Giving Every Modality a Voice in Microservice Failure Diagnosis via Multimodal Adaptive Optimization

Lei Tao
Nankai University
Tianjin, China

Shenglin Zhang
Nankai University & HL-IT
Tianjin, China

Zedong Jia
Nankai University
Tianjin, China

Jinrui Sun
Nankai University
Tianjin, China

Minghua Ma
Microsoft
Redmond, USA

Zhengdan Li*
Nankai University
Tianjin, China

Yongqian Sun
Nankai University & TKL-SEHCI
Tianjin, China

Canqun Yang
National University of Defense Technology & NSCC-TJ
Changsha, China

Yuzhi Zhang
Nankai University
Tianjin, China

Dan Pei
Tsinghua University & BNRist
Beijing, China

ABSTRACT

Microservice systems are inherently complex and prone to failures, which can significantly impact user experience. Existing diagnostic approaches based on single-modal data such as logs, metrics, or traces cannot comprehensively capture failure patterns. For those multimodal data-based failure diagnosis methods, the dominant modality can overshadow others, hindering low-yield modalities from fully leveraging their characteristics. This paper proposes *Medicine*, a modal-independent microservice failure diagnosis framework based on multimodal adaptive optimization. It encodes different modalities separately to retain their unique features and employs adaptive optimization to adjust the learning pace between modalities, thereby enhancing overall diagnostic performance. Experimental results demonstrate that *Medicine* outperforms existing single-modal and multimodal diagnostic approaches on three public datasets, with F1-score improving by 15.72% to 70.84%. Even in cases where individual modal data is missing or of lower quality, *Medicine* maintains high diagnostic accuracy.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

KEYWORDS

Microservice System, Failure Diagnosis, Multimodal Adaptive Optimization

*Zhengdan Li is the corresponding author. Email: lzd@nankai.edu.cn
HL-IT, TKL-SEHCI, NSCC-TJ, and BNRist are short for Key Laboratory of Data and Intelligent System Security, Ministry of Education, Tianjin Key Laboratory of Software Experience and Human Computer Interaction, National SuperComputer Center in Tianjin, and Beijing National Research Center for Information Science and Technology, respectively.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10…$15.00
https://doi.org/10.1145/3691620.3695489

ACM Reference Format:
Lei Tao, Shenglin Zhang, Zedong Jia, Jinrui Sun, Minghua Ma, Zhengdan Li, Yongqian Sun, Canqun Yang, Yuzhi Zhang, and Dan Pei. 2024. Giving Every Modality a Voice in Microservice Failure Diagnosis via Multimodal Adaptive Optimization. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3695489

1 INTRODUCTION

The microservice architecture decomposes applications into independently deployable services, each handling specific business functions, offering high flexibility and maintainability [1]. However, its complexity and dynamism pose reliability and availability challenges [2, 3]. Failures in one service can propagate across the interdependent services, significantly impacting user experience and causing substantial losses if not promptly diagnosed and remedied [4]. For instance, on June 13, 2023, due to a potential software defect, a disruption in the AWS Lambda service, which is implemented using a microservices architecture [5], affected several other AWS services, leading to nearly four hours of downtime and significant economic losses [6].

Rapidly diagnosing the type of failure in a microservice system helps operators trace and implement appropriate remedies. Failures in microservice systems typically include hardware failure (e.g., intensive workload, resource exhaustion), software failure (e.g., system bottleneck, code bugs), and network problem (e.g., network exception, packet loss) [7–10]. Accurately diagnosing the type of failure aids in more effectively identifying the root cause of issues, leading to faster resolution and reduced downtime. Understanding the types of frequently occurring failures allows for proactive maintenance planning, preventing recurrence and enhancing system reliability [11, 12].

Collected metrics, logs, and traces in microservice systems can be utilized to detect and diagnose failures [13]. Metrics are time series data reflecting business status and machine performance. Logs are unstructured text outputs from program executions, and traces connect service invocation information into a tree-like structure during a business call. Single-modality failure diagnosis approaches (e.g., metrics [9, 14], logs [10, 15], or traces [16]) in microservice systems face significant limitations due to their restricted focus on

Corrected Version of Record. V.1.1. Published May 7, 2025.

a single data type. For instance, log data might indicate a series of minor errors that do not impact overall system performance, leading to false alarms. Conversely, actual failures might go unnoticed if they do not produce significant anomalies in the chosen data type. Relying solely on performance metrics can result in missing critical issues such as "Security breaches" or "Network misconfigurations," which may only be evident in other data forms. Consequently, single-modality approaches are prone to both false positives and false negatives, undermining the reliability and comprehensiveness of failure diagnosis. Figure 1 illustrates a failure example in a microservice system, with the red-marked sections indicating how the failure manifests in three different modalities of data. Compared to single-modality data, multimodal observability data can offer a more comprehensive insight into the system's state.

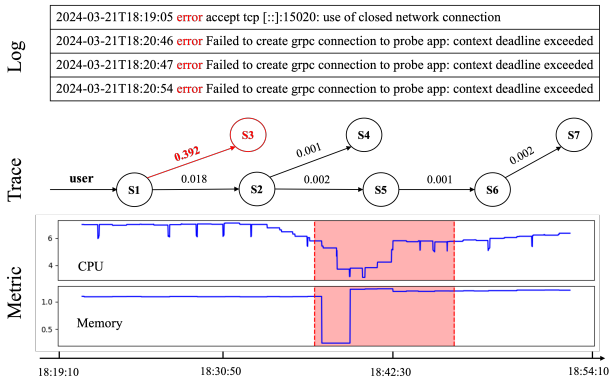

Figure 1: A failure in a microservice system as reflected in multimodal observability data. The red markings indicate anomalies detected across the three modalities.

Significant efforts have been made to diagnose microservice system failures using multiple modalities of data [17–26]. After practicing and analyzing these multimodal data-based failure diagnosis approaches, we identified three challenges: ***1) Inconsistent data formats***: To simplify data processing and capture complementary information among different modalities, [17–21] have mapped multimodal data into a unified representation space. However, they overlook inter-modality differences and disrupt the structure of information stored within each modality. ***2) Incomplete and low-quality data***: existing approaches [17, 18, 20, 22–24] often assume a scenario with complete and high-quality multimodal data, which is not always feasible, leading to potential performance degradation when the data is incomplete or of low quality. ***3) Interference in modality optimization***: Previous multimodal failure diagnosis approaches have mostly been based on deep learning. Training a multimodal data-based failure diagnosis model with unified learning objectives using the same training strategy may not be globally optimal due to different training convergence rates across modalities [21, 22, 25, 26]. During training, high-yield modalities with better performance tend to suppress the optimization of other modalities. As a result, low-yield modalities cannot fully utilize their features due to interference from other modalities.

In this work, we propose *Medicine*, a modal-independent microservice failure diagnosis framework based on multimodal adaptive optimization. It consists of three stages: *1) Feature Encoding*: Each modality of data is encoded separately. *2) Modality Fusion*: Encoded features are fused, retaining initial features for comprehensive representation. *3) Optimization Balancing*: An adaptive optimization module adjusts convergence speeds across modalities to enhance overall performance. Specifically, to tackle the first challenge, we design specific data processing, feature encoding, and classifiers tailored to the knowledge stored in metrics, logs, and traces, respectively. For the second challenge, we use a parallel stream structure and modal modulation to treat each modality equally, significantly reducing dependency on any single modality. For the third challenge, we balance the optimization process during training by suppressing gradients for high-yield modalities and enhancing features for low-yield ones based on modal evaluation results.

We evaluated *Medicine*'s performance on three widely used public datasets collected from benchmark microservice systems. The experimental results demonstrate that *Medicine* achieves comparable or superior failure classification performance compared to state-of-the-art single-modal or multimodal failure diagnosis approaches. Specifically, *Medicine* improves F1-score by 41.49% to 93.90% compared to single-modal approaches and by 15.72% to 70.84% compared to other multimodal approaches.

In summary, the main contributions of this paper are:

- We design specific feature encoders for different modalities of observability data based on their unique characteristics and the types of failures they reflect, enhancing the ability to comprehensively capture knowledge within each modality.
- We introduce *Medicine*, the first parallel microservice failure diagnosis framework that achieves modal independence through multimodal adaptive optimization, reducing reliance on individual modalities. The implementation is publicly available at [28].
- Our designed module, combining modal evaluation, gradient suppression, and feature enhancement, maximizes the potential of multimodal data by adapting to different learning paces. Extensive experiments prove *Medicine*'s superiority over existing multimodal failure diagnosis approaches for microservice systems.

2 BACKGROUND AND RELATED WORK

2.1 Background

Microservice failure diagnosis refers to the process of identifying, localizing, and analyzing the causes of service interruptions or performance degradation within a microservice system. Failure classification in microservices involves recognizing and categorizing various types of failures, which aids operators in more effectively identifying, diagnosing, and resolving issues in microservice systems. Common types of failures in microservices systems are shown in Table 1. Failure classification in microservice systems enhances failure diagnosis and resolution efficiency, leading to quicker issue resolution and reduced downtime. It also improves system reliability and user satisfaction by enabling proactive maintenance, resource optimization, and better system design.

To monitor the operational state of microservice systems in real-time, operators continuously collect observability data, including logs, metrics, and traces. Logs, denoted as $\mathcal{L}$, are semi-structured

Table 1: Failure types in microservice systems and their different manifestations in multimodal data. [27]

Failure Type	Metric	Log	Trace	Remedial Measures
Resource Underprovisioning	✓	-	-	Implement auto-scaling to adjust resources based on demand
Hardware Damage	✓	-	-	Use redundancy and failover systems to ensure continuous operation
Database Query Failure	-	✓	✓	Optimize queries and implement indexing to prevent failures
Login Failure	-	✓	✓	Implement multi-factor authentication to enhance login security
Network Congestion	✓	-	✓	Apply traffic shaping to prioritize critical network traffic efficiently
Code Bugs	✓	-	✓	Use automated testing to detect and fix bugs early
System Misconfigurations	✓	✓	✓	Employ configuration management tools to standardize settings

data that record system information, user behavior, and business information during the operation of a microservice system. They facilitate various system management and diagnostic tasks. Metrics are typically time series data collected at regular intervals [29]. A metric can be defined as $\mathcal{M} = \{x_1, x_2, ..., x_T\}$, where T is the length of the metric and $x_t \in \mathbb{R}$ represents the observed value at time t. Metrics provide the most straightforward depiction of resource usage in a microservice system and can also reflect performance-related issues. When a service within a microservice system is requested by a user, multiple calls (spans) may be generated [30]. These spans form a trace, denoted as $\mathcal{T}$, which includes call status and response information.

The goal of failure classification is to use data from $\mathcal{L}$, $\mathcal{M}$, and $\mathcal{T}$ to predict the presence of failures and determine their types. The failure type is represented by y, where $y \in \{1, 2, ..., N\}$ indicates N possible failure types.

2.2 Related Work

2.2.1 Single-modal Failure Diagnosis. Early single-modal failure diagnosis approaches typically rely on statistical or rule-based approaches. LOGAN [31] constructs a reference model for each request type, comparing current logs to this model to identify failures. LADRA [32] uses custom log feature correlations to determine failure probabilities. DBSherlock [33] identifies potential failure types and confidence levels based on visualized performance metrics and user-specified abnormal instances. FPDB [34] transforms trace data into processing streams, storing these with failure information and using similarity matching for diagnosis.

With the rise of machine learning, effective approaches like LogCluster [10], Cloud19 [15], Déjàvu [14], iSQUAD [9] and MEPFL [16] have emerged. LogCluster [10] clusters logs to extract representative failure sequences and matches them during detection. Cloud19 [15] diagnoses failures by extracting and classifying exception logs related to critical tasks using a pre-trained classifier for each task. Déjàvu [14] maps the metrics of failure instances within the same failure category into fixed-width vectors and uses a graph neural network (GNN) based on the failure dependency graph to diagnose root causes and failure types. iSQUAD [9] diagnoses the root causes of intermittent slow queries by clustering and labeling them offline, then matches new queries to these clusters based on similarity. MEPFL [16] learns behavior patterns from trace data to build a random forest model, classifying real-time trace instances during detection.

2.2.2 Multimodal Failure Diagnosis. Single-modal failure diagnosis can often fail to fully capture the anomalies caused by failures, leading to suboptimal diagnostic outcomes [18]. To overcome this limitation, researchers have developed multimodal data-based approaches for more effective failure diagnosis [13, 17–26, 35, 36]. These approaches can be categorized into feature fusion, model fusion, and result fusion based on the stage at which the multimodal integration occurs.

Feature fusion involves processing multimodal data to extract a unified feature matrix or event representation as the model input, with the model outputting the final diagnostic result. For example, CloudRCA [20] integrates anomalous metric sequences and log patterns into a unified feature matrix, then uses a knowledge-informed hierarchical Bayesian network for diagnosis. DiagFusion [17] converts multimodal data into a unified event representation and employs a trained event embedding model and GNN to determine failure types. Model fusion extracts different features from multimodal data and combines them within a model capable of handling such data. Groot [23] and TrinityRCL [24], for instance, build causal graphs from multimodal data and use graph theory models to integrate information for failure diagnosis. MicroCBR [22] integrates multimodal data into a knowledge graph and inputs anomalies to the graph for failure reporting. Result fusion involves specific analysis of the stage results from single-modality data. For example, PDiagnose [13] performs separate anomaly detection on each modality and uses a voting mechanism to identify the root cause service. Wang et al. [36] proposed reflecting system anomaly levels through log anomaly scores and using mutual information to calculate the correlation between log anomaly scores and metrics, thereby pinpointing the root cause of the failure.

3 MOTIVATION AND CHALLENGES

3.1 Motivation

3.1.1 Enhancing Diagnostic Accuracy through Multimodal Data Integration. Microservice systems generate diverse types of operational data (metrics, logs, traces), each offering unique insights. Diagnosing failures based solely on single-modal data can result in a high incidence of false positives. Abnormal fluctuations in single-modal

data do not necessarily indicate a system failure [25, 37]. For instance, network fluctuations in a microservices system might cause temporary spikes or dips in related metrics that eventually stabilize. Moreover, system failures are not always captured by the monitoring data of a single modality. As demonstrated in Table 1, focusing exclusively on metric data could lead to overlooking critical issues such as "Login failure" and "Database Query Failure" in a microservices system. To maximize diagnostic accuracy, integrating these different modalities is essential, despite their varied formats and information encoding methods. This approach allows for a more holistic view of system health, capturing complex interdependencies and failure characteristics that single-modal approaches might miss.

3.1.2 Ensuring Robustness in Real-World Scenarios. Real-world microservice systems often face scenarios with incomplete and low-quality data. Developing a failure diagnosis framework that remains effective despite missing or poor-quality data is crucial. By leveraging multimodal data, the framework can maintain robust performance, ensuring reliable service continuity and minimizing downtime, even in suboptimal monitoring conditions. To achieve this, a framework that adaptively optimizes and balances different data modalities is essential, as it ensures robust failure diagnosis and effective operation in dynamic environments.

3.2 Challenges and Solutions

3.2.1 Inconsistent Data Formats. Microservice systems generate diverse operational data, such as metrics, logs, and traces, each with distinct formats and methods of encapsulating information. Unifying these multimodal data formats can disrupt the inherent structure and storage methods unique to each modality. For instance, metrics are typically time-series data reflecting system performance, logs are unstructured text detailing system events, and traces are structured data representing service interactions. Harmonizing these different formats without losing critical information is a significant challenge. To address this issue, we design specific data processing, feature encoding, and classifiers tailored to the granular knowledge stored in metrics, logs, and traces, reflecting different failure types. Consider a system where metric anomalies indicate CPU spikes, logs detail error messages, and traces show delayed service interactions. By designing information statistics paradigms and encoding each data type separately, we preserve their unique information, enabling precise failure diagnosis. For more details, see Section 4.1, 4.2, and 4.3.

3.2.2 Incomplete and Low-Quality Data. In real-world microservice environments, the completeness and quality of multimodal data are often lacking. Clear architecture, precise service calls, comprehensive metrics, and standardized log information are rare. Missing or low-quality data from any modality can lead to substantial performance degradation in multimodal failure diagnosis approaches. For example, if logs are incomplete or metrics are noisy, the diagnosis framework may fail to accurately detect and diagnose failures, resulting in prolonged downtime and user dissatisfaction. To address this challenge, we use a parallel stream structure and modal modulation to treat each modality equally. Notably, treating each modality equally does not mean giving each modality the same weight. The parallel stream structure decouples feature extraction and fusion for each modality, and multimodal adaptive optimization reduces dependence on any single modality while enhancing the utilization of other modalities. This design minimizes the impact on the final diagnostic results in scenarios with incomplete or low-quality data, thereby enhancing the robustness and resilience of the failure diagnosis system.

3.2.3 Interference in Modality Optimization. During the training phase of multimodal models, high-yield modalities (e.g., those with better performance metrics) tend to dominate the optimization process. As shown in Figure 2, the metric modality is the dominant modality that is easier to train and optimize in a microservice system. The dominant modality varies depending on the microservice system. This dominance suppresses the optimization of other modalities (e.g., the log and trace modality in Figure 2), preventing them from fully utilizing their features. For example, a model might overfit to metric data because it provides more immediate performance feedback while neglecting the valuable insights from logs and traces. Balancing the optimization process across all modalities to ensure comprehensive utilization and prevent interference is a critical challenge. Hence, we balance the optimization process during training by suppressing gradients for high-yield modalities and enhancing features for low-yield ones based on modal evaluation results. For more details, see Section 4.5.

By addressing these challenges with tailored solutions, we enhance the robustness and accuracy of multimodal failure diagnosis in microservice systems.

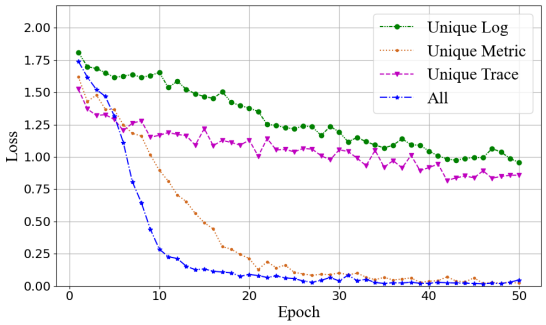

Figure 2: Loss curves of single-modality classifiers in a microservice system. The dominant modality (e.g., metric) may suppress the optimization of other modalities, preventing them from fully utilizing their features.

4 METHODOLOGY

Figure 3 presents the overview of *Medicine*, a multimodal failure diagnosis framework for microservice systems. The framework consists of three main stages: feature encoding, modality fusion, and optimization balancing.

4.1 Log Encoder

We first use statistical methods and the BERT [38] to obtain log representations for each time window in the failure interval. Then, we train our log-specific model using these log representations to obtain a high-level representation of log modality. Log encoding consists of two steps.

1) Log Filtering and Template Fusion: First, we use Drain[39] to derive structured log template data from unstructured log data.

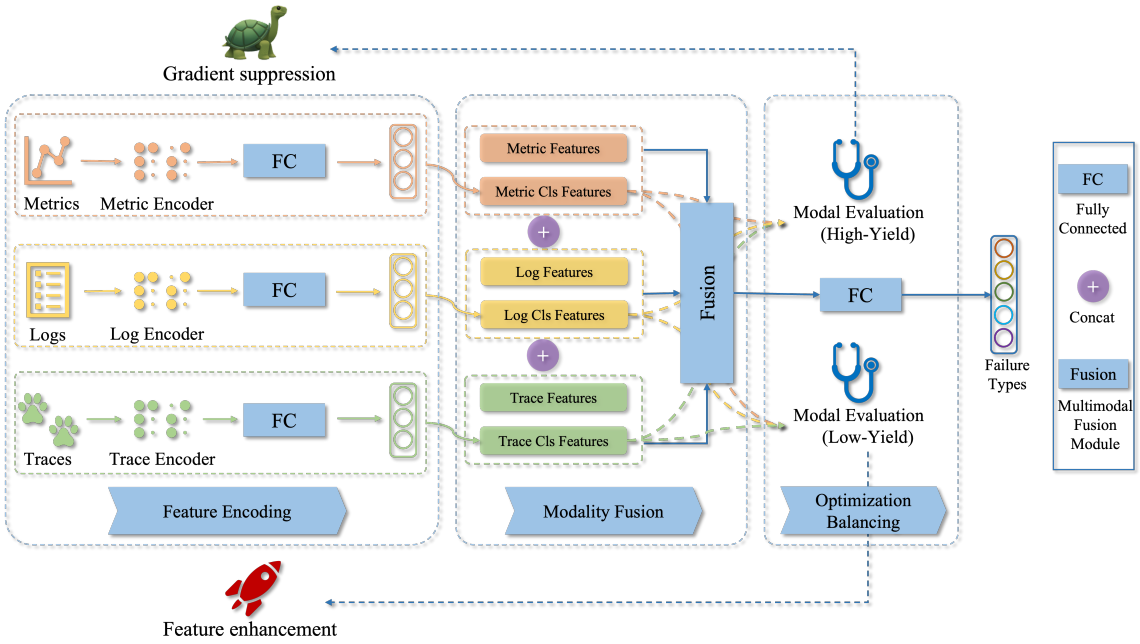

Figure 3: The framework of *Medicine*.

Our analysis of the log data revealed that within the failure interval, there are varying degrees of increase or decrease in the number of log templates, as illustrated in Figure 4. Log templates with a large quantity and small fluctuation, such as Figure 4 (a) and Figure 4 (b), carry less of a message for failure classification. In contrast, templates with sudden changes or significant increases or decreases in quantity, like those in Figure 4 (c) and Figure 4 (d), provide critical information. To capture abnormal information of log templates, we assign higher weight to log templates with obvious fluctuating quantities and lower weights to stable and unchanging ones, for example, in Figure 4, template0 is assigned 0.128 while template3 is assigned 5.117. Based on this, we perform a statistical analysis of the log quantity in each time window within the failure interval to determine the weight of each log template. The following formula calculates the increment (decrement) of each log template in the corresponding window:

$$c'_{i,j} = \begin{cases} |\log_2(c_{i,j}+\epsilon) - \log_2(c_{i,j-1}+\epsilon)|, & 1 < j \le m \\ 0, & j = 1 \end{cases} \tag{1}$$

where $c_{i,j}$ is the occurrence count of the i-th template at time j. The following formula calculates the weight of each log template:

$$w_i = \max_{1 \le j \le m} c'_{i,j} - \overline{c'_i} \tag{2}$$

$$w_i = \frac{w_i}{\sum_{k=1}^{n} w_k} \tag{3}$$

where w_i is the weight of the i-th template. Inspired by NeuralLog[40], we use BERT [38] to obtain the semantic vectors for each log template:

$$h_i = \text{BERT}(t_i) \tag{4}$$

To obtain the log representation of a time window while reducing training costs, we sum the vectors in this window using their corresponding weights:

$$r_j = \sum_{i=1}^{n} w_i h_i \tag{5}$$

Thus, a failure interval should have a log representation sequence $R = \{r_j \mid 1 \le j \le m\}$.

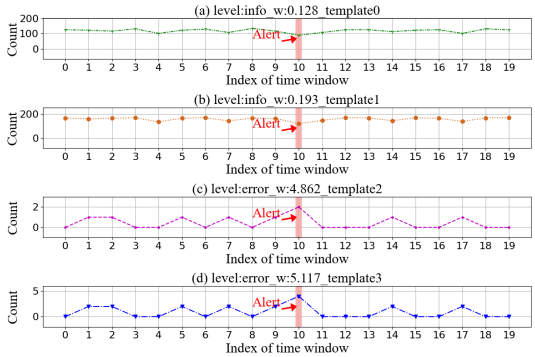

Figure 4: The number of different log templates changes under the access deny failure condition. The level denotes the log level, and w denotes the weight of the template.

2) Log Modality Fusion: We leverage the advantages of the transformer model [41] for processing sequential data, including its ability to model long-distance dependencies and effectively capture both local and global information within sequences, thus better understanding the key information and patterns in log data. Additionally, we utilize global pooling to aggregate, compress and summarize information from the entire sequence, extracting a global feature representation of the sequence.

4.2 Metric Encoder

We extract features from different categories of metrics for each time window within the failure interval. Then, we utilize the extracted features to train our metric-specific model to obtain a high-level representation of metric modality. Metric encoding involves two steps.

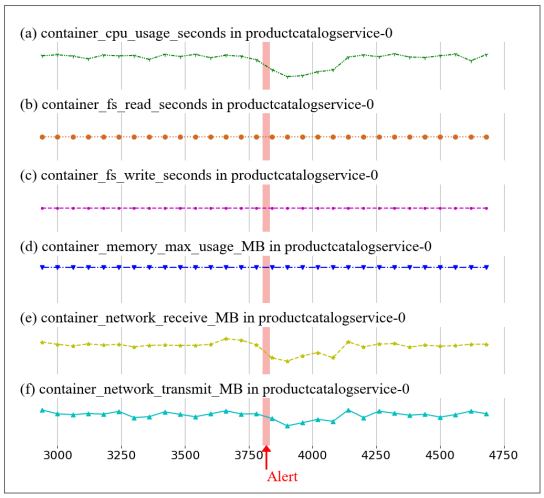

Figure 5: The manifestation of network failures in selected metrics of the productcatalogservice-0

1) Feature Selection and Processing: We find that specific failures often manifest as concentrated anomalies in a few metric categories while appearing normal in others. For example, in Figure 5, there are obvious manifestations in network and CPU related metrics, while fewer in memory and file system related metrics. Based on this, we consider the set of metric categories $M = \{M_i | 1 \leq i \leq f\}$ as the feature set characterizing the failure interval, which can effectively distinguish between different failures. To ensure comparability across metric categories and instances within the same metric, we standardize each feature of each instance using the following formula:

$$x_i' = \frac{n(x_i - \overline{x})}{\sqrt{\sum_{k=1}^{n} (x_k - \overline{x})^2}} \tag{6}$$

Additionally, certain features display trends. To remove these trends and enhance the distinction between abnormal and normal data, we perform first-order differencing:

$$x_i'' = \begin{cases} x_i' - x_{i-1}', & i > 1 \\ 0, & i = 1 \end{cases} \tag{7}$$

In a failure occurrence interval, we compute the average metric value for each instance and metric category within each time window. Thus, for a representation $R = \{F_k | 1 \leq k \leq n\}$ of a failure interval, an instance F_k has such a feature sequence $F_k = \{t_j | 1 \leq j \leq m\}$, where t_j represents the feature set of the j-th time window, expressed as $t_j = \{\overline{m}_{j,i} | 1 \leq i \leq f\}$.

2) Metric Modality Fusion: When designing the metric encoder, we've identified two critical issues. we've observed that the same failure can occur across different instances. For example, Figure 6 and Figure 7 show two cases where the same network failure occurred at different times, with clear manifestations on container_network_receive_MB. Secondly, when a failure occurs, it can have a cascading effect, influencing other instances to varying degrees. As shown in Figure 7, although adservice-0 and frontend-1 are not the root causes of the failure, they still exhibit clear manifestations on network_receive_MB.

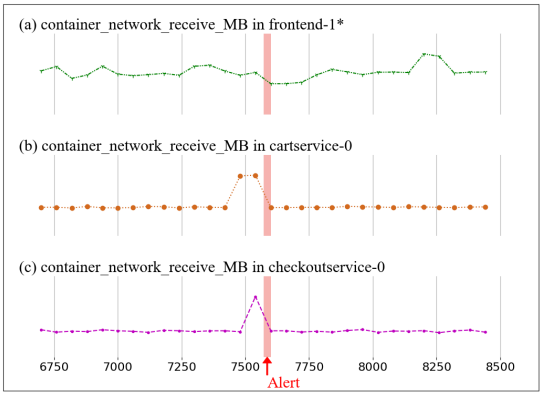

Figure 6: The manifestation of network failure in the container_network_receive_MB of root cause frontend-1 and two nonroot cause.

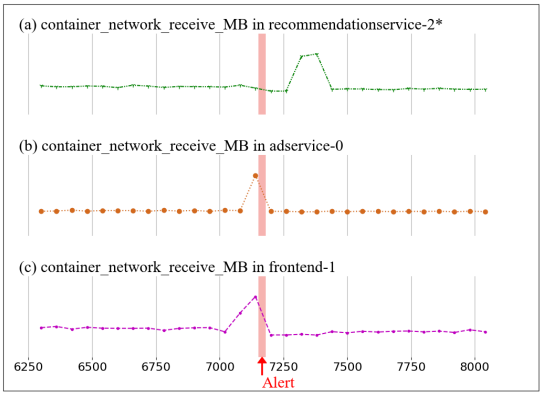

Figure 7: The manifestation of network failure in the container_network_receive_MB of root cause recommendationservice-2 and two nonroot cause.

Based on this, inspired by SENet[42], we treat each instance as a separate channel and deploy a channel attention mechanism [42], which concentrates and compresses the failure information contained in all instances, enabling the extracted features to represent the entire microservice system's characterization following a failure to the fullest extent possible. This method effectively addresses the two previously mentioned issues.

Then, we utilize a transformer model [41] and global pooling to aggregate feature information from the time series into a fixed-length vector, extracting a comprehensive global feature representation.

4.3 Trace Encoder

We extract features from the duration data of different types of spans within the time window of each failure case. Then, we use

the obtained trace representations to train on the model we designed for trace to obtain the high-level representation of trace modality. Trace encoding involves two steps.

1) Feature Selection and Processing: To better extract information from Trace data, we consider the combination of the calling service and the called service as a type of span, thus categorizing multiple types of spans. To eliminate differences in duration across different types of spans, we standardize the duration for each type of span.

$$x_i' = \frac{n(x_i - \overline{x})}{\sqrt{\sum_{k=1}^{n}(x_k - \overline{x})^2}} \tag{8}$$

We detect anomalies for each type of invocation using the 3-sigma method and calculate the anomaly score S to reflect the anomaly situation of this type of span. The specific formula is as follows, where n represents the number of points greater than 3.

$$\mathrm{S} = \begin{cases} \frac{\sum_{x_i>3} x_i}{n} & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases} \tag{9}$$

We analyzed the trace data and found that the number of anomalies for each type of span fluctuates differently under various cases. As shown in Figure 8, four types of spans and their corresponding weights are marked in the figure.

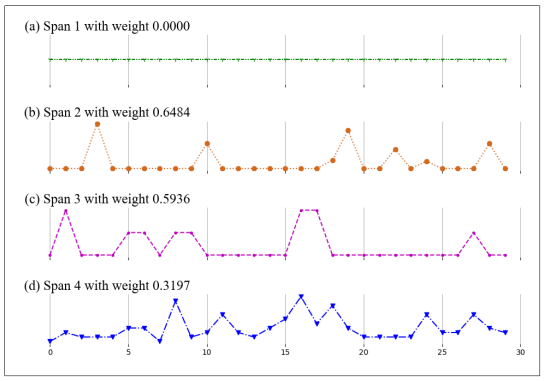

Figure 8: The number of anomalies for spans under different failure cases

For a type of span, if it frequently switches between normal and abnormal states (e.g., Span2 in Figure 8 (b)) or has a large fluctuation in the number of anomalies (e.g., Span3 in Figure 8 (c)), then its importance for failure classification is higher. If it remains consistently normal or abnormal with small fluctuations in the number of anomalies (e.g., Span1 and Span4 in Figure 8), then its importance for failure classification is lower. To increase the weight of the former and decrease the weight of the latter, we use the same statistical analysis method as the log encoder to assign appropriate weights to each type of span.

After determining the weights that should be assigned to each type of call, we use the product of the weight and the anomaly score as the feature for that type of span within the time window. In this way, for a failure interval, we can obtain a sequence of features.

2) Trace Modality Fusion: To obtain an overall understanding of anomaly situations from sequences of features derived from different types of spans, we utilize the Encoder layer of the Transformer [41]. We employ its multi-head self-attention mechanism [41] to capture internal relationships among different types of spans, facilitating a better comprehension and learning of crucial information and patterns within. Subsequently, we employ a linear classifier to compress and summarize the information of the entire feature sequence, thereby extracting a global feature representation of the sequence.

4.4 Multimodal Fusion

In the modality fusion stage, we employ a multimodal fusion module with channel attention [42] to integrate the original statistical features from each modality with the failure classification features extracted by the feature encoders. This module is designed to effectively combine information from different modalities, ensuring that the unique characteristics of each modality are preserved and enhanced. The fusion module performs the following steps:

1) Feature Concatenation: The input features from different modalities (x, y, and z) are concatenated along the feature dimension. This concatenated feature vector is then passed through a fully connected layer [43] to generate an integrated feature representation fc_{out}.

2) Modality-specific Linear Transformations: Each modality's features are separately processed through individual linear layers ($linear_{x_{out}}$, $linear_{y_{out}}$, and $linear_{z_{out}}$). These transformations help to capture modality-specific information and enhance the discriminative power of each modality.

3) Channel Attention Mechanism: The outputs of linear transformations are passed through a sigmoid activation function to produce attention weights ($\sigma(x_{out})$, $\sigma(y_{out})$, and $\sigma(z_{out})$). These weights are used to highlight important features in each modality.

4) Feature Stacking and Squeezing: The original modality features and the attention-weighted features are stacked together, creating a comprehensive feature set. This stacked feature set is then processed using an adaptive average pooling layer to reduce the dimensionality and focus on the most informative features.

5) Classification: The pooled feature representation is passed through a fully connected layer [43] to perform failure classification. This layer outputs the predicted class labels for the input data.

4.5 Multimodal Adaptive Optimization

The multimodal adaptive optimization module (MAO) is designed to address the challenges of inconsistent convergence rates and mutual interference among different modalities in multimodal failure diagnosis. This module comprises three key components: modality evaluation, gradient suppression, and feature enhancement. These components work together to dynamically adjust the training process, ensuring that each modality contributes effectively to the overall model performance. The following introduces these three key components:

Modality Evaluation: Modality evaluation is the initial step in the adaptive optimization process. In an ideal real-world scenario, it is preferable for the reduction in training loss (the difference in training loss between iterations) to be gradual, while the reduction in validation loss should be rapid. It assesses the contribution of each modality to the learning objective by calculating the ratio of

validation loss reduction to training loss reduction:

$$\rho_k = \frac{L_k^V(n-1) - L_k^V(n)}{L_k^T(n-1) - L_k^T(n)} \tag{10}$$

This ratio ρ_k indicates the generalization performance of modality k. The contributions of all modalities are then normalized to ensure their coefficients θ_k sum equals one:

$$\theta_k = \frac{e^{\rho_k}}{\sum_{j=1}^{K} e^{\rho_j}} \tag{11}$$

This evaluation helps to identify dominant (high-yield) and underperforming (low-yield) modalities, providing a foundation for subsequent optimization steps.

Gradient Suppression: The gradient suppression component dynamically adjusts the learning rates of different modalities based on their evaluated contributions. For the dominant modality (the one with the highest θ_k), the gradient is suppressed to prevent it from overwhelming other modalities:

$$s_t^k = \begin{cases} 1 - \alpha \cdot \theta_k & \text{if } k = \arg\max(\theta_t^k) \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

Here, α is a hyperparameter controlling the degree of suppression, and s_t^k represents the suppression factor for the gradients. This approach ensures that the dominant modality does not excessively influence the training process, allowing underperforming modalities to catch up. The network parameters are updated as follows [44, 45]:

$$\omega_{t+1}^k = \omega_t^k - \eta \cdot s_t^k \tilde{g}(\omega_t^k) \tag{13}$$

In this formula, ω_t is the current parameter value at iteration t, and ω_{t+1} is the updated parameter value for the next iteration. The term η represents the learning rate, which controls the step size of the update. Finally, $\tilde{g}$ is the gradient of the loss function with respect to the parameter ω_t, guiding the update direction. This update rule ensures that high-performing modalities do not dominate the optimization process, allowing for more effective and balanced training across all modalities.

Feature Enhancement: To compensate for the lower contribution of underperforming (low-yield) modalities, the feature enhancement component boosts the features of these modalities. The scaling factor for the weakest modality, determined by the lowest evaluation score, is calculated as:

$$s_t^k = \begin{cases} \beta \cdot \theta_k & \text{if } k = \arg\min(\theta_t^k) \\ 1 & \text{otherwise} \end{cases} \tag{14}$$

This formula determines the scaling factor s_t^k for the lowest-performing (low-Yield) modality k at iteration t. Here, β is a scaling parameter. For all other modalities, the scaling factor s_t^k is set to 1, indicating no change in their feature values. The enhanced feature representation is given by [42]:

$$\widetilde{\mathbf{x}}_t^k = \mathbf{F}_{scale}(\mathbf{u}_t^k, s_t^k) = s_t^k \cdot \mathbf{u}_t^k \tag{15}$$

In this formula, $\widetilde{\mathbf{x}}_t^k$ represents the enhanced feature vector for modality k. The term $\mathbf{u}_t^k$ is the original feature vector of the modality. The function $\mathbf{F}_{scale}$ scales the feature vector $\mathbf{u}_t^k$ by the factor s_t^k, effectively enhancing the features of low-performing (low-Yield) modalities. This enhancement helps accelerate the learning process for these modalities, ensuring a more balanced and effective optimization across different types of data.

5 EXPERIMENT

We conduct a variety of experimental studies to answer the following research questions.

RQ1: How well does *Medicine* perform in microservice system failure diagnosis?

RQ2: Does each component contribute to *Medicine*?

RQ3: How do the major parameters of *Medicine* influence its performance?

5.1 Experimental Design

Table 2: Summary of selected multi-modal datasets

	# Instances	# Failure cases	Modality	#
D1	10	174	Log	26,035,183
			Metric	18,497,325
			Trace	44,858,388
D2	40	1099	Log	80,113,843
			Metric	117,411,233
			Trace	26,064,740
D3	9	119	Log	18,665,646
			Metric	50,284,800
			Trace	30,709,790

5.1.1 Datasets. To evaluate the performance of *Medicine*, we conduct extensive experiments on three microservice systems (forming Dataset 1, Dataset 2, and Dataset 3, respectively). Table 2 lists the detailed information of these three datasets. The second column indicates the number of instances of each dataset. The third column indicates the number of failures injected into each dataset. The fourth and fifth columns indicate the number of each modality in each dataset.

- **Dataset 1 (D1)** is collected from a large-scale simulated e-commerce application system operated by a top-tier global commercial bank, which is a microservices architecture [46]. It injects a variety of real failures to simulate the operational challenges faced by e-commerce companies when dealing with a massive of business data. This dataset includes the dynamic topology of application services, real-time traces, metrics, and logs.
- **Dataset 2 (D2)** is the Generic AIOps Atlas (GAIA) dataset from CloudWise [47]. It contains the multimodal data collected from a business simulation system of Cloudwise. This dataset provides records of all injected failures that may occur in real systems to facilitate fair algorithm evaluation, and data of all relevant entities, traces, logs, and metrics.
- **Dataset 3 (D3)** is collected from a recently released microservice benchmark, MicroServo [48], which deploys the open-source Online Boutique [49] provided by GoogleCloudPlatform, employs three collectors to gather metric, log, and trace data separately and employs chaos engineering techniques [50] to simulate the occurrence of real failures. This dataset provides data of logs, metrics, and traces, and abundant fault injection records.

Table 3: The precision, recall and F1-score of different approaches on different datasets.

Approach	Modality			D1			D2			D3		
	Metric	Log	Trace	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
DéjàVu [14]	✓			0.4569	0.5526	0.4972	0.4620	0.4820	0.4682	0.5990	0.1852	0.1962
iSQUAD [9]	✓			0.4291	0.5429	0.4750	0.6798	0.6591	0.6457	0.1600	0.2500	0.1857
Cloud19 [15]		✓		0.5082	0.5429	0.5231	0.5703	0.5682	0.5690	0.3602	0.4167	0.3830
LogCluster [10]		✓		0.4867	0.3714	0.3852	0.4522	0.4862	0.4671	0.4128	0.5000	0.4260
MEPFL [16]			✓	0.3286	0.4571	0.3823	0.2321	0.4818	0.3133	0.2946	0.4035	0.3562
CloudRCA [20]	✓	✓		0.2463	0.1370	0.1143	0.0913	0.2174	0.1180	0.3708	0.2630	0.2652
DiagFusion [17]	✓	✓	✓	0.7326	0.6744	0.7015	0.8176	0.7891	0.7895	0.3870	0.2813	0.3165
MicroCBR [22]	✓	✓	✓	0.6286	0.8000	0.6500	0.4630	0.4310	0.4464	0.4626	0.5714	0.4835
Medicine	✓	✓	✓	**0.9714**	**0.9428**	**0.9508**	**0.9152**	**0.9136**	**0.9136**	**0.8358**	**0.8333**	**0.8260**

5.1.2 Implementation. *Medicine* is implemented in PyTorch and all experiments are conducted on a Linux Server 20.04.1 LTS with two 48C48T Intel(R) Xeon(R) CPU E5-2650 v4@ 2.20GHz, one NVIDIA(R) Tesla(R)M4, and 125 GB RAM.

5.1.3 Performance Metrics. Failure classification is a multi-classification task. Here, we use weighted precision, weighted recall, and weighted F1-score to evaluate the performance of *Medicine* and other models. For convenience, we will refer to them simply as precision, recall, and F1-score later. The formulas for these metrics are as follows: $Precision = TP/(TP + FP)$, $Recall = TP/(TP + FN)$, $F1\text{-}score = 2 \cdot Precision \cdot Recall/(Precision + Recall)$, where TP is the number of correctly identified abnormal samples, FN is the number of anomalies that were not detected, and FP is the number of normal samples that were incorrectly identified as anomalies.

5.1.4 Baselines. We use DéjàVu [14], iSQUAD [9], Cloud19 [15], LogCluster [10], MEPFL [16], CloudRCA [20], DiagFusion [17] and MicroCBR [22] as our baseline approaches. These approaches use one, two, or three modalities for failure classification tasks. The modalities used by each approach are shown in the Table 3. For all approaches, we tune parameters and report the best results.

5.2 RQ1: Overall Performance of *Medicine*

Table 3 illustrates the performance comparison of various failure detection approaches across three datasets (D1, D2 and D3), focusing on three key metrics: Precision, Recall, and F1-score. Analysis of Table 3 reveals that *Medicine* attained F1 scores of 0.9508, 0.9136, and 0.8260 on D1, D2, and D3, respectively, markedly surpassing single-modal approaches like Déjàvu [14]. The inferior performance of single-modal approaches stems from their exclusive reliance on a singular data type, thus failing to harness the informational richness conveyed by other modalities. For example, while log data may chronicle system events, it might not adequately represent resource utilization, and metric data might lack granular operational insights. This inherent limitation in single-modal approaches can potentially lead to erroneous assessments.

In the realm of multimodal methodologies, CloudRCA [20] emerged as the least effective, with F1-score trailing *Medicine* by 0.8365, 0.7965, and 0.5608 on D1, D2, and D3, respectively. This discrepancy is attributed to CloudRCA's [20] heavy dependence on expert knowledge for constructing and maintaining knowledge repositories, thereby constraining its adaptability and robustness. Contrasted with the benchmark multimodal approach, DiagFusion [17], *Medicine* showcased substantial improvements, enhancing F1-score by 35.54% and 15.72% on D1 and D2, respectively. DiagFusion's [17] conversion of diverse modalities into a shared representation space disregards inter-modality distinctions, consequently disrupting the coherence of intra-modality information. This shortcoming manifests particularly in scenarios of incomplete multi-modal data or subpar data collection quality.

The disparity in performance between MicroCBR [22] and *Medicine* is marginally narrower on D1 than on D2 and D3, primarily because failure information in D1 predominantly resides within the singular modality of metrics, whereas D2 and D3 encompass both metrics and logs, reflecting a richer array of failure data. Despite MicroCBR's [22] endeavor to optimize all modalities with a unified learning objective and identical training strategies, it fails to address the inherent imbalance in modality optimization, resulting in F1-score of only 0.4464 and 0.4835 on D2 and D3, respectively.

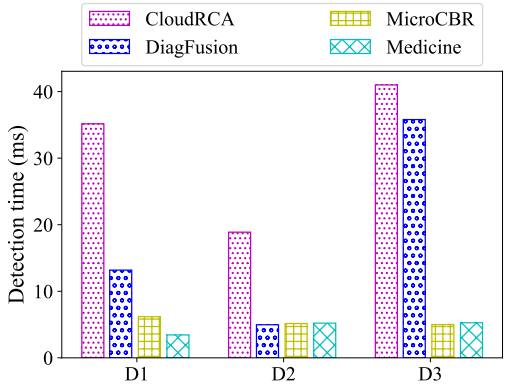

Figure 9: The average detection time for diagnosing a failure.

We simulate the online detection environment to analyze the complexity of *Medicine* and other multimodal baselines by calculating the average detection time required for each failure case. As shown in Figure 9, *Medicine* demonstrates the shortest detection

Table 4: Performance comparison of different components.

Dataset	Approach	Precision	Recall	F1-score
D1	Only Metric	0.8926	0.8857	0.8847
	Only Log	0.3316	0.4571	0.3820
	Only Trace	0.3595	0.4571	0.4020
	w/o MAO	0.9350	0.9143	0.9086
	Medicine	**0.9714**	**0.9428**	**0.9508**
D2	Only Metric	0.7705	0.7909	0.7753
	Only Log	0.8836	0.8500	0.8445
	Only Trace	0.5232	0.5227	0.5139
	w/o MAO	0.8959	0.8955	0.8953
	Medicine	**0.9152**	**0.9136**	**0.9136**
D3	Only Metric	0.6875	0.5000	0.4538
	Only Log	0.4792	0.4583	0.4431
	Only Trace	0.2550	0.2083	0.2179
	w/o MAO	0.7121	0.7083	0.6956
	Medicine	**0.8358**	**0.8333**	**0.8260**

time on D1, taking only 3.44ms, whereas CloudRCA [20] is the slowest at 35.16ms. On D2 and D3, *Medicine*'s average detection time is comparable to that of MicroCBR [22], all around 5ms. Given the frequent failure diagnosis operations performed by operators, *Medicine* meets this requirement efficiently. Furthermore, *Medicine* achieves satisfactory results by effectively leveraging three modalities, demonstrating superiority in both effectiveness and performance.

5.3 RQ2: Contributions of Components

To illustrate the significance and contributions of each component in *Medicine*, we conducted four sets of comparative experiments to assess the performance of its five components. These components are: **1) Only Metric**: To underscore the importance of the specially designed metric encoder, we isolated it from *Medicine* and appended a fully connected layer to output failure categories, utilizing only metric data in this scenario. **2) Only Log**: To emphasize the significance of the log encoder, we solely employed the log encoder for failure classification. **3) Only Trace**: To highlight the importance of the trace encoder, we exclusively utilized trace data in the trace encoder for failure diagnosis. **4) w/o MAO**: In this configuration, we omitted the multimodal adaptive optimization (MAO) module from *Medicine* while retaining all other components unchanged.

As depicted in Table 4, the metric encoder achieved F1-score of 0.8847, 0.7753, and 0.4538 on D1, D2, and D3, respectively, surpassing or closely approaching the best multimodal baseline in Table 3. This indicates that our designed metric encoder consistently characterizes failures, and the proposed feature sequences adeptly differentiate various failures. The log encoder attained an F1-score of 0.8445 on D2, significantly surpassing other log modality baselines. Similarly, the performance of the trace encoder outpaced the corresponding trace modality baselines. However, due to the limited number of failure cases reflected in the trace data, the trace encoder's performance did not realize its full potential.

In comparison to the unimodal encoders, the fusion of carefully encoded multimodal data yielded F1-score of 0.9086, 0.8953, and 0.6956 on D1, D2, and D3, respectively, emerging as the top-performing model alongside *Medicine*. With the incorporation of the multimodal adaptive optimization module, *Medicine*'s F1-score improved by 4.64%, 2.04%, and 18.75% on D1, D2, and D3, respectively. After multimodal adaptive optimization, *Medicine* retains useful information from all modalities. MAO dynamically adjusted and optimized the weights and interactions between different data modalities, thereby enhancing the overall performance of the model. In most scenarios, it is unknown which modality will be the most effective before model deployment. Therefore, we recommend training and optimizing the model using data from all three modalities before deployment.

5.4 RQ3: Parameter Sensitivity

We primarily discuss the impact of two hyperparameters in the multimodal adaptive optimization module (Section 4.5) on the performance of *Medicine*. Figure 10 illustrates how the average optimal F1-score of *Medicine* varies with different values of hyperparameters α and β. Specifically, we increased α in the gradient suppression component from 0.1 to 0.9. The experimental results indicate that if α is too small, the suppression effect on high-yield modalities is similar to that on low-yield modalities, failing to reduce their gradient propagation effectively. Conversely, if α is too large, high-yield modalities will not receive sufficient training. Setting α around 0.5 yields relatively better performance, hence we set $\alpha = 0.5$.

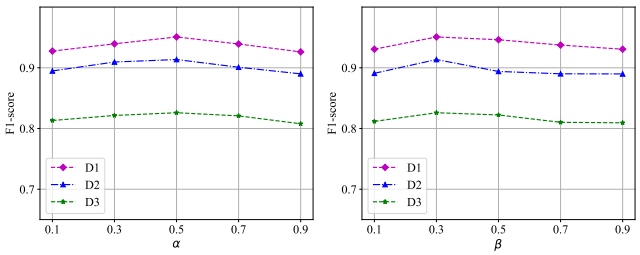

Figure 10: F1-score of *Medicine* under different parameters.

We also increased β in the feature enhancement component from 0.1 to 0.9. The best performance is achieved when $\beta = 0.3$. If β is too large, the performance of *Medicine* declines as low-yield modalities are overly emphasized, causing high-yield modalities to be neglected. During training, the actual gradient update direction (from the multimodal output) and the guiding direction for each modality (from the unimodal output) diverge, leading to increased interference [51]. Overprotecting low-yield modalities disrupts the ability of other modalities to fully utilize their features.

6 DISCUSSION

6.1 Case Study

We applied *Medicine* on a global, top-tier commercial bank, D1, where operators collected metrics, logs, and traces from the monitoring system to initiate a multimodal failure diagnosis task. Figure

11 illustrates a network packet loss failure case, where a significant drop in network and CPU-related metrics of the k8s container was observed, alongside a sharp increase in abnormal service calls. *Medicine* effectively diagnosed the anomaly, pinpointing "k8s container network packet loss" as the root cause. Following *Medicine* 's diagnosis, operators investigated the k8s container's network configuration and identified a DNS misconfiguration, which, once corrected, restored normal operations in the microservices system. Notably, no anomalies were detected in the related logs; however, *Medicine* accurately identified the issue by leveraging the valid information from metrics and traces. By integrating various data modalities, *Medicine* is capable of identifying failures that might elude single-modality approaches, thereby enabling more precise diagnoses.

6.2 Lessons Learned

While *Medicine* represents a significant advancement in failure diagnosis for microservice systems, it still has some limitations. Primarily, the framework's performance is critically dependent on the availability and quality of multimodal data. In real-world scenarios, inconsistencies in data collection may lead to diagnostic gaps, despite *Medicine*'s modular design aimed at preserving the independence of each modality. Additionally, although the adaptive optimization process adeptly balances learning across various modalities, it incurs computational overhead that could impede real-time performance. Furthermore, the reliance on pre-trained models and domain-specific knowledge for encoding different modalities may restrict the framework's applicability across diverse domains without extensive retraining and tuning. Therefore, developing generalized encoding techniques that require minimal retraining and tuning is necessary to enhance the framework's applicability across various domains.

The adaptive optimization module's hyperparameters require meticulous tuning. Once established, these parameters remain static during operation, limiting the framework's capability to dynamically adjust to evolving conditions or new data patterns. This static nature could constrain *Medicine*'s ability to continuously improve and adapt. In future work, we will integrate a feedback loop into *Medicine* that continuously monitors performance and automatically adjusts hyperparameters, enabling *Medicine* to adapt to evolving conditions and new data patterns without manual intervention.

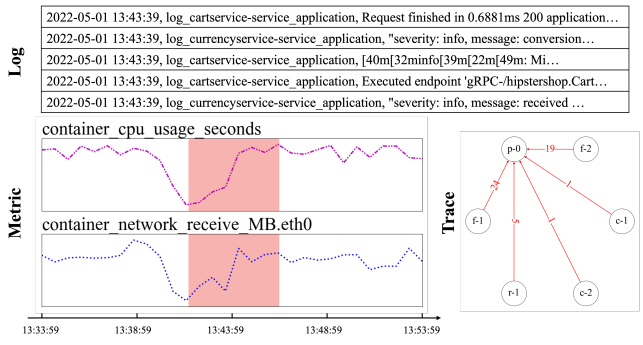

Figure 11: A failure case "k8s container network packet loss" on D1, with the red-highlighted areas indicating the anomalies detected by Medicine across different modalities.

6.3 Threats to Validity

Internal threats. Despite *Medicine* 's effectiveness in diagnosing failure types within the evaluated microservice systems, there exists a potential threat of overfitting to specific data patterns during the training process. Should the framework become overly attuned to the characteristics of the training data, it may fail to generalize effectively to new, unseen failure data exhibiting different patterns or anomalies. Furthermore, *Medicine* 's performance is highly contingent on the tuning of its hyperparameters. Extensive experiments were conducted to ascertain the optimal settings; however, these parameters may not be universally optimal across varied datasets or operational conditions, posing a risk of suboptimal performance in untested scenarios.

External threats. The datasets employed in the experiments might not encompass the full spectrum of failure types and system behaviors present in microservice environments. The specific attributes of these datasets could limit the generalizability of the research findings to other settings. *Medicine* was assessed on particular microservice systems, and the diversity in architectures, configurations, and deployment environments of other microservice systems might constrain the applicability of the results. Systems with markedly different structures or operational behaviors may necessitate adjustments to the *Medicine* framework. Although the framework demonstrated promising results in the test scenarios, its scalability in very large and highly dynamic microservice environments remains to be thoroughly evaluated. The computational overhead introduced by the adaptive optimization process has not been effectively assessed in such environments.

7 CONCLUSION

In this paper, we introduced *Medicine*, a modality-independent failure diagnosis framework for microservice systems based on multimodal adaptive optimization. *Medicine* addresses the limitations of existing single-modal diagnostic approaches by encoding different modalities separately, preserving their unique characteristics, and employing adaptive optimization to balance the learning progress across modalities. This approach significantly enhances overall diagnostic performance. Our extensive experiments demonstrate that *Medicine* outperforms other single-modal and multimodal diagnostic approaches on three public datasets, achieving an F1-score improvement of 15.72% to 70.84%. Notably, *Medicine* maintains high diagnostic accuracy even when some modality data is missing or of low quality, showcasing its robustness and practical applicability in real-world scenarios. In the future, we will further explore how to monitor the operational status of microservice systems in real time, enabling the timely detection and diagnosis of failures to provide users with more stable and reliable services.

REFERENCES

[1] James Lewis and Martin Fowler. Microservices. https://martinfowler.com/articles/microservices.html, 2014. [Online].

[2] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*, pages 338–347. IEEE, 2021.

[3] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. Monitorassistant: Simplifying cloud service monitoring via large language models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, pages 38–49. ACM, 2024.

[4] Lei Tao, Xianglin Lu, Shenglin Zhang, Jiaqi Luan, Yingke Li, Mingjie Li, Zeyan Li, Qingyang Yu, Hucheng Xie, Ruijie Xu, Chenyuan Hu, Canqun Yang, and Dan Pei. Diagnosing performance issues for large-scale microservice systems with heterogeneous graph. *IEEE Transactions on Services Computing*, pages 1–14, 2024.

[5] AWS. Microservices with Lambda. https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/microservices-with-lambda.html, 2024. [Online].

[6] AWS. Summary of the AWS Lambda Service Event in Northern Virginia (US-EAST-1) Region. https://aws.amazon.com/message/061323/, 2023. [Online].

[7] Yang Cai, Biao Han, Jie Li, Na Zhao, and Jinshu Su. Modelcoder: A fault model based automatic root cause localization framework for microservice systems. In *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*, pages 1–6. IEEE, 2021.

[8] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Zhekang Chen, Wenchi Zhang, Xiaohui Nie, Kaixin Sui, and Dan Pei. Practical root cause localization for microservice systems via trace analysis. In *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*, pages 1–10. IEEE, 2021.

[9] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, Feifei Li, Changcheng Chen, and Dan Pei. Diagnosing root causes of intermittent slow queries in large-scale cloud databases. *Proc. VLDB Endow.*, 13(8):1176–1189, 2020.

[10] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 102–111. ACM, 2016.

[11] Liqun Li, Xu Zhang, Shilin He, Yu Kang, Hongyu Zhang, Minghua Ma, Yingnong Dang, Zhangwei Xu, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. CONAN: diagnosing batch failures for cloud systems. In *45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 138–149. IEEE, 2023.

[12] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*, pages 674–688. ACM, 2024.

[13] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. Diagnosing performance issues in microservices with heterogeneous data source. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, September 30 - Oct. 3, 2021*, pages 493–500. IEEE, 2021.

[14] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, Yanhua Wang, Xu Du, Guoqiang Duan, and Dan Pei. 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, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, pages 996–1008. ACM, 2022.

[15] Yue Yuan, Wenchang Shi, Bin Liang, and Bo Qin. An approach to cloud execution failure diagnosis based on exception logs in openstack. In *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*, pages 124–131. IEEE, 2019.

[16] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 683–694. ACM, 2019.

[17] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, Dai Zhang, Zhenyu Zhu, and Dan Pei. Robust failure diagnosis of microservice system through multimodal data. *IEEE Trans. Serv. Comput.*, 16(6):3851–3864, 2023.

[18] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 553–565. ACM, 2023.

[19] Hanzhang Wang, Phuong Nguyen, Jun Li, Selçuk Köprü, Gene Zhang, Sanjeev Katariya, and Sami Ben-Romdhane. GRANO: interactive graph-based root cause analysis for cloud-native distributed data platform. *Proc. VLDB Endow.*, 12(12):1942–1945, 2019.

[20] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. 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*, pages 4373–4382. ACM, 2021.

[21] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 623–634. ACM, 2022.

[22] Fengrui Liu, Yang Wang, Zhenyu Li, Rui Ren, Hongtao Guan, Xian Yu, Xiaofan Chen, and Gaogang Xie. Microcbr: Case-based reasoning on spatio-temporal fault knowledge graph for microservices troubleshooting. In *Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings*, volume 13405, pages 224–239. Springer, 2022.

[23] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selçuk Köprü, and Tao Xie. 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*, pages 419–429. IEEE, 2021.

[24] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. Trinityrcl: Multi-granular and code-level root cause localization using multiple types of telemetry data in microservice systems. *IEEE Trans. Software Eng.*, 49(5):3071–3088, 2023.

[25] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 1750–1762. IEEE, 2023.

[26] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. MULAN: multi-modal causal structure learning and root cause analysis for microservice systems. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, pages 4107–4116. ACM, 2024.

[27] Pradeep Dogga, Karthik Narasimhan, Anirudh Sivaraman, Shiv Kumar Saini, George Varghese, and Ravi Netravali. Revelio: Ml-generated debugging queries for finding root causes in distributed systems. In Diana Marculescu, Yuejie Chi, and Carole-Jean Wu, editors, *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022.

[28] Medicine. https://anonymous.4open.science/r/Medicine-204D, 2024. [Online].

[29] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. Jump-starting multivariate time series anomaly detection for online service systems. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 413–426. USENIX Association, 2021.

[30] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, Dan Pei, Qingwei Lin, and Dongmei Zhang. Robust multimodal failure detection for microservice systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 5639–5649. ACM, 2023.

[31] Byung-Chul Tak, Shu Tao, Lin Yang, Chao Zhu, and Yaoping Ruan. LOGAN: problem diagnosis in the cloud using log-based reference models. In *2016 IEEE International Conference on Cloud Engineering, IC2E 2016, Berlin, Germany, April 4-8, 2016*, pages 62–67. IEEE Computer Society, 2016.

[32] Siyang Lu, BingBing Rao, Xiang Wei, Byung-Chul Tak, Long Wang, and Liqiang Wang. Log-based abnormal task detection and root cause analysis for spark. In *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 389–396. IEEE, 2017.

[33] Dong Young Yoon, Ning Niu, and Barzan Mozafari. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1599–1614. ACM, 2016.

[34] Cuong Pham, Long Wang, Byung-Chul Tak, Salman Baset, Chunqiang Tang, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Failure diagnosis for distributed systems using targeted fault injection. *IEEE Trans. Parallel Distributed Syst.*, 28(2):503–516, 2017.

[35] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. Diagnosing performance issues in microservices with heterogeneous data source. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, September 30 - Oct. 3, 2021*, pages 493–500. IEEE, 2021.

[36] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE International Conference on Web Services, ICWS 2020, Beijing, China, October 19-23, 2020*, pages 142–150. IEEE, 2020.

[37] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R. Lyu. Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In *Proceedings of the 45th International Conference on Software Engineering*, page 1724–1736. IEEE Press, 2023.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[39] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, 2017.

[40] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 492–504, 2021.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[42] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7132–7141. Computer Vision Foundation / IEEE Computer Society, 2018.

[43] Christopher M. Bishop and Hugh Bishop. *Deep Learning - Foundations and Concepts*. Springer, 2024.

[44] Xiaokang Peng, Yake Wei, Andong Deng, Dong Wang, and Di Hu. Balanced multimodal learning via on-the-fly gradient modulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 8228–8237. IEEE, 2022.

[45] Junru Wu, Yi Liang, Feng Han, Hassan Akbari, Zhangyang Wang, and Cong Yu. Scaling multimodal pre-training via cross-modality gradient harmonization. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[46] Zeyan Li, Nengwen Zhao, Shenglin Zhang, Yongqian Sun, Pengfei Chen, Xidao Wen, Minghua Ma, and Dan Pei. Constructing large-scale real-world benchmark datasets for aiops. *arXiv preprint arXiv:2208.03938*, 2022.

[47] CloudWise. GAIA-DataSet. https://github.com/CloudWise-OpenSource/GAIA-DataSet, 2023. [Online].

[48] Microservo. https://anonymous.4open.science/r/microservo, 2024.

[49] Google Cloud Platform Team. Microservices demo. https://github.com/GoogleCloudPlatform/microservices-demo, 2024. Accessed: 2024-06-24.

[50] Principles of Chaos Engineering. Principles of chaos engineering. https://principlesofchaos.org/, 2024. Accessed: 2024-07-02.

[51] Ruize Xu, Ruoxuan Feng, Shi-Xiong Zhang, and Di Hu. Mmcosine: Multi-modal cosine loss towards balanced audio-visual fine-grained learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2023, Rhodes Island, Greece, June 4-10, 2023*, pages 1–5. IEEE, 2023.