ART: A Unified Unsupervised Framework for Incident Management in Microservice Systems

Yongqian Sun
Nankai University &
TKL-SEHCI
Tianjin, China

Binpeng Shi
Nankai University
Tianjin, China

Mingyu Mao
Nankai University
Tianjin, China

Minghua Ma
Microsoft
Redmond, WA, USA

Sibo Xia
Nankai University
Tianjin, China

Shenglin Zhang*
Nankai University &
HL-IT
Tianjin, China

Dan Pei
Tsinghua University &
BNRist
Beijing, China

ABSTRACT

Automated incident management is critical for large-scale microservice systems, including tasks such as anomaly detection (AD), failure triage (FT), and root cause localization (RCL). Currently, most techniques focus only on a single task, overlooking shared knowledge across closely related tasks. However, employing isolated models for managing multiple tasks may result in inefficiencies, delayed responses, a lack of systemic perspective, and complexity in updates and operations. Therefore we propose ART, an unsupervised framework that integrates a full-process solution covering Anomaly detection, failure Triage, and Root cause localization. It reaches the unification of multiple tasks by extracting the shared knowledge. Specifically, we first conduct an empirical study to analyze how the shared knowledge embedded in anomalous deviations manifests in AD, FT, and RCL. To better calculate deviations and extract shared knowledge, we sequentially model channel, temporal, and call dependencies using Transformer Encoder, GRU, and GraphSAGE, respectively. Then unified failure representations enhance the interpretability of abstract features with explicit semantic information, serving as the basis for unsupervised multitask solutions. Our evaluations on the datasets generated from two benchmark microservice systems demonstrate that ART outperforms existing methods in terms of AD (improving by 5.65% to 60.8%), FT (improving by 13.2% to 95.7%), and RCL (improving by 13.3% to 205%).

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools.**

*Shenglin Zhang is the corresponding author. Email: zhangsl@nankai.edu.cn
TKL-SEHCI, HL-IT, and BNRist are short for Tianjin Key Laboratory of Software Experience and Human Computer Interaction, Haihe Laboratory of Information Technology Application Innovation, Beijing National Research Center for Information Science and Technology, respectively.

KEYWORDS

Microservice System, Anomaly Detection, Failure Triage, Root Cause Localization

ACM Reference Format:
Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. ART: A Unified Unsupervised Framework for Incident Management in Microservice Systems. 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, 12 pages. https://doi.org/10.1145/3691620.3695495

1 INTRODUCTION

Microservice systems have become an essential part of our daily lives. However, due to the dynamic and complex nature, incidents (*e.g.*, unplanned failures, outages) are inevitable, which bring huge economic and reputational damage. In 2023 alone, the microservice systems of several large internet companies such as Microsoft [4], Google [2], and Alibaba Cloud [1] experienced large-scale incidents. Furthermore, a 24-hour incident of mission-critical microservices from AWS could result in a direct revenue loss of $3.4 billion [3]. Therefore, it is vital to ensure the quality of microservice systems and manage incidents efficiently and effectively.

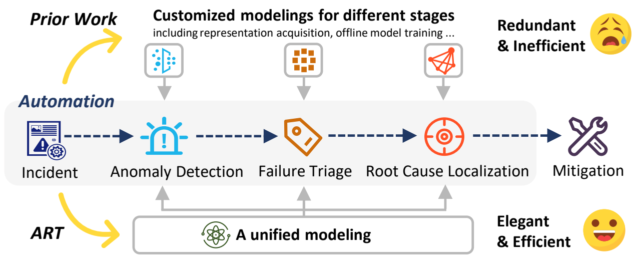

Figure 1: An incident life-cycle: Prior Work vs. ART

As shown in Figure 1, a typical incident life-cycle includes four stages [9, 13]: (1) Detection: The first step is to generate an alert when anomalous system behavior is detected. (2) Triage: Next, the incident is assigned to the appropriate engineering team after an initial assessment of the incident type. (3) Diagnosis: Assigned on-call engineers (OCEs) investigate various aspects of the incident and engage in multiple rounds of communication to localize the root cause. (4) Mitigation: OCEs take appropriate measures to mitigate the incident and restore system health. Each stage is time-consuming and labor-intensive. To reduce the burden on OCEs, significant efforts have been devoted to automated solutions for each stage except mitigation. Mitigation requires specific actions by OCEs. Thus, a large portion of automated incident management techniques primarily consists of anomaly detection (AD) [7, 22, 28, 36, 38, 41, 47, 55, 62, 63], failure triage (FT) [32, 35, 37, 48, 59, 61], and root cause localization (RCL) [6, 12, 16, 19, 34, 52, 57, 58, 64].

At the start, OCEs have to combine single-task techniques to address the challenges at different stages of the incident life-cycle. But this approach requires redundant effort for multiple tasks (*e.g.*, personalized feature extraction, representation acquisition, offline model training, data organization formats, configurations), leading to extended Time-to-Mitigate (TTM) [53]. Furthermore, oversimplified techniques to avoid tedious but effective modeling may introduce noise into subsequent tasks [27]. Therefore, some researchers turn to multi-task techniques [27, 29, 31, 60, 66]. However, the latest ones also could not offer end-to-end modeling covering AD, FT, and RCL, while requiring manual involvement. Some techniques [27, 31, 60, 66] fall under supervised techniques, which heavily rely on high-quality labels. Other techniques [29] may require rules derived from experienced operators to construct a causal graph. In practice, manual involvement makes these multi-task techniques seriously overstretched.

OCEs call for an elegant and efficient unified modeling approach, and self-supervised learning (SSL) provides a promising solution [11]. By pretraining on massive amounts of unlabeled data, it extracts general representations which are then transferred to multiple downstream tasks. However, directly applying SSL to incident management encounters the following challenges:

(1) ***Complexity*** of the status of microservice systems in representation learning. Due to the diverse dependencies, modeling multimodal heterogeneous data, including metrics, logs, and traces, generated from microservice systems is challenging. As shown in Figure 2, a data channel refers to a single time series obtained from the multimodal data serialization (see Section 4.2). Different channels impact each other (channel dependency, CHA). The channels fluctuate over time (temporal dependency, TEM). Additionally, different instances interact with one another (call dependency, CAL). These dependencies are prevalent in microservice systems. Such complex and diverse data patterns call for a model capable enough to extract features comprehensively. However, none of the existing techniques [27, 31, 60–62] can adequately capture all three dependencies.

(2) ***Interpretability*** of representations obtained through SSL. Several studies in other domains aim for interpretability through the semantic consistency of abstract features [8, 15]. However, in incident management, most deep learning techniques [27, 28, 31, 60] pay no attention to the connection between features and the semantics of the microservice system status. This unexplainable black-box nature makes the model's inference process and outputs difficult to understand, potentially causing confusion for OCEs.

(3) ***Scarcity*** of labels for downstream tasks. Although the pretraining phase of representation learning does not require manual labels, SSL typically employs supervised fine-tuning to bridge the gap to downstream tasks. However, manual labels are costly and

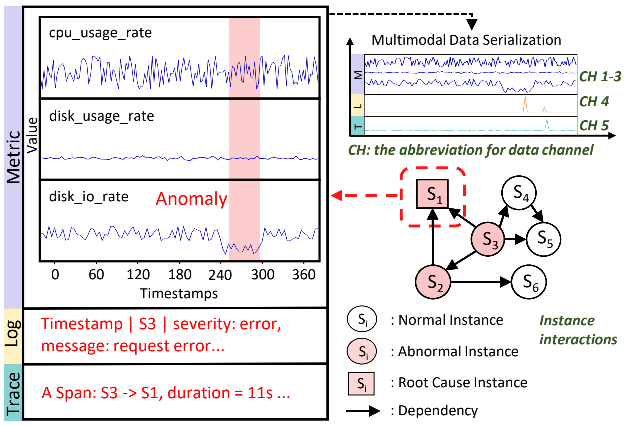
Metric
Value
cpu_usage_rate
disk_usage_rate
disk_io_rate
Anomaly
0 60 120 180 240 300 360
Timestamps
Log
Timestamp | S3 | severity: error, message: request error...
Trace
A Span: S3 -> S1, duration = 11s ...
Multimodal Data Serialization
M
L
T
CH 1-3
CH 4
CH 5
CH: the abbreviation for data channel
S_1
S_2
S_3
S_4
S_5
S_6
S_i : Normal Instance
S_i : Abnormal Instance
S_i : Root Cause Instance
: Dependency
Instance interactions

Figure 2: An incident case: the multimodal data of the root cause instance, *i.e.*, metrics, logs and traces, and examples of instance interactions

difficult to obtain in practice. Providing unsupervised solutions for AD, FT, and RCL based on failure representations while achieving superior performance is challenging.

To tackle these challenges, we propose ART, an unsupervised incident management framework offering a comprehensive solution for Anomaly detection, failure Triage, and Root cause localization based on unified failure representations. Specifically, ART consists of three modules that correspond to the above three challenges: (1) *Dependency-Aware Status Learning*. We sequentially model channel, temporal, and call dependencies utilizing self-attention mechanisms, recurrent neural networks, and graph neural networks, respectively. Integrating these dependencies enables the model to more accurately predict the microservice system's normal status at the next time interval, which forms the foundation for calculating deviations. (2) *Unified Failure Representation Acquisition*. This module calculates instance-level deviations based on predictions and then aggregates them to obtain system-level deviations. The two deviations introduce interpretability by linking each dimension of the representation vectors to the original data channel. (3) *Unsupervised Solutions for Diagnostic Tasks*. Finally, the failure representations are transferred to specific tasks in an unsupervised manner through EVT thresholds, customized cut-tree-based clustering, and correlation analysis, respectively. The contributions of our work are summarized as follows:

- To the best of our knowledge, we are the first to propose an end-to-end unsupervised incident management framework, ART, which integrates AD, FT, and RCL. It reaches the unification of multi-tasks by extracting the shared knowledge in an elegant and efficient manner.
- We empirically study the shared knowledge embedded in anomalous deviations across multi-tasks (Section 2.1). Inspired by it, we define a unified failure representation that explicitly incorporates semantic information into abstract features, enhancing interpretability and serving as the basis for multi-task solutions (Section 4.3).
- To better calculate deviations and extract shared knowledge, ART sequentially models channel, temporal, and call dependencies

through Transformer Encoder, GRU, and GraphSAGE, respectively (Section 4.2). Moreover, we identify the significance of the dependency modeling orders both theoretically (Section 2.2) and experimentally (Section 5.3).
- Extensive experiments are conducted on two popular microservice systems, consisting of 46 and 18 instances, respectively (Section 5). The results demonstrate ART's effectiveness and efficiency. Our source code and experimental data are publicly available [1].

2 MOTIVATION

This section elaborates on our motivation by exploring the following two topics: (1) Is there any shared knowledge among AD, FT, and RCL? (2) How to extract the shared knowledge effectively? The first topic aims to identify the commonalities among multiple tasks, forming the foundation for unified modeling. The second topic guides on effectively extracting these commonalities.

2.1 Is There Any Shared Knowledge among AD, FT, and RCL?

To achieve unified modeling, we should identify the commonalities across all stages of incident management. Anomalous deviation, indicating the degree to which monitored objects (*e.g.*, systems, instances) deviate from normal expectations, has reached wide success in incident management [42, 54, 62]. However, these techniques treat closely related tasks as independent, failing to elevate deviations to shared knowledge of the entire process.

Therefore, we conduct an empirical study to fully explore the shared knowledge about the microservice status embedded in deviations. We utilize the first 30% of the failure cases in dataset D1, which are never included in the test set. It is adequate and representative to support a valid analysis. More details about D1 are provided in Section 5.1.1. Considering the continuity of the time series, we use the previous time step's value as the normal expectation for the current time step, which means regarding the first-order difference as an estimate of deviations. Then, as described in Section 4.3, we obtain the unified failure representations, specifically instance-level and system-level deviations (ILD and SLD). These k-dimensional vectors reflect the fluctuations of each corresponding channel at both the instance and system levels. Next, we analyze whether and how the two forms of deviations manifest in multitasks.

2.1.1 *Deviations Manifested in AD.*

We begin by computing the statistics of the L_1-norms of SLDs during failure and normal hours, as presented in Table 1. $\|SLD\|_1 = \sum_{i=1}^{k} |d_i|$, where d_i denotes the i-th value of the k-dimensional vector. Deviations are typically larger during the failure hours, which is intuitive. Specifically, the $\|SLD\|_1$ in failure hours is 22% higher on average and 17% higher at the median compared to normal hours. The percentile column shows the proportion of values exceeding the $\|SLD\|_1$ of normal hours. During failures, these percentages reach 85% and 73% respectively. The L_1-norms of SLDs summarize the system fluctuations by aggregating deviations across all data channels. This notable distinction highlights deviations as an indicator of whether the system is normal or not.

[1] https://github.com/bbyldebb/ART

Table 1: L_1-norms of SLDs during failure and normal hours

System Status	Metric	Deviations: $\|SLD\|_1$	Percentile
Failure Hours	Mean	100.620	P85
	Median	90.165	P73
Normal Hours	Mean	82.716	P64
	Median	77.147	P49

2.1.2 *Deviations Manifested in FT.*

Next, we focus on the deviation in each data channel of the system. On one hand, certain channels serve as key indicators of failure types with good specificity, *e.g.*, deviations in the duration channel can clearly distinguish between network and non-network failures. On the other hand, some channels are inherently volatile, regardless of failures. Still taking duration as an example, it typically experiences significant fluctuations due to non-failure factors like network load and transmission distance, causing large deviations. To mitigate this interference and identify core channels indicative of different failure types, we compute the standardized SLD. The mean and variance for standardization are derived from SLDs in normal hours, which act as an initialization set to characterize the regular distribution of channels. We rank the channels for each failure type by standardized deviations and highlight the top five ones. The results, shown in Table 2, reveal that different failure types correspond to different channels. For Container Network failure, the most volatile channels include duration in traces, connection_error in logs, and system.net.udp.in_errors in metrics, corroborating our earlier point.

Notably, the standardization approach that removes the effect of inherent channel fluctuations has some drawbacks. Firstly, it requires a high-quality initialization set, which ideally includes comprehensive channel fluctuation characteristics. Moreover, channels with significant fluctuations imply greater sensitivity and tend to have a stronger discriminative power for system conditions. Overlooking this may lead to difficulties in the rapid division of failure cases or even distinguishing subtle differences between failure types. Thus, when conducting the channel-level analysis that maps channel deviations to specific types, ultimately assigning incidents to proper engineering teams, *i.e.*, FT, we must balance the specificity of the channels with their discriminative power for concrete system conditions to achieve better classification.

2.1.3 *Deviations Manifested in RCL.*

Finally, we analyze the correlations between system-level deviations (SLDs), and instance-level deviations of root causes and non-root causes (ILD_i of instance i) through cosine similarity. This analysis aims to determine whether root causes have a higher correlation with system deviations. The results are shown in Table 3. Non-root cause instances display similarities all under 0.5. In contrast, root cause ones reach approximately 71% and 76%, exceeding those of 83% and 90% of instances in the entire system. This suggests that root cause instances often contribute more to system fluctuations, exhibiting deviation patterns more similar to SLD. This valuable insight serves as the basis for the solution to RCL.

In summary, deviations are integral to the analysis process of all three tasks, which contain shared knowledge. While both AD and

Table 2: Top5 channels with the largest deviations for different failure types

Failure Type	Top5 Data Channels with the Largest Deviations				
Container Hardware	container_fs_inodes	container_fs_usage_MB	container_fs_writes	container_memory_cache	container_threads
Container Network	duration	severity_error	connection_error	service_log_other	system.net.udp.in_errors
Node CPU	system.disk.total	system.fs.inodes.free	system.fs.inodes.in_use	system.fs.inodes.total	system.load.15
Node Disk	container_last_seen	system.disk.free	system.disk.pct_usage	system.disk.total	system.disk.used
Node Memory	system.mem.pct_usage	system.mem.real.pct_useage	system.mem.real.used	system.mem.usable	system.mem.used

Table 3: Silimarity between SLDs and ILDs of root cause and non-root cause instances

Instances	Metric	Cosine Similarity	Percentile
Root Cause	Mean	0.714	P83
	Median	0.767	P90
Non-root Cause	Mean	0.487	P46
	Median	0.499	P48

FT deal with system-level changes, AD offers a broader understanding of system deviation, indicating the presence of failure, whereas FT involves a finer analysis by mapping channel fluctuations to failure types. Conversely, RCL focuses on instances rather than the entire system, identifying root causes from the many affected ones that deviate from expectations.

Finding 1: Deviations, in the specific forms of instance-level and system-level deviations, contain shared knowledge about the microservice status, which assists in anomaly detection, failure triage, and root cause localization.

2.2 How to Extract the Shared Knowledge Effectively?

Then, the challenge lies in fully exploiting the shared knowledge in deviations. Essentially, the validity of deviations depends on the accuracy of predictions. Thus, we need to adequately model the status of complex microservice systems to obtain accurate predictions. We analyze the three types of dependencies illustrated by the incident case, as shown in Figure 2: (1) *Channel dependency* (CHA) refers to the correlation between multimodal data channels within a single modality and across different modalities. As the disk IO rate decreases, the corresponding disk IO time rises, leading to a longer response time for external requests. This is reflected in both metrics and the duration of traces. (2) *Temporal dependency* (TEM) visualizes how channels change over time. When trained with normal hours data, the model captures the fluctuations of each channel. For example, disk_usage_rate moves smoothly while cpu_usage_rate fluctuates significantly. Log printing also follows a fixed pattern, with the associated channels remaining relatively stable. (3) *Call dependency* (CAL) represents the interaction of instances due to invocation and deployment. As a result of the cascading effect, instances near the root cause exhibit anomalies, while those further away or unrelated show no issues.

Additionally, in deep learning, extracted features become increasingly abstract as the layers deepen and get closer to the output. This phenomenon is observed in various fields such as CV and NLP [33, 56, 65]. As a result, step-by-step feature extraction from shallow to abstraction may guide the model to capture different levels of representations, thereby improving generalization [20, 21]. Back to the dependencies in microservice systems, CHA and TEM are refined to the data channel level: CHA describes the association of raw channels, and TEM handles complex fluctuation patterns over time. And CAL models interactions at the instance level. Therefore, ART extracts the three dependencies sequentially, from fine-grained and shallow to coarse-grained and abstract, expecting better access to the knowledge embedded in deviations.

Existing work covers the three dependencies mentioned above. In concrete terms, AnoFusion [62] first focuses on CHA between multimodal data. CloudRCA [61] and DiagFusion [60] consider TEM and CAL, either directly or indirectly. Dejavu [31] and Eadro [27] model the three dependencies comprehensively. However, Dejavu uses convolution to capture CHA, which is limited by the receptive field size. Eadro, on the other hand, only considers CHA within a single modality, ignoring cross-modality connections. Additionally, none of them consider the impact of feature extraction order. In conclusion, the introduction of each dependency helps model the status of microservice systems from a theoretical perspective. Moreover, in the ablation experiment of Section 5.3, we validate the effectiveness of the three dependencies and demonstrate the importance of feature extraction order by comparing six different combinations of these dependencies.

Finding 2: Combined consideration of channel, temporal, and call dependencies is beneficial for shared knowledge acquisition. And the order of feature extraction matters.

3 PRELIMINARIES

3.1 Self-supervised Learning

Self-supervised learning (SSL) is an unsupervised learning paradigm leveraging inherent data information for model training. Through pretext tasks such as reconstruction [45], generative adversarial networks [14], contrastive learning [17], and prediction [44], pre-trained models are acquired. Then the inner representations are transferred to downstream tasks or other domains. On one hand, SSL reduces reliance on labels during pre-training, requiring only a small amount for fine-tuning downstream tasks, sometimes even outperforming supervised models. On the other hand, SSL unifies multiple tasks at the representation level. Unlike joint learning, which requires designing a joint loss function and more supervised data for multitasks, SSL learns robust inner representations with strong generalization. Therefore, we adopt SSL as the solution for unified modeling.

3.2 Neural Networks

This section introduces the neural networks used to capture the channel, temporal, and call dependencies.

Transformer Encoder. The self-attention mechanism in Transformer captures long-range information by calculating the correlation between positions in the input sequence [50]. Specifically, we use vectors representing different channels as input tokens, with attention weights describing channel correlations. We employ only the Transformer Encoder instead of a complete encoding and decoding architecture, driven by our primary emphasis on extracting features through the self-attention mechanism while minimizing parameter size and computational costs. Plenty of failure analysis studies [23, 49] achieve good results in representation learning with Transformer Encoder alone. In summary, ART chooses the Transformer Encoder to capture channel dependency.

Gated Recurrent Unit. RNN [25] models temporal dependency with deterministic hidden variables but struggles with long-term dependency in time series. LSTM [24] and GRU [30] are proposed to address this issue. Generally, GRU achieves comparable results to LSTM with fewer parameters and a simpler structure. So we choose GRU to capture the temporal dependency of the data channels.

GraphSAGE. GCN [26] fuses graphical features using convolution. However, it requires all nodes during training and becomes unsuitable for changing graphs, which falls short of our expectations for the optimization of dynamic topology. In addition, GCN's full graph gradient updating is inefficient. GraphSAGE [18] addresses the issues by learning a function that generates embeddings through sampling and aggregating features from a node's local neighborhood, instead of training individual embeddings for each node. Additionally, GAT [51] assigns weights to nodes to differentiate their importance, leading to higher training and inference costs in large graphs. We opt for the Transformer Encoder to assign attention coefficients at a finer-grained channel level. Therefore, GraphSAGE is expected to capture call dependency between instances for scalability and efficiency.

3.3 Problem Statement

Typically, operators continuously collect multimodal data (metrics, logs, and traces) to ensure the reliability of microservice systems. As shown in Figure 2, metrics are multivariate time series that monitor hardware, system, and various services. Logs record system runtime behavior as semi-structured text. A trace is made up of spans, each corresponding to an invocation [43]. In addition, traces come with duration, status code, and other annotations. We transform them into time series as detailed in Section 4.2 and collectively call them data channels, *i.e.*, $X = \{(X_i^{\mathcal{M}}, X_i^{\mathcal{L}}, X_i^{\mathcal{T}})\}_{i=1}^{N}$, where at the i-th of the N instances, $X_i^{\mathcal{M}}$, $X_i^{\mathcal{L}}$, $X_i^{\mathcal{T}}$ denotes metrics, logs and traces after serialization respectively.

Our work attempts to build an end-to-end framework: Given X, the AD module detects failures, denoted by the binary indicator y (0 for normal, 1 for abnormal). If y equals 1 multiple times within a time window, it means a failure is detected and then the FT module is triggered to make an initial assessment of the failure type s from a predefined failure library. Lastly, RCL estimates the probability of each instance being the culprit, denoted by $P = [p_1, \ldots, p_N] \in [0,1]^N$. The framework is built on a parameterized model $\mathcal{F} : X \rightarrow (y, s, P)$.

4 METHODOLOGY

4.1 Overview

As shown in Figure 3, ART consists of three modules. The offline phase mainly involves *Module 1, Dependency-Aware Status Learning*. We utilize serialization and sliding windows to preprocess normal-time multimodal data into window sequences. Then CHA, TEM, CAL are captured in turn to train a model that predicts system states in normal hours. *Module 2, Unified Failure Representation Acquisition*, is designed to get two-level failure representations rich in semantic information, which exposes the shared knowledge obtained from SSL to anomalous deviations. For online diagnosis, we preprocess the new-coming multimodal data into window sequences and feed them into the trained model to get the predictions for the next time steps. We continuously compute the deviation matrices between the predictions and observations and then obtain the two-level failure representations, which form the basis of AD, FT, and RCL. Specifically, *Module 3, Unsupervised Solutions for Diagnostic Tasks*, monitors whether a failure occurs, and if so, it determines the failure type and then localizes the root cause instance.

4.2 Dependency-Aware Status Learning

This module trains a model through SSL with CHA,TEM, and CAL captured, to predict the system's normal states.

Multimodal Data Serialization. Referring to [27, 62], we transform multimodal data (metrics, logs, and traces) into time series. Briefly, metrics are time series requiring only regular preprocessing steps such as normalization. For logs, we count the frequency of log events after parsing to form the time series. For traces, we extract key variables and transform them into multivariate time series by computing statistics such as the average latency per minute, the total number of requests per minute, *etc.* Then we concatenate to obtain a fused multivariate time series $M = [M_{\text{metric}} \| M_{\text{log}} \| M_{\text{trace}}]$. Next, we employ resampling and nearest-neighbor interpolation to standardize all metric intervals to one minute. By performing z-score standardization [5] using a sliding historical window on M, we can obtain a normalized input sequence $X = \{X^{(1)}, \ldots, X^{(T)}\}$ in a time window. T is the length of the time window. The ***snapshot*** of the microservice system at time t can be expressed as $X^{(t)} \in \mathbb{R}^{N \times K}$, where N is the number of instances and K is the number of data channels. That is, $x_{i,j}^{(t)}$ denotes the normalized value of channel j on instance i at time t. The above steps achieve the preliminary fusion across metrics, logs, and traces, which has been proven effective by a large number of studies [22, 27, 28, 62]. Further dependencies are learned through SSL.

Multi-dependency Feature Extraction. As detailed in Section 3.2, we apply Transformer Encoder, GRU, and GraphSAGE to model the CHA, TEM, and CAL of the microservice system in turn. We are more concerned with the necessity of combining the three dependencies and the optimal feature extraction order, which is

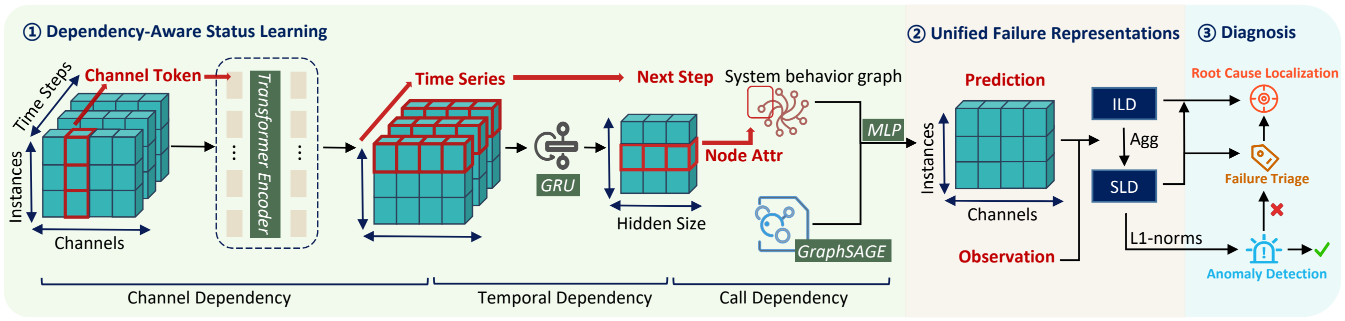

Figure 3: The framework of ART.

theoretically analyzed in Section 2.2 and experimentally illustrated in Section 5.3.

(1) Transformer Encoder for CHA. The system snapshot $X^{(t)}$ can be analyzed from both instance and channel dimensions, *i.e.*, the rows and columns of the matrix. For CHA, we view $X^{(t)}$ as a set of states of data channels. The state of each channel at time t is an N-dimensional vector, and each dimension represents the value of that channel for the corresponding instance. ART treats each channel's N-dimensional description of the microservice system as a token that composes the input sequence of the Transformer Encoder. Therefore, we need to transpose $X^{(t)}$ to $X'^{(t)} \in \mathbb{R}^{K\times N}$. We focus on the multi-head self-attention mechanism, which assigns attention weights to different channels and captures dependencies between them. Formally, for the l-th head of the attention layer, the scaled dot-product attention is defined as:

$$head_l = Attention(X'^{(t)}W_l^Q, X'^{(t)}W_l^K, X'^{(t)}W_l^V) \tag{1}$$

where $Attention(Q,K,V) = Softmax(\frac{QK^T}{\sqrt{d_v}})V$; W_l^Q, W_l^K, and W_l^V are linear projection weights with dimensions $\mathbb{R}^{N\times d_v}$ for the l-th head, and d_v is the dimension for one head of the attention layer.

The Transformer Encoder usually consists of multiple transformer layers. Each transformer layer includes a multi-head self-attention and a position-wise feed-forward network in which a residual connection is employed around each of the two sub-layers, followed by layer normalization [50]. Finally, we transpose the output back to the dimensions $\mathbb{R}^{N\times K}$.

(2) GRU for TEM. Let $H \in \mathbb{R}^{(N\times K)\times T}$ denotes the sequence of updated snapshots. ART uses GRU to capture the TEM, which can be formulated as:

$$\begin{aligned}
r^{(t)} &= \sigma(W_{ir}H^{(t)} + W_{hr}h^{(t-1)} + b_r)\\
z^{(t)} &= \sigma(W_{iz}H^{(t)} + W_{hz}h^{(t-1)} + b_z)\\
n^{(t)} &= tanh(W_{in}H^{(t)} + W_{hn}(h^{(t-1)} \odot r^{(t)}) + b_n)\\
h^{(t)} &= (1 - z^{(t)}) \odot n^{(t)} + z^{(t)} \odot h^{(t-1)}
\end{aligned} \tag{2}$$

where $h^{(t)}$ and $h^{(t-1)}$ are the hidden states, $r^{(t)}$, $z^{(t)}$, and $n^{(t)}$ are the reset, update, and new gates, respectively, σ is the sigmoid function, $\odot$ is the Hadamard product. W and b are trainable parameters. ART takes the final hidden state as the latent features of the snapshot at the next time step, i.e. $f^{(T+1)} = h^{(T)} \in \mathbb{R}^{N\times d_h}$. d_h is the hidden size of GRU layers. Since we are still considering TEM on the channel dimensions, we pack the dimension N in H into the batch in the implementation. The dependency on instance dimensions, *i.e.*, CAL, is modeled by GraphSAGE.

(3) GraphSAGE for CAL. To depict the attributes of instances and the interactions among them, we introduce the concept of ***system behavior graph*** (SBG). An SBG is a directed graph, $G = (V, E, F)$. V is the set of all instances in a microservice system. An edge $(v_i, v_j) \in E$ represents an actual call or deployment from instance v_j to v_i, which means v_j depends on v_i. $F \in \mathbb{R}^{N\times d_h}$ is the latent features of each instance obtained from the previous steps. We construct SBGs every minute, consistent with the granularity of the monitoring data. Then GraphSAGE is applied to the snapshot's instance dimension modeling on SBGs, which consists of several layers and the operation of the k-th layer is formulated as:

$$\begin{aligned}
F_{\mathcal{N}(i)}^{(k)} &= Agg(F_j^{(k-1)}, \forall j \in \mathcal{N}(i))\\
F_i^{(k)} &= Norm(\sigma(W^{(k)} \cdot Concat(F_i^{(k-1)}, F_{\mathcal{N}(i)}^{(k)})))
\end{aligned} \tag{3}$$

where $F_i^{(k)}$ is the k-th layer's features of instance i, $\mathcal{N}(i)$ is the set of neighbors of instance i, Agg is the operation of aggregating the features of the neighbors (*e.g.*, average), $W^{(k)}$ is the learnable weight matrix of the k-th layer, and σ is the LeakyReLU activation function [40]. Information about the neighbors of each instance is obtained through sampling and aggregation, capturing the dependencies within and between instances layer by layer.

Offline Self-Supervised Training. Add an MLP to align the dimensions of the dependency-aware latent space with that of the snapshot. That is, the final output $\hat{X}^{(T+1)} \in \mathbb{R}^{N\times K}$ serves as the snapshot of the next time step obtained from the prediction of the input snapshot sequence. ART adopts mean squared error (MSE) between the prediction $\hat{X}^{(T+1)}$ and the observation $X^{(T+1)}$ as the loss function. We feed sequences of snapshots in normal hours to train a model that predicts the normal expectations of the microservice system.

4.3 Unified Failure Representation Acquisition

ART leverages anomalous deviations as the core to obtain two-level failure representations, enhancing the interpretability with semantic information. For a time window T during the online diagnostic phase, we use MSE between the prediction $\hat{X}$ and the observation

X to compute the *deviation matrix*, *i.e.*, $D \in \mathbb{R}^{N\times K}$.

$$
\begin{aligned}
D_{ij}^{(t)} &= (\hat{X}_{ij}^{(t)} - X_{ij}^{(t)})^2 \\
D_{ij} &= Agg(\{D_{ij}^{(1)}, \ldots, D_{ij}^{(T)}\})
\end{aligned}
\tag{4}
$$

where Agg is the aggregation operation (*e.g.*, average) to eliminate the effect of episodic fluctuations, D_{ij} denotes the aggregated deviation of instance i on channel j in the window T.

Each row of D represents the deviations of each instance over all channels in the window and thus can be viewed as a set of ***instance-level deviations***, *i.e.*, $ILD = \{ILD_1, \ldots, ILD_N\}$. $ILD_i \in \mathbb{R}^K$ is the K-dimensional deviation vector of instance i. Then aggregate the ILD to obtain the ***system-level deviations*** (SLD). To reduce the noise, we exclude normal and less abnormal instances from contributing to the SLD in the aggregation process. Specifically, we compute the $L1$-norm of ILD_i for all instances, indicating the degree of anomaly of the instance. Then we select the q instances with the highest degree of the anomaly to form the set Q and get the weighted sum SLD, *i.e.*, $SLD = \sum_{i\in Q}(\|ILD_i\|_1 \times ILD_i) \in \mathbb{R}^K$. In addition to specifying the number q directly, we also provide a manner of aggregation by probability in the open-source code. That is, the $\|ILD_i\|_1$ is treated as a probability after softmax normalization, and the top z instances are selected such that their sum exceeds a threshold (*e.g.*, 0.9). z is of variable size. ART defaults to the former one because of its simplicity and effectiveness. Each dimension of SLD represents the deviation of the microservice system from the expectation on the corresponding channel. Additionally, the $L1$-norm of SLD reflects the overall level of anomaly in the system.

There are two levels of unified failure representations: ILD and SLD. These k-dimensional vectors, including their $L1$-norms, are associated with semantic information of the microservice system, laying the foundation for downstream unsupervised solutions.

4.4 Unsupervised Solutions for Diagnostic Tasks

Inspired by the empirical study in Section 2.1, ART uses ILD and SLD to accomplish the diagnostic task in an unsupervised manner.

Anomaly Detection. As detailed in Section 2.1.1, the system typically displays more pronounced deviations in failure hours. Therefore, ART utilizes a threshold to detect anomalies, determined automatically and accurately using Extreme Value Theory (EVT) [46]. EVT is a statistical theory that identifies occurrences of extreme values without making assumptions about data distribution. EVT can be utilized to estimate the likelihood of observing extreme values for AD, which has been proven effective by previous anomaly detection methods [38, 39, 62]. Specifically, we collect the $L1$-norm of SLD in normal hours as the initialization set and compute the EVT threshold. For a new coming snapshot sequence, an anomaly is detected if the $\|SLD\|_1$ exceeds the EVT threshold. However, anomalies may be occasional fluctuations due to non-failure factors such as increased task load. Hence, we set a delay time window in which if a second anomaly is detected, we consider the occurrence of a system failure. Subsequently, the FT module is triggered to make an initial assessment of the failure type.

Failure Triage. As detailed in Section 2.1.2, different failure types are associated with deviations in specific data channels. Additionally, channels with frequent fluctuations often demonstrate higher sensitivity to system conditions, showing stronger discriminative abilities. ART designs a cut-tree-based clustering approach to partition failure representations based on system-level channel deviations, specifically each dimension of SLD. As shown in Figure 4, We collect several failure cases and calculate their SLDs to form an initialization set, then construct the cut tree in two steps.

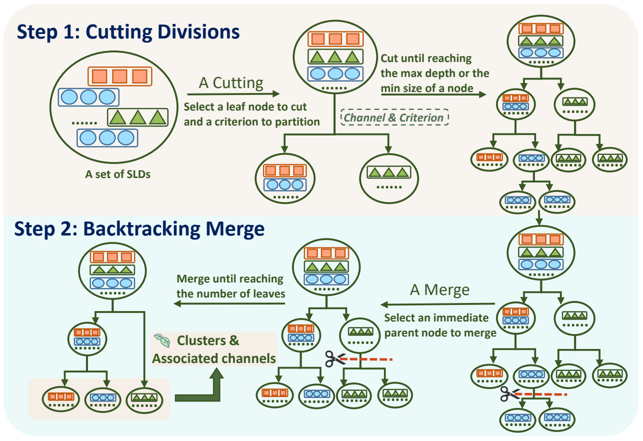
Step 1: Cutting Divisions
A set of SLDs
A Cutting
Select a leaf node to cut and a criterion to partition
Channel & Criterion
Cut until reaching the max depth or the min size of a node
Step 2: Backtracking Merge
Merge until reaching the number of leaves
A Merge
Select an immediate parent node to merge
Clusters & Associated channels

Figure 4: The steps of the cut-tree-based clustering

Step 1: Cutting Divisions. Referring to the decision tree but without labels, we should divide all the SLDs based on certain rules. Each node of the cut tree represents a set of SLDs associated with failure cases and the root contains all the SLDs. For each cutting operation, we need to first select a leaf node. Considering the discriminative power of different channels regarding system conditions, we first compute the variance of the SLDs in each dimension for every leaf node, and take the maximum variance as the gain of cutting the node. To swiftly and accurately divide the nodes, we iteratively select the leaf node with the maximum gain and perform the cutting operation on the corresponding dimension without repetition. Then select a criterion to partition the SLDs within the node into two child nodes. Specifically, to maximize the distinction between the two child nodes, we traverse all the values on the cut dimension within the chosen node, one of which is selected as the division criterion if the cosine distance between the SLDs of two children is maximized after division by it. Repeat the cutting operation until reaching the maximum tree depth or the minimum number of SLDs within a node.

Step 2: Backtracking Merge. To optimize the cut tree and reduce the number of leaf nodes, we calculate the average cosine distance between the SLDs within each immediate parent node of the leaves, which assesses the compactness of the node. Then we choose the least compact immediate parent node to merge iteratively until the number of leaves is no larger than a specified threshold. The resulting leaf nodes form the clusters.

For a new coming SLD, we transfer it from the root to the leaf node (*i.e.*, the cluster) based on the cut dimensions and criteria. The failure type is determined by that of the central node of the cluster. Moreover, during the top-down traversal process, all data channels corresponding to the cut dimensions are attached, providing interpretable channel-level details for FT. The initialization set involved in the tree-building process does not necessitate labels, which are only required to map the clusters to failure types during inference.

Root Cause Localization. As detailed in Section 2.1.3, due to the significant contribution of root cause instances to system fluctuations, the fluctuation patterns of root cause instances often exhibit greater similarity to that of the system. Based on the semantic information of the unified failure representations, ILD_i and SLD describe the fluctuation patterns of instance i and the entire system through deviations on each data channel, respectively. Therefore, ART calculates the cosine similarity between ILD_i of all instances and SLD as suspicious scores and ranks them accordingly, *i.e.*, $P = [p_1, \ldots, p_N] \in [0, 1]^N$. A higher similarity indicates a higher probability of being the root cause.

In summary, when the AD module detects an incident, the FT module initially assesses the failure type and assigns it to the appropriate engineering team for resolution. Channel-level details from the FT module and instance-level culprits from the RCL module help the engineering team restore system health more accurately and quickly.

5 EVALUATION

In this section, we address the following research questions:
RQ1: How well does ART perform in AD, FT, and RCL?
RQ2: Does each component contribute to ART?
RQ3: How do the major hyperparameters of ART influence its performance?

5.1 Experimental Setup

5.1.1 Datasets. To evaluate the performance of ART, we conduct extensive experiments on two microservice system datasets, D1 and D2. We use the earliest timestamp in the last 40% of failure cases as the split point. The period before this point forms the training set for the baselines, while the remaining forms the test set. The cases in the training set are used to initialize the FT module's cut tree in ART, ensuring that the failure numbers are sufficient to build a relatively stable cut tree. Note that the initialization process does not involve manual labels. Table 4 lists more details.

- **D1**[2] is collected from a simulated e-commerce microservice system, which is deployed in a real cloud environment with traffic consistent with real business flow. The system comprises 46 instances, including 40 microservice instances and 6 virtual machines. Failure records were collected by replaying the failures over several days in May 2022. The failure scenarios are derived from actual failures (Container Hardware, Container Network, Node CPU, Node Disk, and Node Memory-related failures). The collected records were labeled with their respective root cause instances and failure types.
- **D2** is collected from the management system of a top-tier commercial bank, which comprises 18 instances, including microservices, servers, databases, and dockers. Two experienced operators examined the failure records from January 2021 to June 2021 and labeled the root cause instances and failure types (Memory, CPU, Netowork, Disk, JVM-Memory, and JVM-CPU-related failures). Each operator conducted the labeling process separately and cross-checked the labels to ensure consensus. D2 has been used in the International AIOps Challenge 2021[3]. Due to the non-disclosure agreement, we cannot make it publicly available.

[2] https://github.com/bbyldebb/ART

[3] https://aiops-challenge.com

Table 4: Detailed information of datasets

Dataset	# Instances	# Failure	# Normal	# Failure Types	# Records	
D1	46	210	3,714	5	trace	44,858,388
					log	66,648,685
					metric	20,917,746
D2	18	133	12,297	6	trace	214,337,882
					log	21,356,870
					metric	12,871,809

5.1.2 Baseline Approaches. First, we select all existing advanced multi-task methods (*i.e.*, Eadro [27], Dejavu [31], DiagFusion [60]) except for ShapleyIQ [29]. Because ShapleyIQ's causal assumption and data requirements regarding queries per second, conditional events, *etc.* , do not align well with D1 and D2. Secondly, we select three representative unsupervised or semi-supervised multimodal methods (*i.e.*, Hades [28] for AD, MicroCBR [35] for FT, and PDiagnose [19] for RCL). Especially for RCL, we favor the methods with instance-level localization granularity to align with ART. More details are given in Section 7. We configure the parameters as specified in the papers. For dataset-specific settings (*e.g.*, window length), we adjust them based on the ranges provided or according to our data. Additionally, due to the absence of an AD module in some methods and to maintain independent performance evaluation for each task, we assume known timestamps of failures during FT and RCL evaluations.

5.1.3 Evaluation Metrics. Both AD and FT are classification tasks. The former is a binary classification of whether a failure occurs, while the latter is a multi-classification problem of which type the current failure belongs to. During evaluations, we adopt True Positive(TP), False Positive (FP), and False Negative (FN), and then calculate $precision = TP/(TP+FP)$, $recall = TP/(TP+FN)$, $F1\text{-}score = 2 \cdot precision \cdot recall/(precision + recall)$. Additionally, we use the $Weighted\ Average\ F1\text{-}score$[10] for FT considering the imbalanced failure types. For RCL, we introduce $TopK = \frac{1}{N}\sum_{i=1}^{N}(g_i \in P_{i,[1:K]})$ to calculate the probability of the root cause within the top-k predicted candidates $P_{i,[1:K]}$, where g_i is the groundtruth root cause for the i-th failure case, N is the number of failures for evaluation. $AVG@5$ is calculated by $AVG@5 = \frac{1}{5}\sum_{K=1}^{5} TopK$.

5.1.4 Implementation. We implement ART and baselines with Python 3.9.13, PyTorch 1.12.1, scikit-learn 1.1.2, and DGL 0.9.0. We run the experiments on a server with 12 × Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz and 128G RAM (without GPUs). We repeat each experiment five times and average the results to minimize the effect of randomness.

5.2 RQ1: Overall Performance

As shown in Tabel 5, ART outperforms baseline methods in terms of AD (improving by 5.65% to 60.8%), FT (improving by 13.2% to 95.7%), and RCL (improving by 13.3% to 205%). Both MicroCBR and

Table 5: Performance comparison for AD, FT, and RCL. "-" means the method does not cover the problem.

#	Method	D1									D2								
		AD			FT			RCL			AD			FT			RCL		
		Precision	Recall	F1	Precision	Recall	F1	Top1	Top3	AVG@5	Precision	Recall	F1	Precision	Recall	F1	Top1	Top3	AVG@5
multiple	ART	**0.899**	**0.990**	**0.942**	**0.836**	**0.809**	**0.812**	**0.667**	**0.810**	**0.776**	**0.877**	**0.960**	**0.917**	**0.851**	**0.796**	**0.802**	**0.722**	**0.889**	**0.870**
	Eadro [27]	0.425	0.946	0.586	-	-	-	0.137	0.315	0.302	0.767	0.935	0.842	-	-	-	0.157	0.315	0.310
	Dejevu [31]	-	-	-	0.369	0.621	0.415	0.411	0.679	0.625	-	-	-	0.718	0.340	0.417	0.402	0.667	0.619
	DiagFusion [60]	-	-	-	0.675	0.500	0.568	0.310	0.452	0.467	-	-	-	0.797	0.527	0.593	0.582	0.709	0.695
single	Hades [28]	0.866	0.863	0.865	-	-	-	-	-	-	0.867	0.868	0.868	-	-	-	-	-	-
	MicroCBR [35]	-	-	-	0.667	0.796	0.717	-	-	-	-	-	-	0.629	0.678	0.636	-	-	-
	PDiagnose [19]	-	-	-	-	-	-	0.615	0.692	0.685	-	-	-	-	-	-	0.037	0.296	0.285

PDiagnose methods overlook the relationships between multimodal data (CHA), and in PDiagnose, the analysis results of each modality are influenced by the preceding one, causing potential cascading effects. Hades fuses information between metrics and logs but only detects anomalies for a single instance, ignoring interactions between multiple instances (CAL). Compared to the above single-task methods, ART achieves the best results by leveraging shared knowledge across closely related tasks, which prevents overfitting and facilitates joint improvement of multiple tasks. Multi-task methods (Eadro, Dejavu, DiagFusion) treat AD, FT, and RCL as classification problems using supervised classifiers. This black-box approach is straightforward but ignores the semantic consistency of abstract features. Conversely, ART maps representations to physical meanings of microservice systems and designs unsupervised solutions around the semantic information, achieving better scores and improved interpretability. The results demonstrate the effectiveness of ART's dependency-aware status learning, and the well-designed unsupervised solutions elicit shared knowledge better than a fully connected layer. Additionally, SSL's ability to learn inherent generalized data representations also contributes to ART outperforming supervised methods to some extent.

As for the efficiency comparisons in Table 6, ART completes more tasks in less time than other baseline methods due to its unified modeling approach, which ensures high performance with a lightweight structure. Efficient data preprocessing (multimodal data serialization) also plays a significant role. Moreover, baseline methods encounter practical challenges that are difficult to quantify experimentally, such as manual label acquisition [27, 28, 31, 60], customized data organization [19, 60], multiple model selection [19, 28, 35], and manual rule integration [35]. These challenges significantly impact incident management efficiency, adding burdens on OCEs. In conclusion, the results illustrate the applicability of ART to real-world applications, exhibiting its capability to perform real-time AD, FT, and RCL in a unified and efficient manner.

5.3 RQ2: Ablation Study

To evaluate the effects of the key technique contributions of ART: (a) three types of dependency modeling (*i.e.*, CHA, TEM, CAL); (b) dependency extraction order, we create eight variants of ART. **A1:** Remove the Transformer Encoder for CHA. **A2:** Remove the GRU for TEM. **A3:** Remove the GraphSAGE for CAL. Apart from ART's choice of CHA-TEM-CAL, five orders for dependency extraction remain: **B1:** CHA-CAL-TEM. **B2:** CAL-CHA-TEM. **B3:** CAL-TEM-CHA. **B4:** TEM-CHA-CAL. **B5:** TEM-CAL-CHA.

Table 6: The comparison of training time (Offline) and diagnosis time (Online) per case. The unit is second. "-" means no need for training.

Method	Target			D1		D2	
	AD	FT	RCL	Offline	Online	Offline	Online
ART	✓	✓	✓	460.262	0.872	1085.767	1.363
Eadro	✓		✓	510.570	0.627	795.416	0.899
Dejavu		✓	✓	1182.468	0.427	1937.330	1.028
DiagFusion		✓	✓	621.309	4.145	310.357	3.297
Hades	✓			1214.528	0.104	2073.0413	0.415
MicroCBR		✓		-	0.278	-	0.306
PDiagnose			✓	-	4.342	-	9.919

Table 7: The evaluation results of ablation study

Method	D1			D2		
	AD: F1	FT: F1	RCL: AVG@5	AD: F1	FT: F1	RCL: AVG@5
ART	**0.942**	**0.812**	**0.776**	**0.917**	**0.802**	**0.870**
A1	0.900	0.558	0.727	0.891	0.727	0.851
A2	0.914	0.671	0.672	0.783	0.754	0.853
A3	0.922	0.700	0.725	0.858	0.638	0.857
B1	0.936	0.794	0.748	0.906	0.717	0.855
B2	0.926	0.728	0.770	0.881	0.621	0.866
B3	0.893	0.680	0.770	0.892	0.728	0.863
B4	0.931	0.769	0.755	0.845	0.786	0.862
B5	0.893	0.758	0.714	0.888	0.570	0.844

Table 7 lists that ART outperforms all the variants on D1 and D2, demonstrating each component's significance. When any of the dependencies (*i.e.*, CHA, TEM, and CAL) is removed (**A1**-**A3**), the performance of ART drops. This is particularly evident in FT, which requires fine-grained analysis at the channel level. The results illustrate that the combined three dependencies enable comprehensive modeling of the microservice system status and facilitate the extraction of a more effective failure representation. When adjusting the dependency extraction order (**B1**-**B5**), we find that the variants generally outperform omitting any single dependency. However, a decrease remains evident compared to the CHA-TEM-CAL sequence of ART. It indicates that the order of dependency extraction matters. When modeling dependencies, we strive to follow the order that progresses from fine-grained and shallow to coarse-grained and abstract. The experiments in this section corroborate the theoretical analysis in Section 2.2.

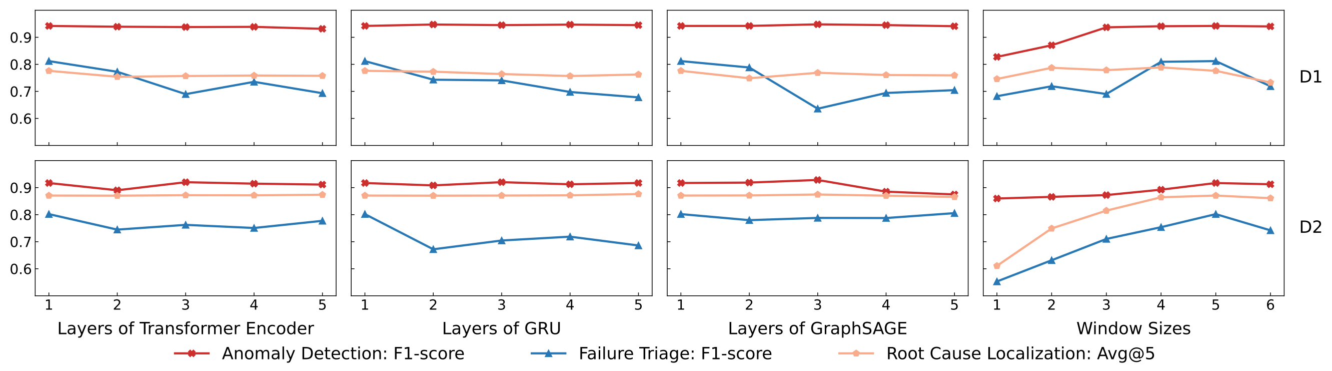

Figure 5: The effectiveness of ART under different hyperparameters

5.4 RQ3: Hyperparameters Sensitivity

We discuss the effect of four hyperparameters on ART' performance. As shown in Figure 5, the performance of AD and RCL demonstrates relative stability when varying the number of neural network layers in dependency-aware status learning (*i.e.*, **Layers of Transformer Encoder, GRU, and GraphSAGE**). FT experiences a slight degradation as the number of layers increases, likely due to overfitting from the more complex models on limited samples. We recommend setting the number of layers to 1 for each component if it works well. In addition, ART exhibits an overall trend of performance improvement followed by a decline with varying **Window Sizes** in acquiring failure representations. Clearly, a window that is too small fails to encompass complete failure information, while an excessively large window contains too much irrelevant data. In our study, setting it to 5 proves to be a suitable choice.

6 DISCUSSION

6.1 Limitations and Possible Solutions

We identify two major limitations of ART and try to suggest possible solutions: (1) Addressing Unknown Failures. AD and RCL are not subject to this limitation because they do not rely on historical failures. For FT, ART needs to collect multiple failure cases to build a cut tree and cover as many failure types as possible, necessitating high-quality historical failure sets. In the early stage of model deployment, ART may fail to achieve good FT results due to fewer historical failures; as the number of failure cases accumulates, it is advisable to periodically rebuild the cut tree to adapt to new failure types. (2) Accommodating frequent creation or destruction at the instance level of a microservice system. In the dependency learning module, ART inputs instance-dimensional channel tokens into the Transformer Encoder to model CHA. To some extent, ART trades flexibility to refine localization to the instance level. To mitigate this, we can aggregate instance data channels by service (*i.e.*, the channel token's dimension becomes the number of services) and then train a version of service-level localization at deployment. Retraining ART is only necessary when system services change, which is rare in a stable business.

6.2 Threats to Validity

Two main threats challenge the validity of ART: (1) The limited sizes of the two studied datasets. D1 and D2 may be less complex and dynamic than large-scale industrial microservice systems. However, they still hold value for evaluation. They originate from representative systems with diverse architectures and businesses. Experimental results support the validity and generalizability of ART. We believe that ART is promising for application in large industrial microservice systems with more complex failure scenarios. (2) The prerequisites for multimodal data collection. ART is designed for three modalities of data (*i.e.*, metrics, logs, and traces), but some real-world systems may lack the ability to collect multimodal data. While it is optimal to provide all data types, the low-coupled nature of multimodal serialization permits the absence of certain data sources. Additionally, if some data sources suddenly disappear due to a failure, the corresponding data channel will be significantly dropped, exhibiting large anomalous deviations. This is consistent with the premise assumptions of ART and theoretically does not impact ART's performance.

7 RELATED WORK

Single-task Techniques. A great deal of effort has been devoted to AD [7, 22, 28, 36, 38, 41, 47, 55, 62, 63], FT [32, 35, 37, 48, 59, 61], and RCL [6, 12, 16, 19, 34, 52, 57, 58, 64]. These single-task techniques cover only certain stages and cannot span the entire lifecycle. In practice, operators must select and combine different techniques based on their understanding of specific scenarios to meet the diverse requirements of each stage. Unfortunately, different techniques may require customized data preprocessing, personalized feature extraction, offline model training, specific configurations, *etc.* , accompanied by lots of redundant yet unavoidable work. Additionally, the simple combination of single-task techniques treats each stage of the failure lifecycle as independent, wasting the rich correlation in closely related tasks. Furthermore, the trade-off between efficiency and accuracy hinders the straightforward integration of state-of-the-art anomaly detectors, classifiers, and root cause localizers [27]. Specifically, using advanced techniques to complete AD, FT, and RCL sequentially often fails to meet the demands of real-time analysis. Therefore, some solutions may apply oversimplified methods, such as N-sigma for AD [57], expert rules for FT [34], and voting mechanism for RCL [19]. However, if the model cannot deliver satisfactory analysis results in the previous stage, it will introduce noise to the next stage, ultimately affecting the effectiveness of the entire process.

Multi-task Techniques. To overcome the limitations of the simple combination approaches, researchers turn to multi-task techniques [27, 29, 31, 60, 66]. For instance, ShapleyIQ [29] employs multimodal data to build a causal graph for RCL via counterfactual evaluation and Shapley values. However, it relies on physical law-based models that require expert knowledge and may not be universally applicable. Its AD module uses seasonal-trend decomposition and statistical tests, which may suffer from oversimplification. Dejavu [31] and MEPEL [66] train classifiers based on historical incidents, intuitively and concisely outputting failure types and locations. DiagFusion [60] and Eadro [27] pay attention to shared knowledge in similar tasks. They adopt joint learning to prevent models from overfitting on individual tasks while reducing training overheads. Existing multi-task techniques aim to design models that cover as many incident life-cycle stages as possible to improve overall performance and efficiency while reducing operational burdens. However, they still fail to cover the full process of AD, FT, and RCL. Moreover, most of these techniques are supervised [27, 31, 60, 66] or rely on artificial rules [29]. In practice, high-quality labels and expert laws are scarce resources, consolidating a large amount of time and effort from experienced operators. OCEs call for a unified unsupervised model to address the challenges at various stages of incident management. This is precisely the work of ART.

8 CONCLUSION

Ensuring reliability in microservice systems is critical. Our study presents ART, an innovative unsupervised framework for automated incident management that encompasses AD, FT, and RCL using multimodal data. We investigate shared knowledge in anomalous deviations across multiple tasks and employ Transformer Encoder, GRU, and GraphSAGE to model channel, temporal, and call dependencies. This enhances deviation calculations and knowledge extraction. Unified failure representations are then enriched with explicit semantic information, improving interpretability and supporting end-to-end unsupervised solutions. We believe that the approach to extracting shared knowledge across closely related tasks will benefit more areas beyond microservice systems.

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] 2023. Alibaba Cloud Health Dashboard. https://status.aliyun.com/#/historyEvent.
[2] 2023. Google Cloud Services Hit by Outage in Paris. https://thenewstack.io/google-cloud-services-hit-by-outage-in-paris/.
[3] 2023. Parametrixinsurance Cloud Outage and the Fortune 500 Analysis. https://www.parametrixinsurance.com/cloud-outage-and-the-fortune-500-analysis.
[4] 2023. Where are we now – Microsoft 363? Cloud suite suffers another outage. https://www.theregister.com/2023/04/24/microsoft_365_search_outage/.
[5] Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. 2006. Data mining: A preprocessing engine. *Journal of Computer Science* 2, 9 (2006), 735–739.
[6] Anunay Amar and Peter C Rigby. 2019. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 140–151.
[7] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. 2020. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3395–3404.
[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[9] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 674–688.
[10] Nancy Chinchor and Beth M Sundheim. 1993. MUC-5 evaluation metrics. In *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*.
[11] Linus Ericsson, Henry Gouk, Chen Change Loy, and Timothy M Hospedales. 2022. Self-supervised representation learning: Introduction, advances, and challenges. *IEEE Signal Processing Magazine* 39, 3 (2022), 42–62.
[12] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ML-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 135–151.
[13] Vaibhav Ganatra, Anjaly Parayil, Supriyo Ghosh, Yu Kang, Minghua Ma, Chetan Bansal, Suman Nath, and Jonathan Mace. 2023. Detection Is Better Than Cure: A Cloud Incidents Perspective. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1891–1902.
[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.
[15] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems* 33 (2020), 21271–21284.
[16] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. 2023. TrinityRCL: Multi-Granular and Code-Level Root Cause Localization Using Multiple Types of Telemetry Data in Microservice Systems. *IEEE Transactions on Software Engineering* (2023).
[17] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, Vol. 2. IEEE, 1735–1742.
[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
[19] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021. 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)*. IEEE, 493–500.
[20] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. 2017. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109* (2017).
[21] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.
[22] Jun Huang, Yang Yang, Hang Yu, Jianguo Li, and Xiao Zheng. 2023. Twin Graph-Based Anomaly Detection via Attentive Multi-Modal Learning for Microservice System. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 66–78.
[23] Shaohan Huang, Yi Liu, Carol Fung, He Wang, Hailong Yang, and Zhongzhi Luan. 2023. Improving log-based anomaly detection by pre-training hierarchical transformers. *IEEE Trans. Comput.* (2023).
[24] Chen Jia and Yue Zhang. 2020. Multi-cell compositional LSTM for NER domain adaptation. In *Proceedings of the 58th annual meeting of the association for computational linguistics*. 5906–5917.
[25] Soopil Kim, Sion An, Philip Chikontwe, and Sang Hyun Park. 2021. Bidirectional rnn-based few shot learning for 3d medical image segmentation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 1808–1816.
[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[27] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, Los Alamitos,CA, 1750–1762.

[28] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R Lyu. 2023. Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1724–1736.

[29] Ye Li, Jian Tan, Bin Wu, Xiao He, and Feifei Li. 2023. ShapleyIQ: Influence Quantification by Shapley Values for Performance Debugging of Microservices. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 287–323.

[30] Zhe Li, Peisong Wang, Hanqing Lu, and Jian Cheng. 2019. Reading selectively via Binary Input Gated Recurrent Unit.. In *IJCAI*. 5074–5080.

[31] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. 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*. 996–1008.

[32] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[33] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).

[34] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. 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)*. IEEE, 338–347.

[35] Fengrui Liu, Yang Wang, Zhenyu Li, Rui Ren, Hongtao Guan, Xian Yu, Xiaofan Chen, and Gaogang Xie. 2022. MicroCBR: Case-Based Reasoning on Spatio-temporal Fault Knowledge Graph for Microservices Troubleshooting. In *International Conference on Case-Based Reasoning*. Springer, 224–239.

[36] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, et al. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 48–58.

[37] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.

[38] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. 2021. {Jump-Starting} multivariate time series anomaly detection for online service systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 413–426.

[39] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. 2018. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 13–24.

[40] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. Atlanta, GA, 3.

[41] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.

[42] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. 2013. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems* 24, 6 (2013), 1245–1255.

[43] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly detection from system tracing data using multimodal deep learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 179–186.

[44] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2536–2544.

[45] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[46] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. 2017. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1067–1075.

[47] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2828–2837.

[48] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. 2023. LogKG: Log Failure Diagnosis through Knowledge Graph. *IEEE Transactions on Services Computing* (2023).

[49] Shimin Tao, Yilun Liu, Weibin Meng, Zuomin Ren, Hao Yang, Xun Chen, Liang Zhang, Yuming Xie, Chang Su, Xiaosong Oiao, et al. 2023. Biglog: Unsupervised large-scale pre-training for a unified log representation. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. IEEE, 1–11.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[51] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* 1050, 20 (2017), 10–48550.

[52] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE international conference on web services (ICWS)*. IEEE, 142–150.

[53] Weijing Wang, Junjie Chen, Lin Yang, Hongyu Zhang, Pu Zhao, Bo Qiao, Yu Kang, Qingwei Lin, Saravanakumar Rajmohan, Feng Gao, et al. 2021. How long will it take to mitigate this incident for online service systems?. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 36–46.

[54] Jingjing Yang, Yuchun Guo, Yishuai Chen, and Yongxiang Zhao. 2023. TraceNet: Operation Aware Root Cause Localization of Microservice System Anomalies. In *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 758–763.

[55] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.

[56] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems* 27 (2014).

[57] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments. In *Proceedings of the Web Conference 2021*. 3087–3098.

[58] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. 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*. 553–565.

[59] Yue Yuan, Wenchang Shi, Bin Liang, and Bo Qin. 2019. An approach to cloud execution failure diagnosis based on exception logs in openstack. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 124–131.

[60] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, et al. 2023. Robust failure diagnosis of microservice system through multimodal data. *IEEE Transactions on Services Computing* (2023).

[61] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. 2021. CloudRCA: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4373–4382.

[62] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, et al. 2023. Robust multimodal failure detection for microservice systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5639–5649.

[63] Nengwen Zhao, Junjie Chen, Zhaoyang Yu, Honglin Wang, Jiesong Li, Bin Qiu, Hongyu Xu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. 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*. 527–539.

[64] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. 2024. MULAN: Multi-modal Causal Structure Learning and Root Cause Analysis for Microservice Systems. In *Proceedings of the ACM on Web Conference 2024*. 4107–4116.

[65] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2921–2929.

[66] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. 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*. 683–694.