*'s key techniques (*i.e.*, , entity embedding, graph embedding, and clustering), we creat five variants (C1-C5) by substituting these techniques with common or state-of-the-art alternatives: (1) **C1** uses RE-NET [58] instead of MOGE for learning entity embedding. (2) **C2** uses TANGO [59] instead of MOGE for learning entity embedding. (3) **C3** uses mean pooling instead of MOGE for learning graph embedding. (4) **C4** uses DBSCAN instead of HAC. (5) **C5** uses K-means instead of HAC.

Table V demonstrates that *UniDiag* outperforms all variants on two datasets, underscoring the significance of each component. Substituting MOGE with RE-NET (C1) or TANGO (C2) results in decreased performance. RE-NET's use of subgraph aggregators and GRUs neglects the structural dependencies and static attributes in TKGs, leading to the potential loss or mixing of historical facts. TANGO's Neural Ordinary Differential Equations (NODE) [60] are better suited for continuous time domains and fail to capture patterns in the discrete snapshots of TKGs amidst frequent microservice changes.

TABLE V: Experimental results of ablation study

Dataset	Method	NMI	ACC	Precision	Recall	F_1-score
D1	***UniDiag***	**0.783**	**0.868**	**0.884**	**0.868**	**0.869**
	C1	0.538	0.658	0.567	0.632	0.587
	C2	0.634	0.737	0.699	0.684	0.687
	C3	0.764	0.789	0.728	0.737	0.73
	C4	0.663	0.658	0.521	0.632	0.547
	C5	0.759	0.816	0.781	0.763	0.763
D2	***UniDiag***	0.602	**0.75**	**0.73**	**0.724**	**0.723**
	C1	0.368	0.566	0.392	0.171	0.203
	C2	0.573	0.697	0.583	0.671	0.617
	C3	0.558	0.684	0.655	0.671	0.662
	C4	**0.651**	0.724	0.413	0.461	0.416
	C5	0.538	0.671	0.679	0.618	0.638

Replacing MOGE with mean pooling (C3) degrades performance as traditional pooling methods (e.g., mean pooling) cannot effectively characterize features. SOPOOL, by modeling general graph information correlations, captures long-distance feature information more effectively.

Substituting HAC with DBSCAN or K-means (C4 & C5) also reduces performance. HAC is robust to random initial values and can identify outliers in KGs. DBSCAN, a density-based method, is insensitive to noise and outliers, making it unsuitable for failure detection. K-means struggles with non-spherical and varied-sized clusters, and the choice of initial cluster centers significantly affects the results.

D. Contribution of Each Modality (RQ3)

We conduct ablation experiments to examine the contributions of different data sources by creating the following variants: (1) ***UniDiag* w/o $\mathcal{M}$** drops metrics while inputs logs and traces. (2) ***UniDiag* w/o $\mathcal{L}$** drops logs while inputs metrics and traces. (3) ***UniDiag* w/o $\mathcal{T}$** drops traces while inputs metrics and logs.

TABLE VI: Contributions of diverse data sources

Dataset	Method	NMI	ACC	Precision	Recall	F_1-score
D1	***UniDiag***	**0.783**	**0.868**	**0.884**	**0.868**	**0.869**
	UniDiag w/o $\mathcal{M}$	0.733	0.763	0.701	0.737	0.696
	UniDiag w/o $\mathcal{L}$	0.717	0.737	0.607	0.737	0.65
	UniDiag w/o $\mathcal{T}$	0.675	0.737	0.753	0.737	0.718
D2	***UniDiag***	**0.602**	**0.75**	**0.73**	**0.724**	**0.723**
	UniDiag w/o $\mathcal{M}$	0.536	0.684	0.57	0.579	0.569
	UniDiag w/o $\mathcal{L}$	0.524	0.645	0.688	0.539	0.582
	UniDiag w/o $\mathcal{T}$	0.574	0.711	0.715	0.684	0.693

Table VI indicates that *UniDiag*'s effectiveness depends on the integration of all data modalities, with each contributing differently. Traces contribute the least in D1 and D2, as indicated by the minor performance drop in *UniDiag* w/o $\mathcal{T}$, due to limited trace records and fluctuating RT and QC. Conversely, *UniDiag* w/o $\mathcal{M}$ shows significant performance degradation, aligning with observations in Table I.

In conclusion, integrating all three data modalities significantly enhances *UniDiag*'s effectiveness, highlighting the crucial role of diverse data in improving failure diagnosis.

E. Computational Efficiency (RQ4)

Table III compares the running times of *UniDiag* with other baseline methods, covering both offline training and online diagnosis durations. Notably, *UniDiag* achieves an average online diagnosis time of 0.6s, meeting real-time diagnostic demands, given that data collection intervals for D1 and D2 are at least 30s. However, *UniDiag* has longer offline training times due to the entity embedding process, which is influenced by the size and complexity of the KG and the GCN. Despite this, *UniDiag* minimizes the need for frequent retraining.

To test *UniDiag*'s efficiency on larger datasets, we constructed a KG with 2,553 entities and 3,807,420 edges by replicating nodes and attributes from D1. *UniDiag* achieved an average offline training time of 5.12h and an average online diagnosis time of 4.73s, demonstrating suitability for real-world deployment. These results confirm *UniDiag*'s robust performance on large-scale datasets.

F. Effect of Hyperparameters (RQ5)

We discuss the impact of *UniDiag*'s three hyperparameters and different training set sizes. Fig. 4 shows how NMI, ACC, and Weighted Average F_1-score vary with these parameters.

The Number of Layers in Entity Embedding. *UniDiag*'s performance declines as the number of GCN layers increases from 1 to 5, with optimal results achieved at 2 or 3 layers. An excessive number of layers causes node representations to converge and lose distinction.

Graph Embedding Dimension. Optimal performance is observed at a graph embedding dimension of 100. Larger dimensions risk sparsity, over-fitting, and increased computational complexity, but they may be suitable for complex microservice systems.

The Number of Clusters. Increasing the number of clusters enhances the utilization of training data; however, too many clusters raise the labeling effort. We select 15 and 20 clusters for D1 and D2, respectively.

The Size of the Training Set. We evaluated *UniDiag* with training set sizes of 30%, 40%, 50%, 60%, and 70%, using the remaining 30% for testing. Fig. 4 demonstrates that *UniDiag* achieves a high Weighted Average F_1-score even with a limited number of failure cases, proving its effectiveness in real-world scenarios with constrained historical data.

VII. Discussion

A. Lessons Learned

1) Data Collection Granularity: Currently, most IT enterprises collect data at minute-level granularity, typically every 1 or 5 minutes [61], [62]. This granularity generally does not significantly impact the model's effectiveness. In our approach, once the metric granularity is set, logs and traces must be aligned with the metrics.

2) Effectiveness: UniDiag demonstrates superior effectiveness in failure diagnosis by integrating comprehensive multi-modal data and leveraging TKGs to capture the full system state over time. Unlike other methods that rely on limited data sources, insufficient labeled data, or simplistic models, *UniDiag* incorporates a broader range of features, which ensures accurate diagnosis in microservice systems.

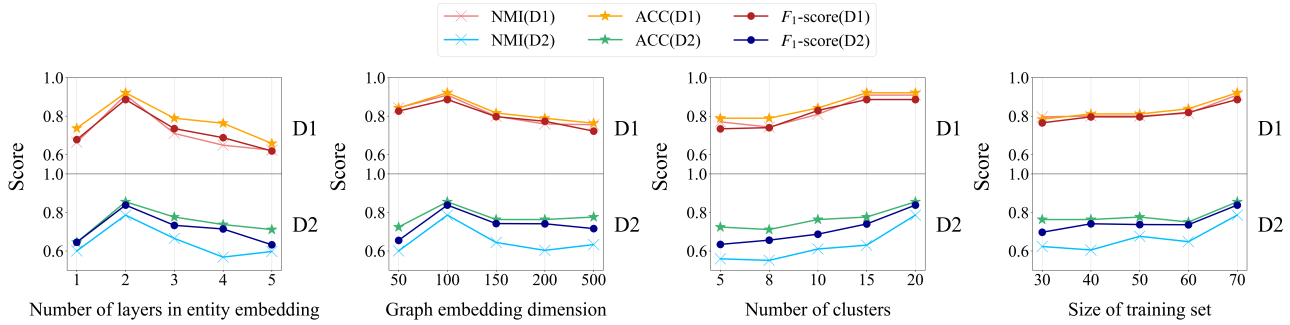


Fig. 4: Hyperparameters sensitivity on *UniDiag*.

3) Flexibility: *UniDiag*'s flexibility is driven by its dynamic TKG updates and incremental learning. The system continuously adapts to new data, integrating evolving failure patterns and architectural changes without requiring full retraining, ensuring it remains effective in diagnosing emerging failures as the environment evolves.

4) Scalability: *UniDiag* exhibits a high degree of adaptability to the dynamic nature of microservice architectures. As microservice instances are created or destroyed, *UniDiag* requires only the addition or removal of the corresponding entities and relations within the TKG. The workflow of *UniDiag* is inherently general and does not rely on any specific data modality, thereby maintaining its effectiveness even in production environments where not all three modalities of data are concurrently monitored.

B. Threats to Validity

Regarding internal validity threats, the primary concern is the implementation process. To address this, we conducted repeated experimental tests and iteratively optimized the approach. The results presented are averages of multiple trials.

Externally, there are two primary threats. First, the limited scale of our datasets compared to real-world microservice systems in production environments restricts generalizability. These datasets might not fully represent all microservice systems. Nevertheless, we believe our approach is sufficiently general. We plan to collaborate with a top-tier Internet service provider to deploy *UniDiag* and validate its effectiveness in real-world scenarios. Second, our datasets may not capture all failure types compared to industry-collected data. Future work will involve evaluating our approach on a broader range of microservice systems, incorporating larger-scale real-world cases with diverse failure types to address this limitation.

C. Limitations

Based on the conducted experiments, we draw attention to two primary limitations of *UniDiag*, as follows:

1) While *UniDiag* is designed to leverage metrics, logs, and traces for comprehensive failure diagnosis, it remains robust even when data from some modalities are unavailable. This is crucial since acquiring synchronized multimodal data can be challenging in certain microservice deployments. Although *UniDiag* can function with missing data sources due to its loosely-coupled nature, providing all data types maximizes its effectiveness. Open-source toolkits such as cAdvisor [63] for metrics, Elasticsearch [64] for logs, and Jaeger [65] for tracing facilitate monitoring data collection in microservice systems. These tools simplify equipping microservices with comprehensive data collection capabilities [18].
2) *UniDiag* relies on a data-driven approach. If specific failures do not manifest anomalies in the collected data, accurate diagnosis becomes challenging. This issue is common and also recognized in MEPFL [14] and SCWarn [16].

VIII. CONCLUSION

Recognizing the limitations of relying solely on single-modal data, which may fail to capture a substantial amount of crucial information, we present *UniDiag* as a novel approach for failure diagnosis. *UniDiag* fuses multimodal data, including metrics, logs, and traces, using TKG. It includes a three-step process: (1) effective fusion of metrics, logs, and traces through TKGs, (2) utilization of MOGE for entity embedding and graph embedding, and (3) training of the failure diagnosis model to detect failures and identify failure types accurately. We have conducted extensive evaluation experiments on two datasets to validate the efficacy, efficiency, and generalization of *UniDiag*. The empirical findings affirm the robustness and effectiveness of *UniDiag* in achieving accurate and efficient failure diagnosis. Moreover, to overcome the limitations of the data-driven approach, a contextual analysis module could effectively analyze external factors, such as deployment changes and system updates, which might provide additional clues about potential failures for more accurate failure diagnosis.

ACKNOWLEDGMENTS

This work is supported by the Advanced Research Project of China (No. 31511010501), and the National Natural Science Foundation of China (62272249, 62302244, 62072264).

REFERENCES

[1] T. Colanzi, A. Amaral, W. Assunccao, A. Zavadski, D. Tanno, A. Garcia, and C. Lucena, "Are we speaking the industry language? the practice and literature of modernizing legacy systems with microservices," in *15th Brazilian Symposium on Software Components, Architectures, and Reuse*. New York, NY: ACM, 2021, pp. 61–70.

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

[3] A. Mahimkar, C. E. de Andrade, R. Sinha, and G. Rana, "A composition framework for change management," in *Proceedings of the 2021 ACM SIGCOMM Conference*. New York, NY: ACM, 2021, pp. 788–806.

[4] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang, "Cyrus: Towards client-defined cloud storage," in *Proceedings of the Tenth European Conference on Computer Systems*. New York, NY: ACM, 2015, pp. 1–16.

[5] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Los Alamitos, CA: IEEE, 2019, pp. 111–120.

[6] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Los Alamitos, CA: IEEE, 2019, pp. 364–375.

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

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

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

[10] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui *et al.*, "Actionable and interpretable fault localization for recurring failures in online service systems," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY: ACM, 2022, pp. 996–1008.

[11] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. Los Alamitos, CA: IEEE, 2016, pp. 102–111.

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

[13] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin *et al.*, "Onion: identifying incident-indicating logs for cloud systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY: ACM, 2021, pp. 1253–1263.

[14] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY: ACM, 2019, pp. 683–694.

[15] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, "Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA: IEEE, 2023, pp. 1724–1736.

[16] N. Zhao, J. Chen, Z. Yu, H. Wang, J. Li, B. Qiu, H. Xu, W. Zhang, K. Sui, and D. Pei, "Identifying bad software changes via multimodal anomaly detection for online service systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY: ACM, 2021, pp. 527–539.

[17] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[18] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "Eadro: An end-to-end troubleshooting framework for microservices on multi-source data," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA: IEEE, 2023, pp. 1750–1762.

[19] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin, D. Zhang, Z. Zhu, and D. Pei, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Transactions on Services Computing*, 2023.

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

[21] P. Dogga, K. Narasimhan, A. Sivaraman, S. Saini, G. Varghese, and R. Netravali, "Revelio: Ml-generated debugging queries for finding root causes in distributed systems," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 601–622, 2022.

[22] C. Deng, Y. Jia, H. Xu, C. Zhang, J. Tang, L. Fu, W. Zhang, H. Zhang, X. Wang, and C. Zhou, "Gakg: A multimodal geoscience academic knowledge graph," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. New York, NY: ACM, 2021, pp. 4445–4454.

[23] A. V. Kannan, D. Fradkin, I. Akrotirianakis, T. Kulahcioglu, A. Canedo, A. Roy, S.-Y. Yu, M. Arnav, and M. A. Al Faruque, "Multimodal knowledge graph for deep learning papers and code," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. New York, NY: ACM, 2020, pp. 3417–3420.

[24] X. Li, Y. Chen, and Z. Lin, "Towards automated inter-service authorization for microservice applications," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. New York, NY: ACM, 2019, pp. 3–5.

[25] S. Chakraborty, S. Garg, S. Agarwal, A. Chauhan, and S. K. Saini, "Causil: Causal graph for instance level microservice data," in *Proceedings of the ACM Web Conference 2023*. New York, NY: ACM, 2023, pp. 2905–2915.

[26] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs," in *Proceedings of the 34th International Conference on Machine Learning*. New York, NY: PMLR, 2017, pp. 3462–3471.

[27] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European semantic web conference*. Berlin: Springer-Verlag, 2018, pp. 593–607.

[28] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. Stroudsburg,PA: Association for Computational Linguistics, 2014, p. 1724–1734.

[29] Z. Wang and S. Ji, "Second-order pooling for graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 6870–6880, 2020.

[30] UniDiag, "Open source repository of unidiag," 2024. [Online]. Available: https://github.com/AIOps-Lab-NKU/UniDiag.git

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

[32] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, "Microhecl: High-efficient root cause localization in large-scale microservice systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Los Alamitos, CA: IEEE, 2021, pp. 338–347.

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

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

[35] J. Wang, Y. Li, Q. Qi, Y. Lu, and B. Wu, "Multilayered fault detection and localization with transformer for microservice systems," *IEEE Transactions on Reliability*, pp. 1–14, 2024.

[36] Z. Zhu, C. Lee, X. Tang, and P. He, "Hemirca: Fine-grained root cause analysis for microservices with heterogeneous data sources," *ACM Trans. Softw. Eng. Methodol.*, Jul 2024, just Accepted.
[37] L. Zheng, Z. Chen, J. He, and H. Chen, "Mulan: Multi-modal causal structure learning and root cause analysis for microservice systems," in *Proceedings of the ACM on Web Conference 2024*, ser. WWW '24. New York, NY, USA: ACM, 2024, p. 4107–4116.
[38] A. Workshop, "2022 ccf aiops challenge," 2022. [Online]. Available: https://workshop.aiops.org
[39] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. Los Alamitos, CA: IEEE, 2017, pp. 33–40.
[40] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu, D. Pei, J. Xu, Y. Chen, H. Dong, X. Qu, and L. Song, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. Los Alamitos, CA: IEEE, 2017, pp. 1–10.
[41] J. Kulkarni, S. Joshi, S. Bapat, and K. Jambhali, "Analysis of system logs for pattern detection and anomaly prediction," in *Proceeding of International Conference on Computational Science and Applications: ICCSA 2019*. Berlin: Springer-Verlag, 2020, pp. 427–436.
[42] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Inc., Tech. Rep., 2010.
[43] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu, "Adaptive performance anomaly detection for online service systems via pattern sketching," in *Proceedings of the 44th International Conference on Software Engineering*. New York, NY: ACM, 2022, pp. 61–72.
[44] S. Kim, S. An, P. Chikontwe, and S. H. Park, "Bidirectional rnn-based few shot learning for 3d medical image segmentation," in *Proceedings of the AAAI conference on artificial intelligence*. Palo Alto, CA: AAAI, 2021, pp. 1808–1816.
[45] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.
[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[47] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.
[48] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang *et al.*, "An empirical investigation of practical log anomaly detection for online service systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY: ACM, 2021, pp. 1404–1415.
[49] Y. Wang, G. Li, Z. Wang, Y. Kang, Y. Zhou, H. Zhang, F. Gao, J. Sun, L. Yang, P. Lee, Z. Xu, P. Zhao, B. Qiao, L. Li, X. Zhang, and Q. Lin, "Fast outage analysis of large-scale production clouds with service correlation mining," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Los Alamitos, CA: IEEE, 2021, pp. 885–896.
[50] Y. Sun, D. Cheng, T. Yang, Y. Ji, S. Zhang, M. Zhu, X. Xiong, Q. Fan, M. Liang, D. Pei *et al.*, "Efficient and robust kpi outlier detection for large-scale datacenters," *IEEE Transactions on Computers*, vol. 72, no. 10, pp. 2858–2871, 2023.
[51] Y. Fu, M. Yan, Z. Xu, X. Xia, X. Zhang, and D. Yang, "An empirical study of the impact of log parsers on the performance of log-based anomaly detection," *Empirical Software Engineering*, vol. 28, no. 1, p. 6, 2023.
[52] Y. Sui, Y. Zhang, J. Sun, T. Xu, S. Zhang, Z. Li, Y. Sun, F. Guo, J. Shen, Y. Zhang *et al.*, "Logkg: Log failure diagnosis through knowledge graph," *IEEE Transactions on Services Computing*, 2023.
[53] S. Zhang, Z. Pan, H. Liu, P. Jin, Y. Sun, Q. Ouyang, J. Wang, X. Jia, Y. Zhang, H. Yang *et al.*, "Efficient and robust trace anomaly detection for large-scale microservice systems," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Los Alamitos, CA: IEEE, 2023, pp. 69–79.
[54] C. Zhao, M. Ma, Z. Zhong, S. Zhang, Z. Tan, X. Xiong, L. Yu, J. Feng, Y. Sun, Y. Zhang *et al.*, "Robust multimodal failure detection for microservice systems," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. New York, NY: ACM, 2023, pp. 5639–5649.
[55] B. M. Sundheim, "Overview of the fourth message understanding evaluation and conference," NAVAL COMMAND CONTROL AND OCEAN SURVEILLANCE CENTER RDT AND E DIV SAN DIEGO CA, Tech. Rep., 1992.
[56] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of statistical mechanics: Theory and experiment*, vol. 2005, no. 09, p. P09008, 2005.
[57] N. Zhao, L. Zhang, B. Du, Q. Zhang, J. You, and D. Tao, "Robust dual clustering with adaptive manifold regularization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2498–2509, 2017.
[58] W. Jin, M. Qu, X. Jin, and X. Ren, "Recurrent event network: Autoregressive structure inferenceover temporal knowledge graphs," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Stroudsburg,PA: Association for Computational Linguistics, nov 2020, pp. 6669–6683.
[59] Z. Han, Z. Ding, Y. Ma, Y. Gu, and V. Tresp, "Learning neural ordinary equations for forecasting future links on temporal knowledge graphs," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Stroudsburg,PA: Association for Computational Linguistics, nov 2021, pp. 8352–8364.
[60] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning," in *International Conference on Learning Representations*. OpenReview.net, 2018.
[61] Z. Li, Y. Zhao, R. Liu, and D. Pei, "Robust and rapid clustering of kpis for large-scale anomaly detection," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. Los Alamitos, CA: IEEE, 2018, pp. 1–10.
[62] Z. Li, W. Chen, and D. Pei, "Robust and unsupervised kpi anomaly detection based on conditional variational autoencoder," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. Los Alamitos, CA: IEEE, 2018, pp. 1–9.
[63] Google, "Container advisor," 2024. [Online]. Available: https://github.com/google/cadvisor
[64] Elastic, "Elasticsearch," 2024. [Online]. Available: https://www.elastic.co/
[65] C. N. C. Foundation., "Jaeger," 2024. [Online]. Available: https://www.jaegertracing.io/

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 the Department of Computer Science and Technology, 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.

Yongxin Zhao received her B.S. and M.S. degrees in software engineering from the College of Software, Nankai University, Tianjin, China, in 2021 and 2024, respectively. She is currently a Ph.D. student at the College of Software, Nankai University, Tianjin, China. Her research interests include failure detection and failure diagnosis.

Sibo Xia received his B.S. degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2022. He is currently a Ph.D. student at the College of Software, Nankai University. His research interests include anomaly detection and failure diagnosis.

Shirui Wei received her B.S. in software engineering from the College of Software, Nankai University, Tianjin, China, in 2023. She is currently a master student at the National Astronomical Observatories, Chinese Academy of Sciences, Beijing, China. Her research interests include astroinformatics and virtual observatories.

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 the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2018. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His research interests include anomaly detection and root cause localization in service management.

Chenyu Zhao received her B.S. and M.S. degrees in software engineering from the College of Software, Nankai University, Tianjin, China, in 2020 and 2023, respectively. Her research interests are focused on anomaly detection and root cause analysis.

Shiyu Ma received her B.S. degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2022. She is currently a master's student at the College of Software, Nankai University. Her research interests include failure diagnosis and root-cause localization.

Junhua Kuang received his B.S. degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2024. He is currently a master's student at the College of Software, Nankai University. His research interests include anomaly detection and anomaly localization.

Bolin Zhu received his B.S. in software engineering from the College of Software, Nankai University, Tianjin, China, in 2023. He is currently a master's student in the Computer Science and Technology Department, Nanjing University. His research interests include the reasoning of large language models.

Lemeng Pan received his Ph.D. in Statistics from the University of Maryland College Park, Maryland, USA. He is currently a research scientist at Huawei Technologies Co., Ltd.

Yicheng Guo received his B.S. degree in particle and nuclear physics from University of Science and Technology of China in 2014, and Ph.D. degree in particle and nuclear physics from the Department of Modern Physics, University of Science and Technology of China, Hefei, China, in 2019. He is currently a senior engineer at Huawei Technologies Co., Ltd.

Dan Pei received the B.E. and M.S. degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, 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 at Tsinghua University, Beijing, China. His research interests include network and service management in general. He is an IEEE senior member and an ACM senior member.