ADmM: Anomaly Detection for Microservice Systems with Incomplete Metrics

ZEKUN ZHANG, School of computer science, Wuhan University, Wuhan, China
JIAN WANG, School of Computer Science, Wuhan University, Wuhan, China and Zhongguancun Laboratory, Beijing, China
BING LI, School of Computer Science, Wuhan University, Wuhan, China and Zhongguancun Laboratory, Beijing, China
LIUXIAOXIAO ZHANG, R&D Center, Taikang Insurance Group, Wuhan, China
YU LIU, School of Computer Science, Wuhan University, Wuhan, China
PATRICK HUNG, Ontario Tech University, Oshawa, Canada

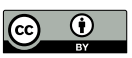
The rapid development of the internet has led to an exponential increase in the scale of computing, storage, networking, and service resources. Traditional monolithic architectures are increasingly insufficient for managing these complexities. In contrast, microservice architectures have emerged as the mainstream solution with their inherent flexibility in deployment and scalability. To ensure system reliability, modern microservice architectures rely heavily on observability data, including logs, metrics, and traces. However, challenges such as network instability, service instance restarts, and system overloads frequently lead to intermittent loss of metric data. These missing data points impede comprehensive assessments of system health, significantly threatening system stability and reliability. To address the above challenge, we propose an anomaly detection model, ADmM, which integrates logs, metrics, and traces. ADmM first extracts template-level and semantic-level features from multimodal inputs. Then, a multi-scale autoencoder module is applied to impute missing metrics. For anomaly detection, the model represents microservice dependencies as a directed acyclic graph and leverages a graph neural network to learn generative patterns from normal system behavior. By measuring the deviation between observed values and reconstructed values, ADmM assigns anomaly scores to identify anomalies. Experiments conducted on three open-source benchmarks demonstrate that ADmM outperforms state-of-the-art methods across multiple anomaly detection metrics. Notably, it achieves F1-Score improvements of 5.77%, 5.48%, and 2.16% in scenarios with 40% incomplete metrics.

CCS Concepts: • **Software and its engineering → Maintaining software;**

Additional Key Words and Phrases: Microservice, anomaly detection, incompleteness, metrics

ACM Reference Format:
Zekun Zhang, Jian Wang, Bing Li, Liuxiaoxiao Zhang, Yu Liu, and Patrick Hung. 2026. ADmM: Anomaly Detection for Microservice Systems with Incomplete Metrics. *ACM Trans. Web* 20, 2, Article 21 (April 2026), 35 pages. https://doi.org/10.1145/3801729

This work is supported by the National Natural Science Foundation of China (Grant Nos. 62032016 and 62572362) and the Shenzhen Science and Technology Program (Grant No. CJGJZD20230724091659002).
Authors' Contact Information: Zekun Zhang, School of computer science, Wuhan University, Wuhan, China; e-mail: zekunzhang@whu.edu.cn; Jian Wang (corresponding author), School of Computer Science, Wuhan University, Wuhan, Hubei, China and Zhongguancun Laboratory, Beijing, China; e-mail: jianwang@whu.edu.cn; Bing Li, School of Computer Science, Wuhan University, Wuhan, Hubei, China and Zhongguancun Laboratory, Beijing, China; e-mail: bingli@whu.edu.cn; Liuxiaoxiao Zhang, R&D Center, Taikang Insurance Group, Wuhan, China; e-mail: zhanglxx@taikanglife.com; Yu Liu, School of Computer Science, Wuhan University, Wuhan, Hubei, China; e-mail: liuyuforwh@gmail.com; Patrick Hung, Ontario Tech University, Oshawa, Ontario, Canada; e-mail: patrick.hung@ontariotechu.ca.



ACM 1559-1131/2026/04-ART21
https://doi.org/10.1145/3801729

1 Introduction

The rapid development of the Internet has accelerated the accumulation of resources, including computing, storage, networking, services, and data. Traditional monolithic architectures are becoming increasingly inadequate for managing this complexity [32]. Microservice architectures, offering independent deployment and scalability, have become the preferred choice for building modern software applications. However, as the number of microservices grows rapidly, issues such as network latency, version incompatibility, and load-balancing failures become more pronounced. These issues not only disrupt service communication but can also trigger cascading failures, leading to performance degradation and increased difficulty in fault diagnosis [1]. The increasing complexity and interdependence of microservices make it difficult to understand system behavior through intuition alone. Failures may propagate silently across services, and performance degradations can remain undetected until they impact end users. To effectively manage such systems, operators require comprehensive observability data that provides timely and detailed insights into the internal state of each service.

Observability, which is typically achieved through metrics, logs, and traces, serves as the foundation for detecting, diagnosing, and mitigating failures in microservice systems. **Metrics** provide real-time, quantitative snapshots of system health, such as CPU usage, memory consumption, and network delay. **Logs** capture detailed system events, offering context for debugging and root cause analysis. **Traces** record request flows across services, revealing inter-service dependencies and invocation patterns. By integrating these complementary data sources, operators can achieve comprehensive monitoring and effective anomaly detection.

Despite its importance, acquiring complete observability data in real-world microservice systems is challenging. Especially for metrics, the collection is often disrupted by (*i*) network instability, causing delays or packet loss; (*ii*) frequent instance restarts, interrupting ongoing data gathering; (*iii*) distributed storage failures, leading to delayed or lost writes; and (*iv*) misconfigurations in monitoring components. These challenges often result in incomplete monitoring data, particularly for continuous metrics. The incompleteness hampers fault detection and recovery, ultimately affecting system reliability and maintainability. Real-world incidents exemplify this issue: in a 2023 production event,[1] GitLab's observability stack (Prometheus + Thanos) suffered hours-long metric loss due to memory saturation in the Thanos rule process. Similarly, the 2023 Cloudflare outage report[2] highlighted that the loss of internal observability signals impeded rapid anomaly detection and fault localization. These cases illustrate that metric loss is not merely a technical inconvenience but a direct factor that increases **Mean Time To Recovery** (**MTTR**) in microservice operations.

Existing anomaly detection methods for microservice systems have explored metrics-based, log-based, or trace-based approaches [5, 6, 8, 10, 18, 22, 26, 29, 41, 45, 46, 56], as well as more recent multimodal approaches that integrate these three data sources [7, 24, 25, 37, 52, 55]. While effective under complete observability, their performance can degrade significantly when such data is incomplete. On the other hand, several imputation-based anomaly detection methods [13, 53], while designed to mitigate missing data issues, are not tailored for microservice systems. Consequently, they often overlook distinctive characteristics such as inter-service dependencies and cross-modality correlations, which can lead to distorted imputations and inaccurate anomaly detection results.

To address the above issues, we propose **Anomaly Detection model with missing Metric data** (**ADmM**), an anomaly detection method for microservice systems with incomplete metrics. ADmM begins by extracting modality-specific information from logs, metrics, and traces. It refines log features at both coarse and fine granularity using log templates and semantic representations

[1] https://gitlab.com/gitlab-com/gl-infra/production/-/issues/8345
[2] https://blog.cloudflare.com/post-mortem-on-cloudflare-control-plane-and-analytics-outage

while normalizing metrics and traces to improve feature quality. It then employs multi-scale **Convolutional Neural Networks** (**CNN**) to fuse multimodal features, enabling comprehensive system state representation. To handle missing metric data, ADmM incorporates a Transformer-based autoencoder for imputation, ensuring robustness in the presence of incomplete observations. Furthermore, ADmM applies **Graph Neural Networks** (**GNN**) to propagate information across services, incorporating contextual dependencies from upstream and downstream components. This enhances anomaly detection by leveraging system-wide interactions. Finally, a generative model learns the normal patterns of the system and estimates the likelihood of observed states. The **anomaly scores** (**AS**) are then assigned based on deviations from learned distributions, enabling precise anomaly detection. It is worth clarifying the scope of this work that we focus on anomaly detection under incomplete observability data, rather than identifying whether observability data is missing. The detection of missing or corrupted observability signals itself, which may require dedicated monitoring or data quality assessment mechanisms, is considered out of the scope of this article. We assume that the absence of metric data is already known to the system and aim to study how anomaly detection can remain effective under such incomplete observations.

The main contributions of ADmM are summarized as follows:

(1) We design a metric imputation model that integrates various observability sources of microservices. By leveraging multi-scale dependencies across different modalities, the model captures both high-level patterns and fine-grained details of system behavior. Then, it uses an encoder-decoder architecture with temporal-aware representations to reconstruct missing metric data.
(2) We propose a novel anomaly detection framework with incomplete metric data named ADmM. ADmM learns the structural and behavioral patterns of normal samples by modeling evolving microservice dependencies. It then performs joint learning to reconstruct expected observations and uses the discrepancy between observed and reconstructed values as the basis for computing AS.
(3) We conduct extensive experiments on three open-source benchmarks, and the results demonstrate that ADmM demonstrates superior performance with incomplete metric data, confirming the robustness and effectiveness of ADmM. In addition, the source code of ADmM is publicly available on GitHub.[3]

The rest of this article is structured as follows: Section 2 gives the background and motivation. Section 3 provides a detailed explanation of ADmM. Section 4 presents and analyzes experimental results on three datasets. Section 5 describes related work. Finally, Section 6 summarizes our work and outlines future directions.

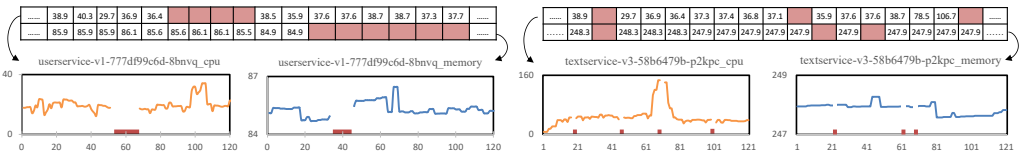
2 Background and Motivation

2.1 Handling Incomplete Metrics in Microservice Systems

In real-world microservice systems, monitoring data typically passes through three stages. Microservices first generate various observability signals, including logs, metrics, and traces. These data are then collected and stored by monitoring platforms such as Prometheus,[4] Filebeat,[5] and Jaeger.[6] Finally, anomaly detection applications access the collected data for analysis and modeling.

In practical monitoring scenarios, data loss frequently occurs as a consequence of metric collection delays, transient network instability, irregular sampling intervals, and the dynamic behavior of

[3] https://github.com/WHU-AISE/ADmM
[4] https://prometheus.io
[5] https://www.elastic.co/beats/filebeat
[6] https://www.jaegertracing.io



(a) Chunk-wise incompleteness in metric data (b) Point-wise incompleteness in metric data

Fig. 1. Example of incomplete metric data in a microservice system, illustrating challenges in real-world observability.

Table 1. Detection Performances Over Diverse Imputation Methods

Method \ Dataset	Social Network			Train-Ticket			GAIA		
	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
mean	0.4112	0.4373	0.4238	0.4725	0.3818	0.4223	0.5665	0.6095	0.5872
ffill	0.4896	0.4625	0.4757	0.4395	0.4285	0.4339	0.6095	0.6842	0.6477
bfill	0.5087	0.4348	0.4689	0.3703	0.3794	0.3748	0.5764	0.5707	0.5735
0-fill	0.4242	0.3755	0.3984	0.4045	0.4012	0.4028	0.5344	0.5166	0.5253
max-fill	0.4102	0.3363	0.3696	0.3825	0.3808	0.3816	0.5265	0.5065	0.5163
complete observability	**0.6320**	**0.6751**	**0.6528**	**0.6062**	**0.578**	**0.5918**	**0.7538**	**0.7480**	**0.7509**

Bold values indicate the best performance, and underlined values indicate the second-best performance.

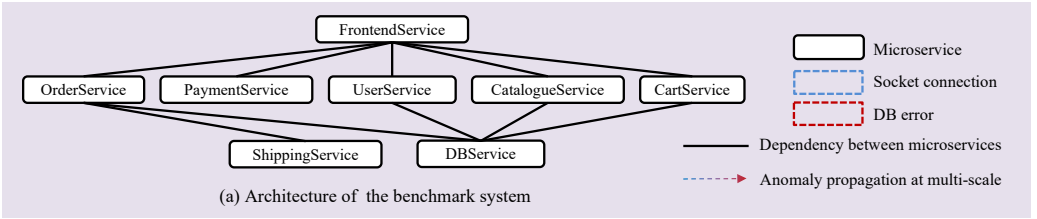
containers and Pods, such as migrations and restarts. These missing values often create observability blind spots at the monitoring layer rather than reflecting actual service faults. For example, the absence of a CPU metric at a specific time does not necessarily imply a CPU malfunction or directly affect the success rate of user requests.

However, missing data poses a challenge to the observability completeness of microservice systems, with metric data loss being the most common issue. This significantly impacts the accuracy of anomaly detection results. Currently, there is no unified standard in the literature for categorizing missing data in microservice systems. Based on engineering practice, prior studies [36, 43], and an analysis of several commonly used microservice benchmark datasets (**Social Network** (**SN**),[7] **Train-Ticket** (**TT**),[8] and Generic AIOps Atlas (**GAIA**),[9]) we identify two common forms of metric data incompleteness: chunk-wise and point-wise incompleteness. Figure 1 illustrates incomplete metric data collected from two different microservice instances in the SN benchmark, where the missing segments are highlighted in red. Such gaps compromise the integrity of the system's data, making it more difficult to accurately detect anomalies and pinpoint their root causes.

Typical approaches to handling incompleteness of metrics include deletion and imputation [11]. Deletion is typically employed when the proportion of missing data is minimal, as removing a few missing values has little impact on downstream tasks. Imputation techniques such as **forward filling** (**ffill**), **backward filling** (**bfill**), mean filling (mean), **zero filling** (**0-fill**), and **maximum-value filling** (**max-fill**) are widely used and particularly effective for datasets with smooth and predictable temporal trends [2, 35, 38]. However, these methods often struggle with multimodal data that exhibit complex relationships.

To evaluate the effectiveness of existing imputation strategies in microservice systems, we applied these methods to the aforementioned benchmark datasets and employed the classic SVM [3] for anomaly detection. Table 1 presents the anomaly detection results under complete observability

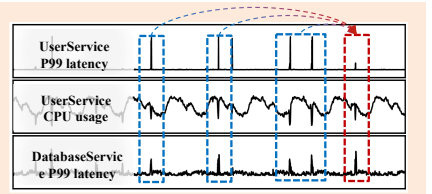
[7]https://github.com/delimitrou/DeathStarBench/tree/master/socialNetwork
[8]https://github.com/FudanSELab/train-ticket
[9]https://github.com/CloudWise-OpenSource/GAIA-DataSet



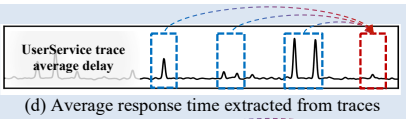
(a) Architecture of the benchmark system

Time	Level	Class name	Description
14:32:16	DEBUG	UserService	Fetching user profile from cache key=user:john_doe
14:32:20	INFO	UserService	user_id=john_doe action=visit url="/products/t-shirt"
14:32:22	DEBUG	CatalogueService	Fetching product details product_id=TSHIRT-123
14:32:25	INFO	UserService	user_id=doe action=add_to_cart product_id=TSHIRT-123
14:32:26	DEBUG	CartService	Added item to cart cart_id=CART-555 item_count=3
14:32:30	INFO	UserService	user_id=doe action=checkout
14:32:31	INFO	OrderService	user_id=doe order_id=ORD-98765 status=initiated
14:32:33	DEBUG	OrderService	Preparing database transaction for order_id=ORD-98765
14:32:35	WARN	DataBaseService	Can't Get DB Connection error=SocketException
14:32:36	WARN	DataBaseService	Retrying database connection attempt=1 backoff=2s
14:32:39	WARN	DataBaseService	Can't Get DB Connection error=TimeoutException
14:32:40	WARN	DataBaseService	Retrying database connection attempt=2 backoff=3s
14:32:44	WARN	DataBaseService	Can't Get DB Connection error=TimeoutException
14:32:45	ERROR	DataBaseService	Max retry attempts reached, marking as UNAVAILABLE
14:32:46	WARN	OrderService	order_id=ORD-98765 Database unavailable
14:32:52	ERROR	DataBaseService	Fatal error: Cannot establish DB connection
14:32:54	ERROR	OrderService	order_id=ORD-98765 Transaction failed, aborting order
14:32:55	INFO	DataBaseService	Transaction rollback order_id=ORD-98765
14:32:55	WARN	OrderService	Compensation order_id=ORD-98765 error="DB failure"
14:32:57	ERROR	PaymentService	Refund failed order_id=ORD-98765 error="No record"

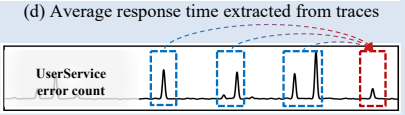
(b) Log sequences



(c) Metric sequences



(d) Average response time extracted from traces



(e) Service error count from extracted traces

Fig. 2. Multi-scale anomaly propagation from minor fluctuations to critical failures across logs, metrics, and traces.

(labeled as complete observability in the table), as well as the results under a scenario where 40% of the metric data is missing and different imputation methods are applied. Among them, the ffill outperforms other approaches, while the max-fill imputation yields the worst results. Although the ffill imputation outperforms other methods, there is still a significant gap compared to the complete observability. It is essential to develop an effective imputation method for incomplete metrics in microservice systems.

2.2 Multi-scale Impact of Anomalies in Microservice Systems

Anomalies in microservice systems often exhibit multi-scale characteristics, meaning their impacts on observability data manifest differently across time scales. As illustrated in Figure 2, we take a database connection issue in the benchmark system (Sock-shop[10]) as a case study to show how such problems propagate across scales. Specifically, we analyze a one-minute window of logs, metrics, and trace data collected between 14:32 and 14:33. For the UserService, P99 latency denotes the 99th percentile of response latency, while CPU usage reflects its current utilization. We focus on UserService, OrderService, and DatabaseService because they best capture the evolution of the connection issue.

As the socket request rate approaches and exceeds the operating system's capacity, the system first exhibits subtle fluctuations. These are visible in the blue log entries in Figure 2(b) and the blue boxes in Figures 2(c)–2(e), which show minor jitters in response latency and related metrics. Such variations serve as early warning signals of instability. If left unresolved, the system comes under increasing stress, leading to a growing number of connection errors. These errors are highlighted

[10]https://github.com/ocp-power-demos/sock-shop-demo

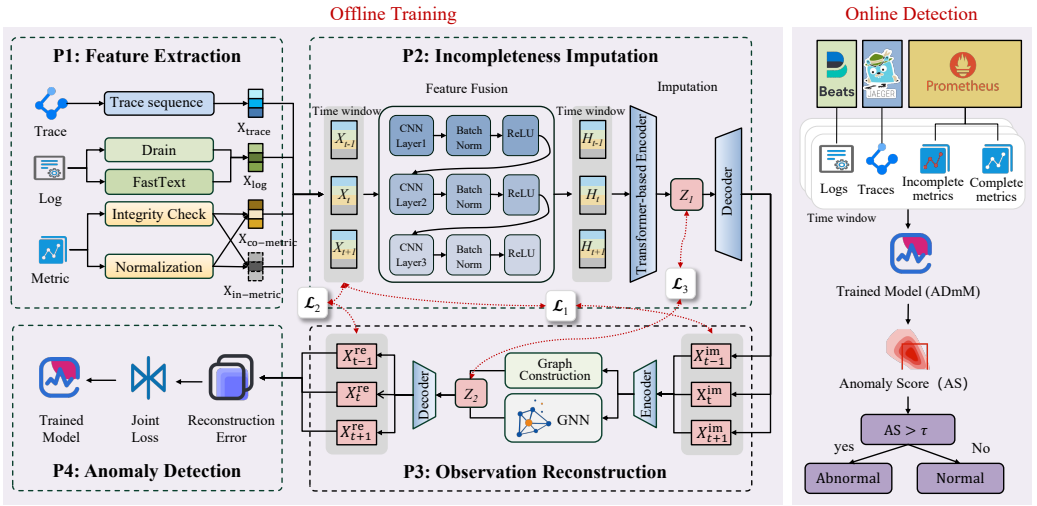


Fig. 3. Overall framework of ADmM.

by the red entries in the same figures. Importantly, connection issues often emerge gradually, progressing from localized disturbances to system-wide problems.

With sustained overload, the system may eventually experience bursts of failures across multiple scales, such as surges in socket errors or widespread connection breakdowns. If unchecked, this can culminate in the termination of all socket connections and complete database failure, pushing the system into a critical fault state.

Therefore, capturing and modeling these multi-scale variations is essential. It not only enables accurate reconstruction of incomplete observability data but also improves detection accuracy and enhances overall system reliability.

3 ADmM

3.1 Overall Framework

3.1.1 Overall Framework. Figure 3 illustrates the complete architecture of ADmM, detailing the internal modules, learning objectives, and interactions among different components during offline training and online detection. The offline training stage consists of four main components: (P1) feature extraction, (P2) incompleteness imputation, (P3) observation reconstruction, and (P4) anomaly detection.

P1. Feature extraction involves processing logs, metrics, and traces of the system to capture their unique characteristics. This step simplifies raw data into multi-dimensional feature vectors that represent the system's state, providing a foundation for subsequent analysis.

P2. In the incompleteness imputation phase, ADmM incorporates a multi-scale CNN to fuse multimodal features. Subsequently, we extract the relationships among multimodal data collected in normal states and fill in incomplete metrics using a transformer-based AE model.

P3. In the observation reconstruction phase, ADmM leverages a GNN to capture the dependencies between instances in the microservice system. We apply an optimization process to guide data reconstruction by measuring the difference between observed and reconstructed values.

P4. ADmM computes an AS during the detection phase by comparing reconstructed values with the observed data. The loss function is designed to minimize the gap between them, enabling the model to effectively differentiate normal samples from abnormal ones.

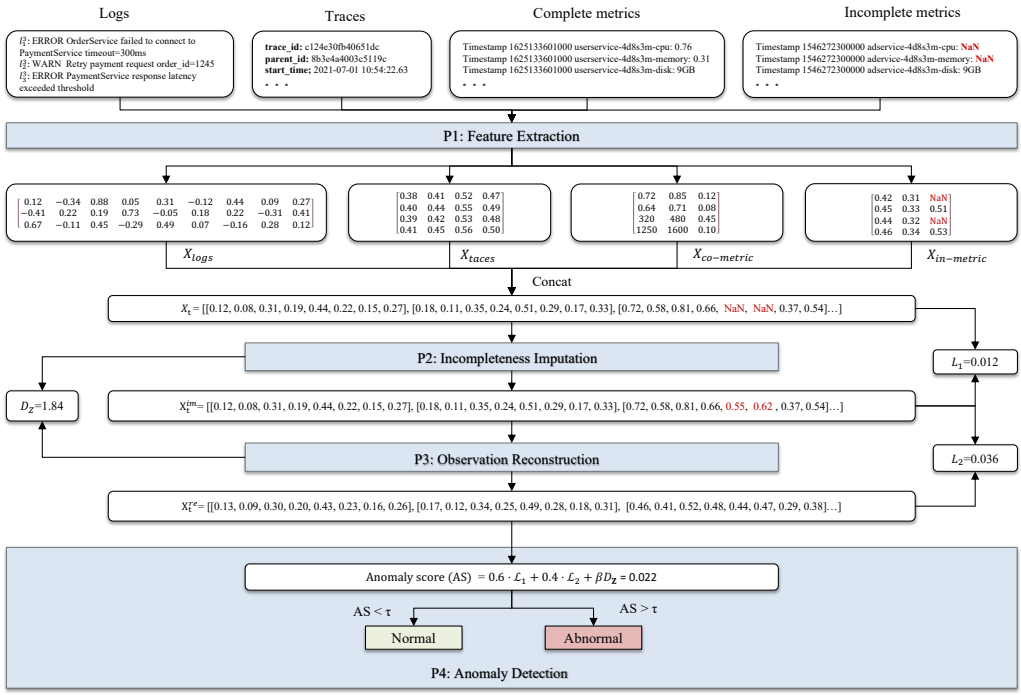


Fig. 4. A simplified running example illustrating the workflow of ADmM.

During the online detection stage, the model receives three types of observability data collected from different components of the monitoring stack. Logs are collected through Filebeat, metrics are gathered via Prometheus, and traces are obtained from Jaeger. These heterogeneous data streams are first pre-processed and organized into fixed-length time windows, which preserve the temporal relationships among the different modalities. The windowed data are then fed into the ADmM model, where multimodal feature extraction and fusion are performed to capture both intra-service behaviors and inter-service dependencies. Finally, the model generates an AS for each time window, which is then used to automatically determine whether the microservice environment is in a normal or anomalous state, thereby enabling real-time detection and prompt response to potential failures.

3.1.2 Running Example. To provide an intuitive understanding of how ADmM processes heterogeneous observations and handles incomplete metrics, we present a simplified running example in Figure 4.

Consider a time window in which a microservice system produces three types of observability data: logs, traces, and metrics. The log stream contains warning and error messages indicating abnormal behaviors such as increased response latency or failed service invocations. The trace data records invocation information, including trace identifiers, parent—child relationships, and timestamps. Meanwhile, the metric stream reports resource usage statistics, where some metric values (e.g., CPU and memory usage) are missing due to monitoring failures or transient collection issues.

In the feature extraction stage (P1), raw logs, traces, and metrics are transformed into fixed-length feature representations, denoted as X_{logs}, X_{traces}, $X_{\text{co-metric}}$, and $X_{\text{in-metric}}$, respectively. These modality-specific features are then concatenated to form a unified multimodal observation vector X_t, in which missing metric entries are explicitly represented as undefined values.

In the incompleteness imputation stage (P2), ADmM infers plausible values for the missing entries by exploiting cross-modal correlations and temporal dependencies, yielding an imputed representation X_t^{imp}. For instance, missing metric values are recovered based on correlated log patterns and trace behaviors observed within the same time window.

Next, in the observation reconstruction stage (P3), the imputed representation is further refined by incorporating service dependency information through graph-based reconstruction, producing a reconstructed observation X_t^{re} that characterizes expected system behavior under normal conditions.

Finally, in the anomaly detection stage (P4), ADmM computes an AS by synthesizing the discrepancies among the original observation X_t, its imputed counterpart X_t^{im}, and its reconstruction X_t^{re}. A system state is then classified as anomalous if the reconstruction error surpasses a set threshold, and normal otherwise.

All numerical values shown in Figure 4 are fictive and provided solely for illustration purposes, aiming to clarify the data flow and intermediate representations rather than to reflect exact model outputs.

The following sections provide a detailed explanation of each component in ADmM.

3.2 Multimodal Feature Extraction

First, we extract features tailored to the unique characteristics of each type of observable data.

3.2.1 Feature Extraction for Logs. The process consists of three steps:

(a) Template feature extraction. In a microservice system, abnormal events may lead to significant changes in log templates, resulting in a large volume of anomaly logs.

Let the entire log stream be divided into a sequence of W consecutive time windows, indexed by $w \in \{1, 2, \dots, W\}$. For each time window w, we collect all raw log entries observed within that window and denote this set as $L_{original}^{(w)} = \{l_1^{(w)}, l_2^{(w)}, \dots, l_{N_w}^{(w)}\}$, where N_w is the number of log entries in window w.

We then use Drain [17], a pre-trained template model, to extract log templates for each log entry:

$$L_{template}^{(w,i)} = Drain\big(l_i^{(w)}\big), \tag{1}$$

where $L_{template}^{(w,i)}$ is the template of the ith log entry in time window w.

Each log template is directly mapped to a dense vector representation through an embedding function:

$$\mathbf{L}_{id}^{(w,i)} = g\big(L_{template}^{(w,i)}\big), \quad \mathbf{L}_{id}^{(w,i)} \in \mathbb{R}^{d_{id}}, \tag{2}$$

where $g(\cdot)$ combines template indexing and embedding lookup, producing a d_{id}-dimensional vector representation for each log template.

(b) Semantic feature extraction. While templates capture structural changes, they ignore the semantic information within logs, such as key tokens indicating failure causes. To model this fine-grained information, we train a FastText [12] embedding model on historical logs and use TF-IDF weighting to highlight important words.

For the ith log entry in time window w, assume it contains $n_{w,i}$ tokens $\{o_1^{(w,i)}, \dots, o_{n_{w,i}}^{(w,i)}\}$. The **term frequency** (**TF**) of token o is:

$$TF^{(w,i)}(o) = \frac{\text{count}^{(w,i)}(o)}{N^{(w,i)}}, \tag{3}$$

where $\text{count}^{(w,i)}(o)$ is the count of token o in the ith log entry of window w, and $N^{(w,i)}$ is the total number of tokens in that log entry.

The **inverse document frequency** (**IDF**) of token o is:

$$IDF(o) = \log\left(\frac{n_{doc}}{n_o}\right) + 1, \tag{4}$$

where n_{doc} is the total number of log entries in the entire training corpus, and n_o is the number of log entries containing token o.

The TF-IDF value is then computed as:

$$tfidf^{(w,i)}(o) = TF^{(w,i)}(o) \cdot IDF(o). \tag{5}$$

Each token o has a pre-trained embedding vector $\mathbf{e}(o) \in \mathbb{R}^{d_{sen}}$. The semantic embedding for the ith log entry in window w is computed as a weighted sum of its token embeddings:

$$\mathbf{L}_{sen}^{(w,i)} = \sum_{j=1}^{n_{w,i}} tfidf^{(w,i)}(o_j^{(w,i)}) \cdot \mathbf{e}(o_j^{(w,i)}), \tag{6}$$

where d_{sen} is the dimension of the semantic embedding. This formulation highlights semantically important words while downweighting common or uninformative tokens.

(c) Concatenation. Finally, for each log entry, we concatenate its template-level and semantic-level representations:

$$\mathbf{X}_{log}^{(w,i)} = \left[\mathbf{L}_{id}^{(w,i)}; \mathbf{L}_{sen}^{(w,i)}\right], \tag{7}$$

where $[\cdot;\cdot]$ denotes vector concatenation, resulting in a log embedding of dimension $d_{id} + d_{sen}$.

For each time window w, we aggregate the embeddings of all N_w logs into a matrix:

$$\mathbf{X}_{log}^{(w)} = \left[\mathbf{X}_{log}^{(w,1)}; \mathbf{X}_{log}^{(w,2)}; \ldots; \mathbf{X}_{log}^{(w,N_w)}\right] \in \mathbb{R}^{N_w \times (d_{id}+d_{sen})}. \tag{8}$$

This hierarchical structure preserves both the fine-grained ordering of logs within each window and the temporal progression across windows, providing a comprehensive input for downstream anomaly detection models.

3.2.2 Feature Extraction for Metrics. Metrics such as KPIs and resource utilization aggregated within a time window are used as feature representations. The representation of complete metrics (co-metric) within each time window w can be expressed as:

$$\mathbf{X}_{\text{co-metric}}^{(w)} = \begin{bmatrix} x_1^{(w)} \\ x_2^{(w)} \\ \vdots \\ x_{M_w}^{(w)} \end{bmatrix} \in \mathbb{R}^{M_w \times d_{\text{metric}}}, \tag{9}$$

where M_w is the number of metric entries in window w, and d_{metric} is the feature dimension of each metric entry.

For the incomplete metric (in-metric) data $\mathbf{X}_{\text{in-metric}}^{(w)}$, we define a masking matrix $\mathbf{M}$ to indicate which metric data points are missing. This masking matrix $\mathbf{M}$ has the same shape as the metric data $\mathbf{X}_{\text{co-metric}}$, where missing values are represented by 0, and complete data are represented by 1:

$$\mathbf{M}_{ij} = \begin{cases} 1, & \text{if the metric data at position } (i,j) \text{ is complete,} \\ 0, & \text{if the metric data at position } (i,j) \text{ is missing.} \end{cases} \tag{10}$$

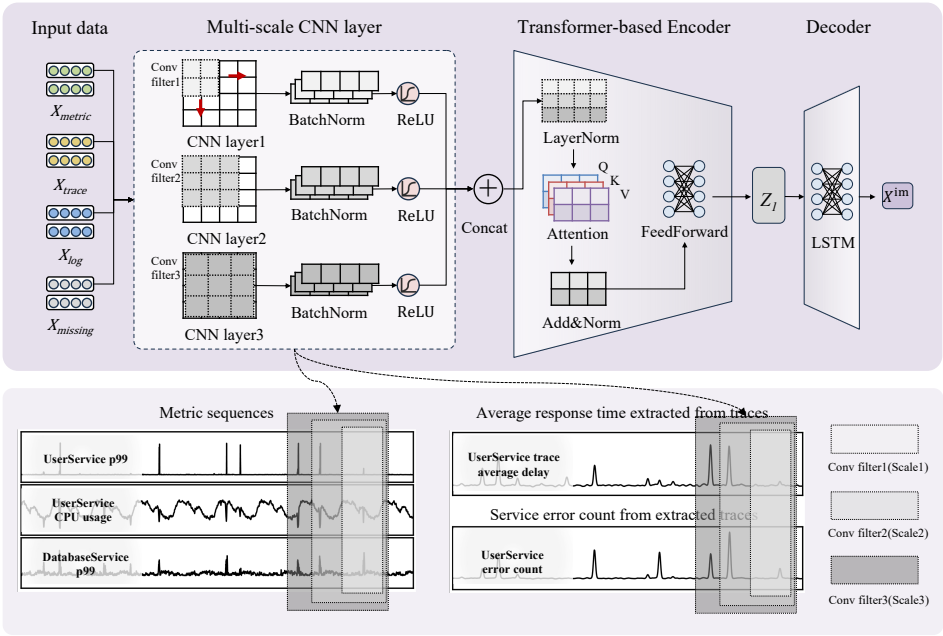


Fig. 5. Metric imputation module with a multi-scale convolution and a transformer-based encoder.

3.2.3 Feature Extraction for Traces. Each trace in a microservice system is composed of a sequence of spans, where each span represents a single service call with its latency and other metadata (e.g., service name and timestamp).

To unify their representation, we construct a trace feature matrix for window $w \in \{1, 2, \ldots, W\}$:

$$\mathbf{X}_{trace}^{(w)} = \begin{bmatrix} s_{1,1}^{(w)} & s_{1,2}^{(w)} & \cdots & s_{1,m_{\max}}^{(w)} \\ s_{2,1}^{(w)} & s_{2,2}^{(w)} & \cdots & s_{2,m_{\max}}^{(w)} \\ \vdots & \vdots & \ddots & \vdots \\ s_{T_w,1}^{(w)} & s_{T_w,2}^{(w)} & \cdots & s_{T_w,m_{\max}}^{(w)} \end{bmatrix}, \tag{11}$$

where $m_{\max} = \max_k m_k$ is the maximum trace length in window w, and traces shorter than $m_{\max}$ are padded with placeholder values (e.g., 0) to ensure consistent dimensions.

3.3 Imputation of Incomplete Metrics

As described in Section 2.2, anomalies in microservice systems often exhibit multi-scale characteristics. We use convolutional kernels of different scales to extract features from multimodal data. Large-scale kernels are applied to capture slowly varying signals, such as pressure changes, while small-scale kernels are more suitable for rapidly fluctuating signals [4, 9]. Next, the multimodal features are projected into a lower-dimensional space, and the incomplete data is reconstructed using an encoder-decoder architecture. As illustrated in Figure 5, this module consists of three core components: a multi-scale convolutional feature extractor, a Transformer-based encoder, and an LSTM-based decoder. The detailed description of each component is as follows.

3.3.1 Feature Fusion. For each modality m (log, co-metric, trace, and in-metric), within the time window W, we extract the features of the input data $\mathbf{X}_m$ using a multi-scale 2D convolution,

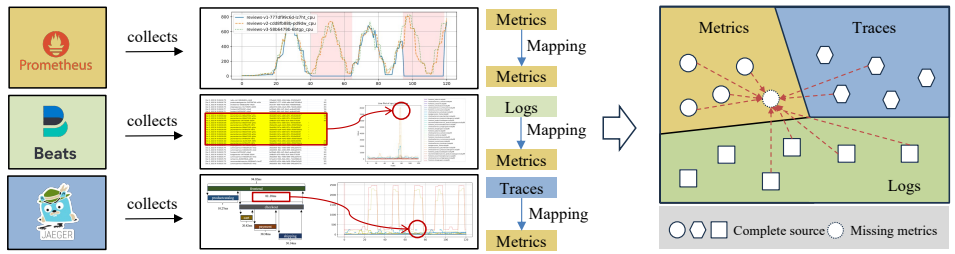


Fig. 6. The imputation method fills in incomplete metric data using modalities such as logs, traces, and metrics.

followed by the batch normalization and the ReLU activation:

$$\mathbf{H}^i_{m,conv} = Conv2d(\mathbf{X_m}), \tag{12}$$

$$\mathbf{H}^i_{m,bn} = BatchNorm2d(\mathbf{H}^i_{m,conv}), \tag{13}$$

$$\mathbf{H}^i_{m,relu} = ReLU(\mathbf{H}^i_{m,bn}), \tag{14}$$

where i denotes the convolutional layer index, $Conv2d$ refers to the convolution operation applied at different scales, $BatchNorm2d$ denotes the 2D batch normalization, and $ReLU$ is the activation function. The outputs from all convolutional layers are then concatenated to produce a comprehensive feature representation that captures information across all scales:

$$\mathbf{H}_{m,concat} = [\mathbf{H}^i_{m,relu}, \mathbf{H}^i_{m,relu}, ..., \mathbf{H}^i_{m,relu}]. \tag{15}$$

Finally, the features from all modalities are concatenated to form a multimodal representation:

$$\mathbf{H}_{multimodal} = [\mathbf{H}_{log,concat}, \mathbf{H}_{co\text{-}metric,concat}, \mathbf{H}_{trace,concat}, \mathbf{H}_{in\text{-}metric,concat}]. \tag{16}$$

3.3.2 Imputation Encoder. The observable data of microservices often exhibit long-term and deep dependencies. For example, the CPU usage of a microservice may be influenced by the burst workload of other microservices that occurred seconds or even minutes earlier. Furthermore, multimodal data can contain complex patterns, such as periodicity and trends, which correlate with different modalities, as shown in Figure 6. These correlations serve as an essential clue to calculate incomplete metrics. Therefore, we utilize a Transformer-based encoder to effectively model the dependencies across the log, co-metric, and trace data to impute incomplete metrics. The multimodal feature representation $\mathbf{H}_{multimodal}$ serves as an input to the Transformer.

To embed temporal positional information into the Transformer architecture and address its inherent lack of sequential context, we apply a linear projection and add a learnable positional encoding:

$$\mathbf{H}_0 = \mathbf{H}_{multimodal} W_e + \mathbf{P}, \tag{17}$$

where W_e is the learnable projection matrix, and $\mathbf{P}$ represents the positional encoding. Each Transformer encoder layer consists of a **multi-head self-attention** (**MHSA**) mechanism and a **feedforward network** (**FFN**):

$$\mathbf{Q} = \mathbf{H}_l W_Q, \quad \mathbf{K} = \mathbf{H}_l W_K, \quad \mathbf{V} = \mathbf{H}_l W_V, \tag{18}$$

$$\mathbf{A} = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \tag{19}$$

where $\mathbf{Q}$,$\mathbf{K}$,$\mathbf{V}$ denote the query, key, and value representations, respectively, which are obtained by linearly projecting the input feature $\mathbf{H}_l$ trough the learnable matrices W_Q, W_K, W_V. The attention

matrix $\mathbf{A}$ captures the pairwise dependency between timesteps, and d_k is the key dimension. The output of MHSA undergoes the layer normalization and passes through an FFN:

$$\mathbf{H}_{l,attn} = \text{LayerNorm}(\mathbf{A} + \mathbf{H}_{l-1}), \tag{20}$$

$$\mathbf{H}_{l,ffn} = \text{LayerNorm}(\text{ReLU}(\mathbf{H}_l W_1 + b_1)W_2 + b_2 + \mathbf{H}_l), \tag{21}$$

where $\mathbf{H}_{l-1}$ denotes the output of the previous layer, W_1 and W_2 are the learnable weight matrices of the FFN, and b_1 and b_2 are the corresponding bias terms. The output of the final (i.e., Lth) Transformer-based encoder layer is denoted as $\mathbf{H}_{trans}$.

3.3.3 Imputation Decoder. The decoder transforms the fused representation $\mathbf{H}_{trans}$ into a lower-dimensional latent representation $\mathcal{Z}_1$ using a **multi-layer perceptron** (**MLP**):

$$\mathcal{Z}_1 = \sigma(W_e \mathbf{H}_{trans} + b_e), \tag{22}$$

where W_e and b_e are the weight matrix and bias vector of the MLP, respectively, and σ denotes an activation function.

Then, the decoder reconstructs the incomplete metric data using an LSTM network:

$$h_t = f_{\text{lstm}}(h_{t-1}, \mathcal{Z}_1), \tag{23}$$

$$\hat{x}_t = \sigma(W_o h_t + b_o), \tag{24}$$

$$\hat{\mathbf{X}}_i^{im} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_W]. \tag{25}$$

where W_o and b_o are the output weight and bias, $\hat{x}_t$ is the predicted metric value at timestep t, and $\hat{\mathbf{X}}_i^{im}$ is the reconstructed metric sequence of length W for the ith sample.

We introduce a masking mechanism to refine the optimization objective to ensure that the model optimizes only the incomplete metrics and avoids error propagation. The loss function is defined as:

$$\mathcal{L}_1 = \sum_i \mathbf{M}_i \cdot \left\| \mathbf{X}_i^{in\text{-}metric} - \hat{\mathbf{X}}_i^{im} \right\|^2, \tag{26}$$

where $\mathbf{M}_i$ represents the set of indices corresponding to missing values in the metric data, $\hat{\mathbf{X}}_i^{\text{im}}$ denotes the reconstructed values for the missing metrics, $\mathbf{X}_i^{\text{in-metric}}$ represents the real values. By minimizing $\mathcal{L}_1$ using the Adam optimizer [21], we can obtain imputed data closer to the complete observability. The process of imputing incomplete metrics is shown in Algorithm 1.

3.4 Observation Reconstruction Model

After imputing incomplete metrics, we further implemented the anomaly detection process. First, a GNN captures dependencies between services, constructing a global state-pattern representation of the system. This provides a comprehensive view of the system's health. Next, a generative model extracts features from the latent space of normal samples. Given an observed sample, the model evaluates its likelihood of being reconstructed as normal. Since anomalies typically exhibit larger reconstruction errors, we use this error as the AS. The detection process follows these steps:

3.4.1 Graph Encoder. The encoder consists of three parts:

(a) Graph construction: When microservices invoke remote procedures, they typically transmit metadata through HTTP or other communication protocols. The trace data captures service interactions, allowing the construction of a directed graph G with the source and target spans. In this graph, nodes V represent services, and edges E represent the call dependencies between them.

ALGORITHM 1: Imputation for incomplete metrics

Input: Multimodal input $\mathbf{X} = \{\mathbf{X}_{log}, \mathbf{X}_{co\text{-}metric}, \mathbf{X}_{trace}, \mathbf{X}_{in-metric}\}$, Mask matrix $\mathbf{M}$
Output: Latent representation of imputed metrics
for *each modality* $m \in \{log, co\text{-}metric, trace, in\text{-}metric\}$ **do**
  for $i = 1$ **to** *num_conv_layers* **do**
    $\mathbf{H}_i \leftarrow \texttt{ReLU}(\texttt{BatchNorm2d}(\texttt{Conv2d}(\mathbf{X})))$ ▷ Multi-scale CNN layer ;
    $\mathbf{H}_{concat}.append(\mathbf{H}_i)$;
$\mathbf{H}_{fusion} \leftarrow \text{Concat}(\mathbf{H}_{log,concat}, \mathbf{H}_{co\text{-}metric,concat}, \mathbf{H}_{trace,concat})$ ▷ Multimodal fusion;
$\mathbf{T}_0 \leftarrow \text{Linear}(\mathbf{H}_{fusion})$;
for $l = 1$ **to** *num_layers* **do**
  $\mathbf{T}_l \leftarrow \texttt{LayerNorm}(\mathbf{T}_{l-1} + \texttt{MultiHeadAttention}(\mathbf{T}_{l-1}))$ ▷ Transformer;
$\mathbf{H}_{trans} \leftarrow \texttt{LayerNorm}(\mathbf{T}_{num\_layers} + \texttt{FeedForward}(\mathbf{T}_{num\_layers}))$;
$\mathcal{Z}_1 \leftarrow \text{Encoder}(\mathbf{H}_{trans})$;
$\hat{\mathbf{X}}_{\text{in-metric}} \leftarrow \text{Decoder}(\mathcal{Z}_1)$;
$\mathcal{L}_1 \leftarrow \sum(\mathbf{M} \odot (\mathbf{X}_{\text{in-metric}} - \hat{\mathbf{X}}_{\text{im}}))^2$ ▷ Imputation loss ;
return $\mathcal{Z}_1$;

(b) Node feature extraction: Then, the features of microservice nodes are propagated based on the GNN model:

$$h_i^{(k)} = \text{MLP}^{(k)}\left((1 + e^{(k)}) \cdot h_i^{(k-1)} + \sum_{j \in N(i)} h_j^{(k-1)}\right), \tag{27}$$

where $h_i^{(k)}$ represents the feature of node i at layer k, $N(i)$ is the set of neighbors of node i, $e^{(k)}$ is a learnable parameter at layer k, and MLP refers to a multi-layer perceptron. The summation $\sum_{j \in N(i)} h_j^{(k-1)}$ gathers features from neighboring nodes in the previous layer. This operation captures the graph's topology by integrating each node's attributes with its neighbors. This update rule ensures that the GNN integrates structural information and node features during learning.

(c) State-pattern representation: A comprehensive representation of the entire graph is constructed by concatenating the features of all nodes after multiple iterations. The purpose of various iterations is to capture node dependencies at different levels: fewer iterations emphasize local patterns, while more iterations capture global and refined representations. The final graph representation is defined as follows:

$$\mathbf{H}_G = \text{Concat}\left(\left(\left\{h_i^{(r)} | i \in G\right\}\right) | r \in \{1, 2, 3, \cdots, R\}\right), \tag{28}$$

where $h_n^{(r)}$ represents the feature of node n after the rth iteration. The encoder transforms the fused representation $\mathbf{H}_G$ into a lower-dimensional latent representation $\mathcal{Z}_2$ using a fully connected layer:

$$\mathcal{Z}_2 = \sigma(W_e \mathbf{H}_G + b_e). \tag{29}$$

This process effectively integrates both local dependencies and global relationships within the microservice system, resulting in a more comprehensive representation.

3.4.2 MLP Decoder. Then, we use a fully connected layer for decoding, and the output feature $\hat{\mathbf{X}}_i^{re}$ can be expressed as:

$$\hat{\mathbf{X}}_i^{re} = \sigma(\mathcal{Z}_2 \mathbf{W} + \mathbf{b}), \tag{30}$$

where $\mathcal{Z}_2$ is the output from the encoder, and $\hat{\mathbf{X}}_i^{re}$ is the reconstructed feature predicted by the decoder. For the reconstruction of node features, we use the **Mean Squared Error (MSE)**

loss function:

$$\mathcal{L}_2 = \frac{1}{|N_s|} \sum_{i=1}^{N_s} \left\| \hat{\mathbf{X}}_i^{re} - \mathbf{X}_i \right\|^2, \tag{31}$$

where N_s denotes the number of samples, $\hat{\mathbf{X}}_i^{re}$ represents the generated feature of node i, and $\mathbf{X}_i$ is the observed sample. The optimizer (e.g., Adam) is employed to minimize the loss, and the model parameters are updated for reconstructing normal samples. The training process aims to learn a general latent representation for modal reconstruction, which is then used in anomaly detection tasks.

3.5 Anomaly Detection

3.5.1 Model Training. The imputation module and the reconstruction module focus on two distinct tasks. If the latent representation spaces of these two modules are too similar, the tasks may interfere with each other, reducing overall model performance. By maximizing the distance between these representations, the imputation module is encouraged to focus on addressing local data missing issues, while the reconstruction module concentrates on capturing the global structure of normal samples, thereby improving the effectiveness of both tasks. We use the L_1 norm to quantify the distance between $\mathcal{Z}_1$ and $\mathcal{Z}_2$, which is defined as:

$$D_{\mathcal{Z}} = \|\mathcal{Z}_1 - \mathcal{Z}_2\|_1. \tag{32}$$

The final training objective for anomaly detection can be expressed as:

$$\min \mathcal{L} = \lambda \mathcal{L}_1 + (1 - \lambda)\mathcal{L}_2 + \beta D_{\mathbf{Z}}, \tag{33}$$

where λ and β are hyperparameters, $\beta < 0$, and $|\beta| \ll 1$. To prevent divergence of the latent vector distance during training, β is typically set to -1×10^{-8} [13].

3.5.2 Anomaly Detection with Anomaly Scores. Following the imputation and reconstruction process, the model effectively reconstructs normal samples and struggles to reconstruct anomalous ones. Leveraging this disparity, we compute the reconstruction error as an AS for detecting system anomalies:

$$AS(\mathbf{X}) = \lambda \mathcal{L}_1(\mathbf{X}) + (1 - \lambda)\mathcal{L}_2(\mathbf{X}) + \beta D_{\mathbf{Z}}(\mathbf{X}). \tag{34}$$

The AS represents the probability of reconstructing the current data as normal using the generative model. In reconstruction-based anomaly detection models, normal samples are expected to have lower AS compared to anomalous samples.

ADmM is an unsupervised anomaly detection method that typically identifies anomalies using a threshold. Unlike traditional methods with fixed thresholds, we set a dynamic threshold based on the data distribution. The threshold is calculated as follows:

$$\tau = \mu + c \cdot \sigma, \tag{35}$$

where μ and σ represent the mean and standard deviation, respectively. The constant c is chosen based on the desired confidence level, commonly set to 3 (corresponding to 99.74%). The health status of the system is then assessed using the AS as follows:

$$\mathrm{AD}(\mathbf{X}) = \begin{cases} \text{Normal}, & AS(\mathbf{X}) \leqslant \tau \\ \text{Abnormal}, & AS(\mathbf{X}) > \tau \end{cases}, \tag{36}$$

where $\mathrm{AD}(\mathbf{X})$ represents the final result.

In addition, Algorithm 2 details the complete procedure for anomaly detection.

ALGORITHM 2: Anomaly detection with anomaly scores

Input: Multimodal input $\mathbf{X}$, Mask matrix $\mathbf{M}$, Hyperparameters λ, β, and threshold τ
Output: Anomaly status of the entire system

$\mathcal{L}_1 \leftarrow \sum(\mathbf{M} \odot (\mathbf{X} - \hat{\mathbf{X}}))^2$ ▷ Imputation loss ;
$\mathcal{L}_2 \leftarrow \frac{1}{M}\sum_{i=1}^{M} \|\hat{\mathbf{X}}_i^{re} - \mathbf{X}_i\|_1$ ▷ Reconstruction loss;
$\mathcal{L} \leftarrow \lambda\mathcal{L}_1 + (1-\lambda)\mathcal{L}_2 + \beta D_{\mathcal{Z}}$ ▷ Final loss ;
for *epoch in* 1 *to* N **do**
  for *each batch* $(\mathbf{X}, \mathbf{M})$ **do**
    $\hat{\mathbf{X}} \leftarrow f_\theta(\mathbf{X}, \mathbf{M})$ ▷ Forward pass;
    Update θ via backpropagation
$AS(\mathbf{X}) \leftarrow \lambda L_1 + (1-\lambda)L_2 + \beta D_{\mathcal{Z}}$ ▷ Anomaly score ;
if $AS(\mathbf{X}) \leq \tau$ **then**
  return Normal;
else
  return Abnormal;

4 Experiments

To evaluate the performance of our proposed method, we present extensive experiments on three open-source benchmarks to answer the following research questions:

— RQ1: How effective is ADmM with incomplete metrics?
— RQ2: What are the contributions of different modalities and components to anomaly detection?
— RQ3: How do different missing rates affect the detection accuracy?
— RQ4: How do parameters of ADmM affect its performance?
— RQ5: How efficient is ADmM in terms of training and testing cost?

4.1 Setups

4.1.1 Datasets. We evaluated ADmM on three widely used datasets for anomaly detection: SN, TT, and GAIA. Each dataset encompasses rich scenarios of real-world anomalies. Table 2 summarizes the dataset statistics, with detailed descriptions provided below:

— **SN** is a microservice-based benchmark system that simulates SN functionalities, with services communicating via REST and gRPC protocol. The system consists of 21 microservices interacting through Thrift [14], with 14 microservices directly related to business logic processing. We injected various types of faults into selected microservices using Chaos Mesh[11] to simulate system-level failures, including issues such as CPU saturation, network latency, and packet loss. This resulted in 641.5 MB of observability data, containing 2,862 traces, with each trace averaging 6–7 spans.
— **TT** is a microservice-based reservation system for TT, covering core functionalities such as user authentication, ticket inquiry, order management, and payment. It consists of 42 microservices communicating via gRPC protocol, with 27 instances directly involved in business processing. Similar to the SN dataset, we deployed the services in a Kubernetes cluster and injected failures using Chaos Mesh, resulting in 603.9 MB of observability data. The dataset contains 5,486 traces, with an average of seven spans per trace.

[11] https://chaos-mesh.org

Table 2. Dataset Characteristics

Dataset	Social Network	Train-Ticket	GAIA
Modality number	3	3	3
Anomaly number	63	99	1,620
Anomaly rate	4.46%	2.22%	6.32%
Metric dimensions	7	7	10,817
Metric length	1,412	6,462	18,824
Log entry number	40,184	80,942	48,938
Span number	38,400	17,172	12,960

— **GAIA** is a comprehensive dataset designed for AIOps (Artificial Intelligence for IT Operations), supporting tasks such as anomaly detection, log analysis, and fault localization. The dataset contains two weeks of continuously collected multimodal data from the MicroSS microservice system, which simulates the process of web-based QR code login. It involves 13 microservices deployed across five hosts. GAIA simulates a variety of real-world anomaly scenarios in microservice environments, such as login failures and memory anomalies. This ultimately generated 4.62 GB of observability data, which contains 2,592 traces with an average of five spans per trace.

The datasets used in our experiments include several common microservice anomalies, such as CPU overload, network delay, and network packet loss. These anomaly types reflect typical performance and communication issues that occur in real-world microservice environments. To simulate incomplete metrics, we applied targeted masking strategies to each dataset, generating missing data according to both chunk-wise and point-wise patterns, with the number of missing metrics kept balanced between the two patterns. The masking strategy was applied uniformly across all anomaly types to ensure that the evaluation covers a comprehensive range of abnormal behaviors.

To evaluate the accuracy of the reconstruction model, each dataset was split into a training set D_{train} and a testing set D_{test}. The testing set contains data that occur after the training set in time and is never used during training, ensuring temporal separation $D = D_{train} \cup D_{test}$, $D_{train} \cap D_{test} = \emptyset$. In our experiments, 80% of the data is used for training and the remaining 20% is allocated for testing.

4.1.2 Evaluation Metrics. We employed the Precision, Recall, and F1-score metrics to assess anomaly detection performance. These metrics are defined as follows:

$$Pre = \frac{TP}{TP + FP}, \tag{37}$$

$$Rec = \frac{TP}{TP + FN}, \tag{38}$$

$$F1 = \frac{2 \times Pre \times Rec}{Pre + Rec}, \tag{39}$$

where TP, FP, and FN represent true positives, false positives, and false negatives, respectively. Precision (*Pre*) quantifies the accuracy of positive predictions, while Recall (*Rec*) measures the model's ability to identify actual anomalies. The F1-Score balances *Pre* and *Rec* as their harmonic mean, providing a comprehensive evaluation.

In addition, to evaluate the impact of imputation accuracy on anomaly detection results, we use the MSE as the validation metric:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2, \tag{40}$$

where x_i denotes the true value of the ith missing entry, $\hat{x}_i$ represents the imputed value, and N is the total number of missing entries. A lower MSE indicates more accurate imputation.

4.1.3 Comparison Methods. To validate the performance of the proposed method, we compared ADmM with ten advanced baseline methods, including three unimodal-based methods and seven multimodal-based methods. Several methods depend on metric features, and for portions with incomplete metrics, ffill was employed as the imputation strategy. A brief introduction to these comparison methods is provided below:

Unimodal-based methods:

- **SwissLog** [26] is a log-based method that associates logs from different components by constructing a graph of identifier relationships. It then employs a dictionary-based log parsing to detect anomalies in microservice systems.
- **TraceAnomaly** [29] is a trace-based method. It constructs the **service-level trace vectors** (**STV**) that encodes microservice call paths and response times into feature vectors. The model then uses deep Bayesian networks for anomaly detection.
- **OmniAnomaly** [41] is a metric-based method that uses a stochastic recurrent neural network for anomaly detection. It learns a robust representation of the time series data to model normal behavior and detects anomalies by reconstructing the input data and evaluating the likelihood of reconstruction.

Multimodal-based methods:

- **DAM** [7] is an unsupervised anomaly detection method that combines log and metric data. It leverages an LSTM model to capture the interdependencies and feature representations between metrics and logs. By employing dynamic thresholds, DAM effectively detects anomalies without relying on labeled data.
- **Eadro** [24] is an end-to-end framework that integrates anomaly detection with root cause localization in a unified model. It uses three types of data sources: traces, logs, and performance metrics, and employs multimodal learning together with **Graph Attention Networks** (**GAT**) to model both intra-service behaviors and inter-service dependencies.
- **DeepTraLog** [52] is a deep learning-based anomaly detection method for microservices, which leverages a unified graph representation that captures the parallel invocation structures in traces, along with embedded log events. The generative model calculates an AS for the current system state.
- **AnoFusion** [55] is an unsupervised anomaly detection method. It combines metric, trace, and log data to construct a heterogeneous graph, which is then used to predict future data patterns. The deviation between the predicted and observed values is then used as a fault score to detect anomalies in the system.
- **MADMM** [42] is an anomaly detection method based on multimodal data for microservice systems. It first leverages **Graph Convolutional Networks** (**GCN**) and GAT to capture the spatial and contextual relationships of metrics and logs. Then, multi-grained contrastive learning is applied to extract robust cross-modal features for effective anomaly detection.
- **MM-SVM** [3] is an SVM-based anomaly detection method with multi-modality, which finds a hyperplane to separate positive and negative samples. It is particularly effective in scenarios with an imbalance between positive and negative samples in anomaly detection tasks.

Table 3. Comparison of Baseline Methods (AD: Anomaly Detection, RCA: Root Cause Analysis)

Method	Observability	Missing Data Robustness	Core Algorithm	Graph-based Dependency Modeling	Supervision	Detection Task	Application Domain
TraceAnomaly	Traces	Medium	Glow's flow + VAE	No	Unsupervised	AD	Microservice systems
SwissLog	Logs	Low	BERT+Bi-LSTM	No	Unsupervised	AD	Microservice systems
DyCause	Metrics	Low	Causal inference	No	Unsupervised	AD+RCA	Microservice systems
DAM	Metric,Logs	Low	LSTM	No	Unsupervised	AD	Microservice systems
DeepTralog	Logs,Traces	Low	Gated GNN	Yes	Unsupervised	AD	Microservice systems
AnoFusion	Metrics,Logs,and Traces	Medium	Graph transformer	Yes	Unsupervised	AD	Microservice systems
Eadro	Metrics,Logs,and Traces	Medium	Casual convolution+GAT	Yes	Supervised	AD+RCA	Microservice systems
MADMM	Metrics,Logs,and Traces	Medium	GRU+Contrastive learning	Yes	Unsupervised	AD	Microservice systems
KDOTS	-	High	Teacher + Student model	No	Unsupervised	AD	Liquid Rocket Engine
UU-Net	-	High	AutoEncoder	No	Unsupervised	AD	Liquid Rocket Engine
ADmM	Metrics,Logs,and Traces	High	Multi-scale CNN+GNN	Yes	Unsupervised	AD	Microservice systems

— **MM-iForest** [28] is an iForest-based anomaly detection method with multi-modality, which detects anomalies by isolating samples. It is commonly applied to continuous data, such as time series.

In addition, since there are few works on anomaly detection with incomplete data for microservice systems, we chose two detection methods in the industrial field:

— **KDOTS** [53] is an anomaly detection algorithm with incompelte observable data. It consists of a teacher model and a student model. The teacher model reconstructs missing data through estimation. The student model is used to identify anomalies based on the knowledge transferred from the teacher model.

— **UU-Net** [13] proposes an unsupervised multimodal method for anomaly detection in the scenario where some modal data of the liquid rocket engine system is missing. The model handles modal fusion and anomaly detection in a unified framework consisting of multiple autoencoders and a skip-connected autoencoder.

To further highlight the novelty of ADmM compared to baseline approaches, we perform a comprehensive comparison across several dimensions, including observability, core algorithm, missing data robustness, supervision, detection task, and applicable domain, as summarized in Table 3.

Single-modality methods, such as TraceAnomaly, SwissLog, and DyCause, suffer from limited representational capacity, making it difficult to capture the complex interactions within microservice systems. As system complexity increases, many studies have shifted toward integrating multiple data sources. Approaches such as AnoFusion, Eadro, and MADMM combine metrics, logs, and traces, providing a more holistic representation of system behavior.In terms of supervision, Eadro is the only fully supervised baseline method. However, its reliance on labeled data introduces substantial labeling costs, which limits scalability and practical deployment in large production environments. In contrast, unsupervised approaches like MADMM and AnoFusion require no labeled data, facilitating deployment.

Regarding robustness and missing data, multimodal baselines such as AnoFusion and Eadro generally assume complete input, making them sensitive to missing metrics or irregular sampling—conditions frequently encountered in real-world microservice systems. KDOTS and UU-Net incorporate mechanisms for handling incomplete data, but they are tailored for industrial control systems (e.g., liquid rocket engines) and do not account for cross-modality correlations or service dependencies in microservices.

Against this backdrop, ADmM introduces a novel capability as the first approach specifically designed to address metric data incompleteness in microservice systems. By integrating multimodal data and explicitly modeling inter-service dependencies, ADmM maintains high detection accuracy and robustness even under incomplete conditions, thereby bridging a critical gap in current research for microservice environments.

Table 4. Overall Performance with Complete Metrics

Method	Modality			SN			TT			GAIA		
	$\mathcal{M}$	$\mathcal{L}$	$\mathcal{T}$	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
TraceAnomaly			✓	0.6694	0.6606	0.6650	0.6749	0.6805	0.6777	0.7082	0.7370	0.7223
SwissLog		✓		0.6141	0.6785	0.6447	0.5615	0.5829	0.5720	0.7179	0.7906	0.7525
DyCause	✓			0.8725	0.9428	0.9063	0.7723	0.7890	0.7806	0.8493	0.8834	0.8660
DAM	✓	✓		0.8676	0.8978	0.8824	0.8179	0.8305	0.8242	0.8648	0.8634	0.8641
DeepTralog		✓	✓	0.7448	0.7489	0.7468	0.6897	0.6750	0.6823	0.8159	0.7766	0.7958
AnoFusion	✓	✓	✓	<u>0.9848</u>	<u>0.9904</u>	<u>0.9876</u>	**0.9496**	<u>0.9447</u>	<u>0.9471</u>	**0.9632**	<u>0.9704</u>	**0.9668**
Eadro	✓	✓	✓	0.9738	0.9802	0.9770	0.9194	0.9372	0.9282	0.9421	0.9596	0.9571
MADMM	✓	✓	✓	0.9254	0.9347	0.9300	0.9058	0.9145	0.9101	0.9284	0.9315	0.9299
MM-SVM	✓	✓	✓	0.6320	0.6751	0.6528	0.6062	0.5870	0.5964	0.7538	0.7480	0.7509
MM-iForest	✓	✓	✓	0.5782	0.5883	0.5832	0.5663	0.6067	0.5858	0.7128	0.6735	0.6926
KDOTS	✓	✓	✓	0.8166	0.8065	0.8115	0.7967	0.8015	0.7991	0.7805	0.7863	0.7834
UU-Net	✓	✓	✓	0.9228	0.9292	0.9260	0.9038	0.8960	0.8999	0.8886	0.8663	0.8773
ADmM	✓	✓	✓	**0.9935**	**0.9928**	**0.9931**	<u>0.9394</u>	**0.9672**	**0.9531**	<u>0.9465</u>	**0.9795**	<u>0.9627</u>
w/o-scale	✓	✓	✓	0.9654	0.9574	0.9614	0.9052	0.9245	0.9147	0.9177	0.9682	0.9423
w/o-GNN	✓	✓	✓	0.9540	0.9416	0.9478	0.8997	0.9076	0.9036	0.9206	0.9554	0.9377
Improved%	-	-	-	+0.87%	+0.24%	+0.55%	-1.02%	+1.95%	+0.60%	-1.67%	+0.91%	-0.04%

Bold values indicate the best performance, and underlined values indicate the second-best performance.

4.1.4 Experimental Implementation. All experiments were conducted on a host running Ubuntu 20.4.4, equipped with an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20 GHz and 32 GB of RAM. We implemented ADmM using Python 3.7.4, PyTorch 1.5.1, and CUDA 10.2. The model utilizes the Adam optimizer with an initial learning rate of 0.001. The learning rate decays at a rate of 0.9 for each learning cycle and the initial batch size is set to 8. Additionally, the final results were obtained by the outcomes across 10 independent runs, where each run involved a different initialization of the model parameters.

4.2 Overall Performance Comparison (RQ1)

Tables 4 and 5 summarize the performances of various methods with complete data and incomplete data, respectively. The experimental results lead to the following observation and analysis:

As shown in Table 4, ADmM achieves better results across all evaluation metrics on the SN dataset, outperforming current state-of-the-art methods with complete metrics. This result is attributed to its effective feature learning strategies, including both the coarse-grained and fine-grained feature extraction and detection strategies. In contrast, methods like KDOTS and DAM lack specific feature learning strategies for each modality. Besides, approaches that explicitly model **service dependency graphs** (**SDG**) (e.g., DeepTralog, AnoFusion, Eadro, MADMM, and ADmM) generally achieve stronger detection performance. By aggregating information from neighboring nodes, graph models can learn both local and global feature distributions, enhancing their ability to detect anomalies within service interactions. It should be noted that ADmM performs suboptimally on some evaluation metrics on the TT and GAIA datasets. This is primarily because the anomaly detection performance of ADmM is influenced by its imputation model. To address this issue, we balance the imputation and detection tasks by tuning the hyperparameters of the joint loss function, as described in Section 4.5.

Table 5 shows the detection results with 40% incompleteness of metric data. We did not analyze the situation for higher missing rates (incompleteness) because if the missing rate reaches half, we strongly recommend that operators carefully check the monitoring system to avoid greater losses caused by incomplete observability. With 40% incompleteness, the accuracy of all metric-related methods declines. Especially for DyCause, which relies solely on metric data, the Recall decreases

Table 5. Overall Performance with 40% Incompleteness

Method	Modality			SN			TT			GAIA		
	$\mathcal{M}$	$\mathcal{L}$	$\mathcal{T}$	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
TraceAnomaly			✓	0.6694	0.6606	0.6650	0.6749	0.6805	0.6777	0.7082	0.7370	0.7223
SwissLog		✓		0.6141	0.6785	0.6447	0.5615	0.5829	0.5720	0.7179	0.7906	0.7525
DyCause	✓			0.5694	0.6226	0.5948	0.5441	0.5542	0.5491	0.4055	0.4650	0.4332
DAM	✓	✓		0.6606	0.6336	0.6468	0.5056	0.5166	0.5110	0.6303	0.6281	0.6292
DeepTralog		✓	✓	0.7448	0.7489	0.7468	0.6897	0.6750	0.6823	<u>0.8159</u>	<u>0.7766</u>	<u>0.7985</u>
AnoFusion	✓	✓	✓	0.7095	0.7193	0.7144	0.6225	0.6573	0.6394	0.7118	0.7433	0.7272
Eadro	✓	✓	✓	0.7693	<u>0.7606</u>	<u>0.7649</u>	0.6809	0.7022	0.6914	0.7725	0.7579	0.7651
MADMM	✓	✓	✓	0.7328	0.7253	0.7290	0.6613	0.6514	0.6563	0.7254	0.6792	0.7015
MM-SVM	✓	✓	✓	0.4896	0.4625	0.4757	0.4395	0.4285	0.4339	0.6095	0.6842	0.6477
MM-iForest	✓	✓	✓	0.4328	0.4188	0.4257	0.4188	0.4376	0.4280	0.6194	0.6736	0.6454
KDOTS	✓	✓	✓	0.6793	0.6605	0.6698	0.6181	0.6195	0.6188	0.5478	0.5536	0.5507
UU-Net	✓	✓	✓	<u>0.7730</u>	0.7515	0.7621	<u>0.7135</u>	<u>0.7322</u>	<u>0.7227</u>	0.7375	0.7478	0.7426
ADmM	✓	✓	✓	**0.8095**	**0.8303**	**0.8198**	**0.7531**	**0.8036**	**0.7775**	**0.8243**	**0.8159**	**0.8201**
w/o-scale	✓	✓	✓	0.7268	0.7287	0.7277	0.7225	0.7573	0.7395	0.8118	0.7833	0.7973
w/o-GNN	✓	✓	✓	0.7584	0.7457	0.7520	0.7163	0.7223	0.7193	0.8075	0.7878	0.7975
Improved%	-	-	-	+3.65%	+7.88%	+5.77%	+3.96%	+7.14%	+5.48%	+0.84%	+3.93%	+2.16%

Bold values indicate the best performance, and underlined values indicate the second-best performance.

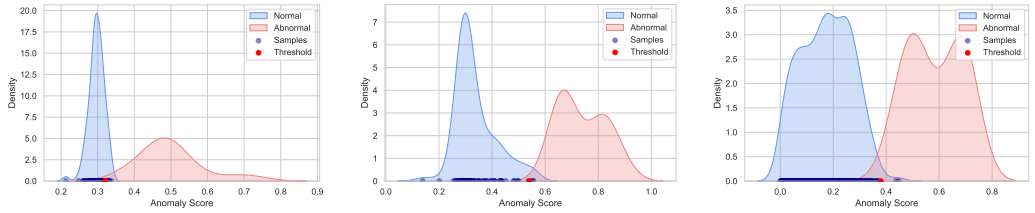
by 31.02% and 41.84% on the SN and GAIA datasets, respectively, showing the most significant drop. Other metric-related methods also experience notable performance degradation. For instance, the F1 scores of DAM and AnoFusion on the TT dataset decrease by 31.32% and 28.75%, respectively. However, methods incorporating metric imputation, such as UU-Net and ADmM, demonstrate greater robustness under high missing rates, with the F1-score reductions of only 9.69% and 16.59% on the TT dataset. In addition, methods such as TraceAnomaly, SwissLog, and Deeplog are not affected in scenarios with missing metric data.

It is worth noting that supervised approaches like Eadro achieve excellent performance when all data sources are complete, reaching an average F1-Score of 95.41% across the three datasets. However, their performance is heavily dependent on complete collection of traces, logs, and KPIs. Under 40% missing metric data, Eadro's average F1-Score drops to 74.2%, whereas ADmM maintains a higher average F1-Score of 80.85%, highlighting its robustness under incomplete observability.

To better demonstrate the performance of ADmM, we visualize the distribution of AS, as shown in Figures 7 and 8. Specifically, in the scenario with complete metric data, the AS generated by ADmM can effectively distinguish between normal and abnormal data, showing good performances across all datasets. In the scenario with 40% incompleteness, we use ADmM's imputation module to fill in the missing values. However, compared to the score distributions with complete metrics, there is still a noticeable shift in the scoring distribution. This is due to the fact that some of the missing data points contains actually anomalies, which results in a blurred distinction between normal and abnormal data.

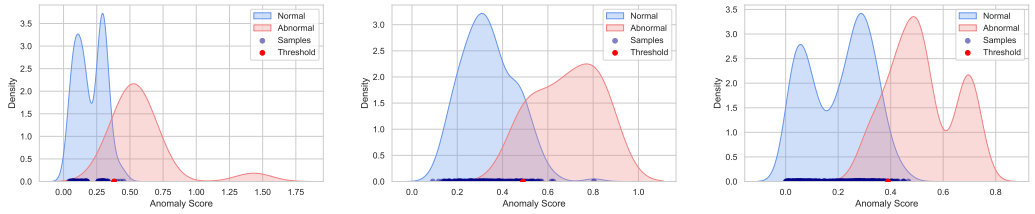
4.3 Ablation Study (RQ2)

In this section, we first assess the contribution of imputation and reconstruction modules by replacing the multi-scale Transformer and GNN with normal MLPs, as shown in Tables 4 and 5, ablation experiments demonstrate that both the imputation and reconstruction modules play a significant role in improving anomaly detection performances. The multi-scale CNN and Transformer capture sequential and contextual patterns across various time scales, enabling the model to extract fine-grained and coarse-grained temporal dependencies from the data. Meanwhile, the graph network provides a comprehensive and structured perspective for modeling the state of microservice systems.



(a) AS distribution on the SN dataset. (b) AS distribution on the TT dataset. (c) AS distribution on the GAIA dataset.

Fig. 7. The distribution of AS with complete metrics.



(a) AS distribution on the SN dataset. (b) AS distribution on the TT dataset. (c) AS distribution on the GAIA dataset.

Fig. 8. The distribution of AS with 40% incompleteness.

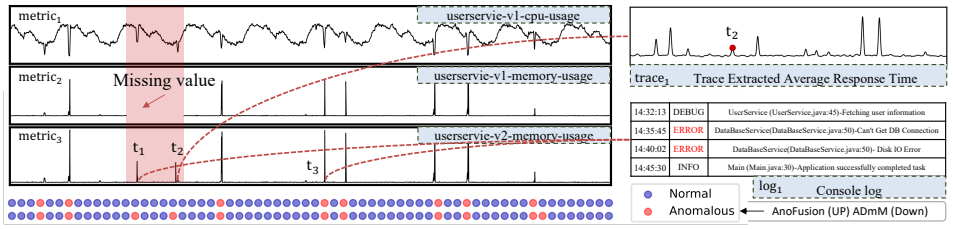
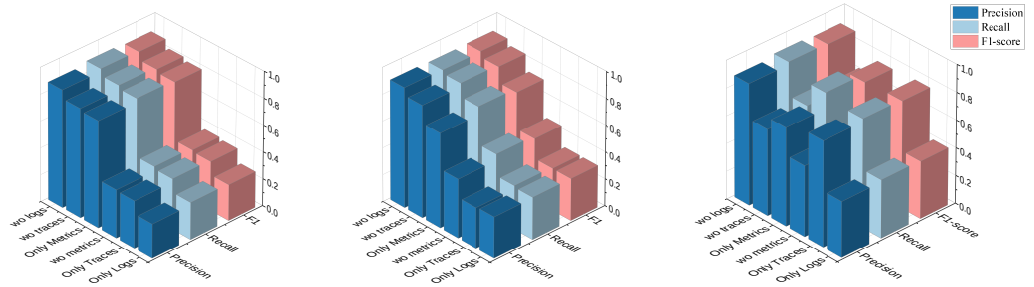


Fig. 9. Contribution among multi-modalities in metric imputation.

Next, we evaluate the contribution of different modalities for imputation. Figure 9 illustrates the role of these modalities in imputing missing metrics. The red bar in the figure indicates the period when $metric_2$ is incomplete, where system anomalies are detected at t_1 and t_2. Specifically, anomalies are observed in $metric_1$, $metric_3$, and log_1 at t_1, whereas anoamlies are found in $metric_1$, $metric_3$, and $trace_1$ at t_2. By utilizing similar anomalous patterns from historical data, the multi-scale CNN and Transformer impute missing metrics with other modalities, such as logs and traces. This capability significantly alleviates the challenge of anomaly detection in scenarios where metric data is incomplete. The scatter plots in Figure 9 compare the anomaly detection results of AnoFusion (Up) and ADmM (Bottom), where red points represent anomalous samples, while blue points indicate normal samples. A clear improvement is observed in the bottom row, which shows superior performance of ADmM.

In addition, we analyzed the contributions of different modalities to anomaly detection. As illustrated in Figure 10, log data contributes the least, and its removal has a minimal impact on three datasets. This is because logs primarily provide coarse-grained, node-level information. In contrast, the trace and metric data play a more significant role due to their comprehensive content.



(a) Modality contribution on the SN dataset (b) Modality contribution on the TT dataset (c) Modality contribution on the GAIA dataset

Fig. 10. Contribution of multi-modalities for anomaly detection.

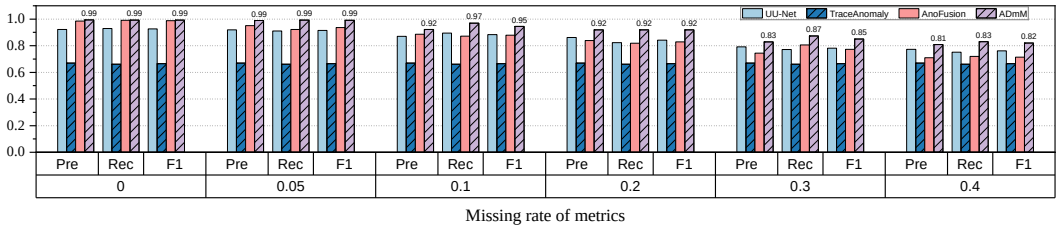
As trace data encodes edge-level features derived from the system's dependency graph, capturing inter-service dependencies, its absence may lead to incomplete representations of system-wide interactions and a decline in detection performance. The metric data, however, has the greatest impact. Its removal drastically reduces the system's anomaly detection capability, as metrics provide precise temporal and quantitative information about individual service nodes, which is critical for assessing node health and identifying anomalies.

It is also noticed in Figure 10, the results of ablation experiments differ significantly across datasets. Compared to the TT and GAIA datasets, the performance degradation on the SN dataset is more pronounced. This difference can be attributed to the smaller size of the SN dataset, where the multimodal data fusion essentially serves as a form of data augmentation. In cases with limited samples, integrating different modalities increases the information density of training samples, helping to mitigate the challenges posed by data scarcity. This further demonstrates that the multimodal fusion strategy provides a more comprehensive perspective on system faults, which is crucial to improving the accuracy and robustness of anomaly detection. Additionally, since most of the anomalies in the GAIA dataset manifest in both metric and trace data, its performance in the ablation experiments differs slightly from the other two datasets.

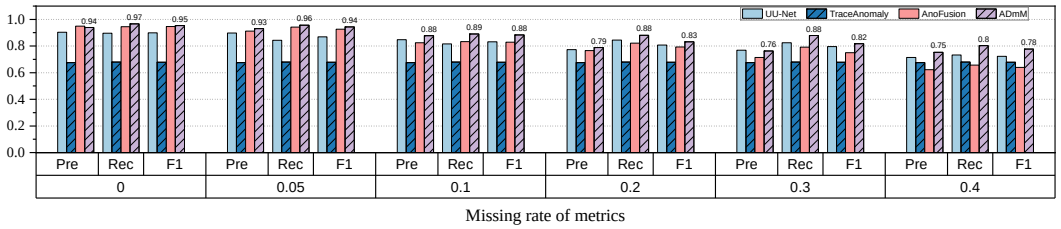
4.4 Performances for Diverse Missing Rates (RQ3)

In real-world scenarios, the metric data often suffers from varying degrees of incompleteness. Figure 11 illustrates the performance of all methods across different missing rates, ranging from 0.05 to 0.4. Among the comparison methods, we selected three representative comparison methods: TraceAnomaly, AnoFusion, and UU-Net, which represent the unimodal-based method, multimodal-based method, and incompleteness-based method, respectively. These methods perform relatively well in each category according to Table 4.

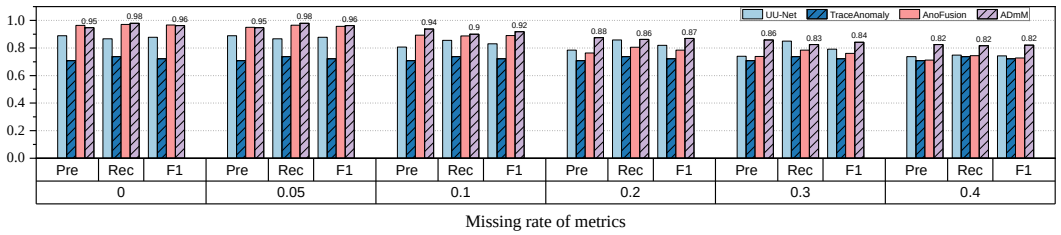
Figure 11 reveals the overall trend as the missing rate increases. When the missing rate is 0.05, all methods show only a slight decline in performance. However, once the missing rate exceeds 0.1, conventional methods, which heavily depend on complete data for anomaly detection, begin to suffer from suboptimal performance. As the missing rate further rises to 0.4, the detection performances of TraceAnomaly and AnoFusion drop significantly, exposing the limitations of conventional detection methods in handling incompleteness. In contrast, methods containing imputation demonstrate superior performance under such conditions. For example, a comparison between UU-Net and AnoFusion shows that with complete metrics, UU-Net performs slightly worse than AnoFusion. However, when the missing rate reaches 40%, UU-Net outperforms AnoFusion



(a) Effect of different missing rates on the SN dataset.



(b) Effect of different missing rates on the TT dataset.



(c) Effect of different missing rates on the GAIA dataset.

Fig. 11. Detection results on diverse missing rates.

across all three datasets. Additionally, ADmM consistently outperforms all other methods with incomplete metrics.

To comprehensively demonstrate the performance of ADmM under different missing rates, Table 6 provides the mean and variance of each method. ADmM achieves the best average performance across all three datasets. In particular, the F1 scores on the three datasets improved by 4.93%, 5.58%, and 1.15%, respectively, compared to the second-best method. By effectively leveraging available data to approximate missing portions, ADmM demonstrates strong robustness and practical effectiveness.

To evaluate the quality of metric reconstruction, we conducted experiments on three benchmark datasets with varying missing ratios (0.1–0.4). Reconstruction performance was measured using MSE, with results presented in Table 7. As the missing ratio increases, MSE also rises, reflecting the growing difficulty of accurately imputing missing values. However, ADmM consistently outperforms mean-fill across the 0.1–0.4 missing ratio range, demonstrating the effectiveness of its imputation. At higher missing ratios, reconstruction uncertainty increases significantly, which may lead to a decline in anomaly detection performance. We recommend that when the proportion of missing data is extremely high, practitioners first examine the monitoring infrastructure itself (e.g., monitoring agent failures or network instability) rather than relying solely on anomaly detection algorithms.

Table 6. Average Performance Under Different Missing Rates

Dataset	Method	Precision		Recall		F1-Score	
		Mean	Variance	Mean	Variance	Mean	Variance
SN	UU-Net	0.8567	0.0033	0.8468	0.0048	0.8516	0.0039
	TraceAnomaly	0.6694	0.0000	0.6606	0.0000	0.6650	0.0000
	AnoFusion	0.8525	**0.0101**	0.8547	**0.0076**	0.8533	**0.0086**
	ADmM	**0.9105**	0.0050	**0.9295**	0.0038	**0.9198**	0.0043
	Improvement	+5.38%	-	+7.48%	-	+6.65%	-
TT	UU-Net	0.8172	0.0050	0.8262	0.0024	0.8210	0.0032
	TraceAnomaly	0.6749	0.0000	0.6805	0.0000	0.6777	0.0000
	AnoFusion	0.7982	**0.0126**	0.8318	**0.0095**	0.8143	**0.0109**
	ADmM	**0.8419**	0.0059	**0.8964**	0.0030	**0.8678**	0.0042
	Improvement	+2.47%	-	+6.46%	-	+4.68%	-
GAIA	UU-Net	0.8074	0.0039	0.8406	0.0018	0.8228	0.0022
	TraceAnomaly	0.7082	0.0000	0.7370	0.0000	0.7223	0.0000
	AnoFusion	0.8366	**0.0104**	0.8825	**0.0091**	0.8574	**0.0087**
	ADmM	**0.8981**	0.0023	**0.8937**	0.0044	**0.8956**	0.0031
	Improvement	+6.15%	-	+1.12%	-	+3.82%	-

Bold values indicate the best performance, and underlined values indicate the second-best performance.

Table 7. MSE Under Different Missing Rates

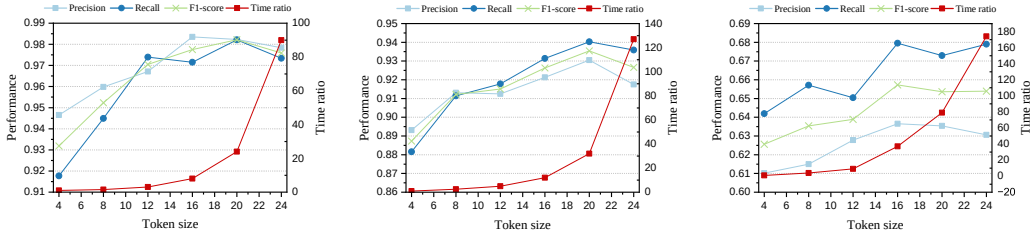
Dataset	Social Network				Train Ticket				GAIA			
Missing rate	0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4
MSE (ADmM-imputation)	0.0123	0.0185	0.0221	0.0304	0.0102	0.0157	0.0210	0.0283	0.0051	0.0105	0.0182	0.0250
F1-Score (ADmM-imputation)	0.9881	0.9436	0.8573	0.8198	0.9434	0.9027	0.8152	0.7775	0.9730	0.9417	0.8564	0.8201
F1-Score (ADmM-mean-fill)	0.9642	0.9123	0.7721	0.7214	0.9205	0.8754	0.7503	0.6511	0.9402	0.8901	0.8078	0.7205

4.5 Parameter Analysis (RQ4)

The performance of ADmM is affected by the token size and hyperparameters, which are discussed in this section.

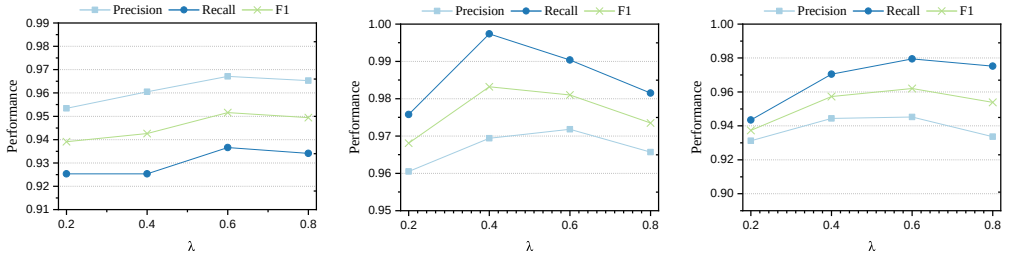
In our setting, the time window is determined by the token size and sampling interval. The token size specifies the number of data points in each input sequence, and multiplying it by the sampling interval defines the temporal coverage. Larger tokens correspond to longer windows, while the sampling interval controls the temporal granularity of the observations. As shown in Figure 12, increasing the token size enhances anomaly detection by enabling the model to capture more complex temporal dependencies among features. However, a larger token size increases model complexity by expanding the input dimension, which raises the risk of overfitting, especially when the dataset is insufficient to support high-dimensional representations. In such cases, the model may overfit to noise in the training data, reducing its generalization capability. With a default sampling interval of 5 seconds, the optimal token size is 16 for the SN and GAIA datasets, corresponding to a time window of 80 seconds. For the TT dataset, the optimal token size is 20, yielding a time window of 100 seconds.

In addition, λ determines the relative importance of the imputation loss ($\mathcal{L}_1$) and the reconstruction loss $\mathcal{L}_2$ in the final loss function. When λ approaches 1, $\mathcal{L}_1$ dominates, driving the model to prioritize minimizing the reconstruction error of the imputation. This forces the model to focus on handling incomplete data. Conversely, when λ is close to 0, $\mathcal{L}_2$ takes precedence, guiding the



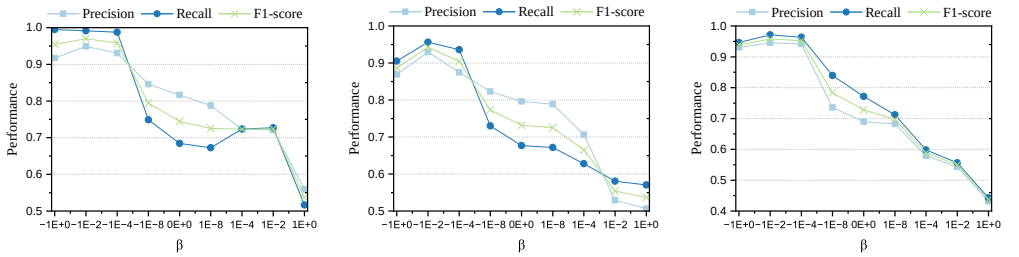
(a) Performance on the SN dataset. (b) Performance on the TT dataset. (c) Performance on the GAIA dataset.

Fig. 12. Performance impacts of the token size.



(a) Performance on the SN dataset. (b) Performance on the TT dataset. (c) Performance on the GAIA dataset.

Fig. 13. Performance impacts of λ.



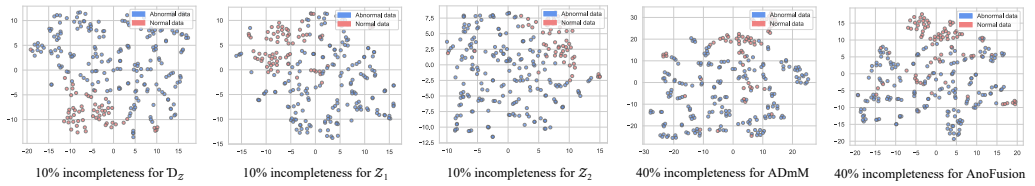
(a) Performance on the SN dataset. (b) Performance on the TT dataset. (c) Performance on the GAIA dataset.

Fig. 14. Performance impacts of β.

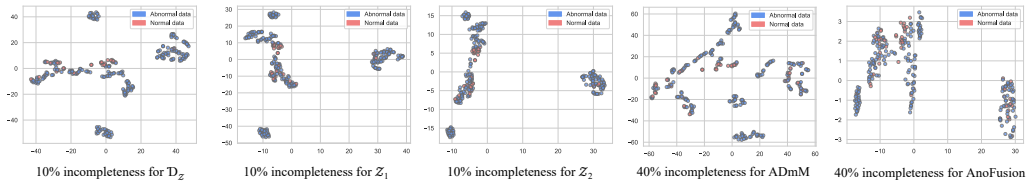
model to prioritize reconstructing observed data rather than imputing incomplete values. Figure 13 illustrates how varying λ impacts the anomaly detection performance. On the TT dataset, the optimal value of λ is 0.6, whereas on the SN and GAIA datasets, the best results are achieved with $\lambda = 0.4$.

Another crucial hyperparameter in the loss function is β, which controls the distance between latent vectors $\mathcal{Z}_1$ and $\mathcal{Z}_2$. A well-calibrated β prevents the imputation and reconstruction models from collapsing into similar feature representations, ensuring that anomalies remain distinguishable. As shown in Figure 14, when $\beta = -1$, the loss function overly prioritizes $\mathcal{D}_Z$, disrupting the anomaly detection objective and leading to degraded performance. Conversely, when β is set to -0.01, ADmM achieves the best performance across three datasets.

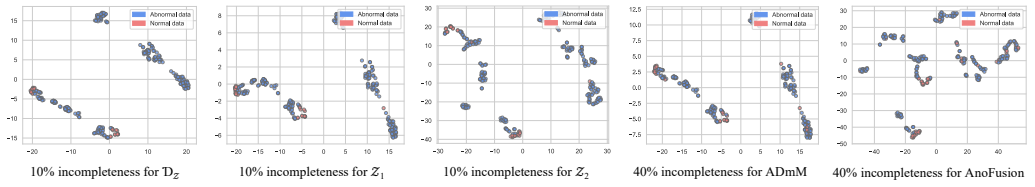
Further analysis reveals that when $\beta > 0$, enforcing similarity between $\mathcal{Z}_1$ and $\mathcal{Z}_2$ significantly weakens anomaly detection, as the model struggles to differentiate normal and anomalous patterns.



(a) t-SNE results on the SN dataset.



(b) t-SNE results on the TT dataset.



(c) t-SNE results on the GAIA dataset.

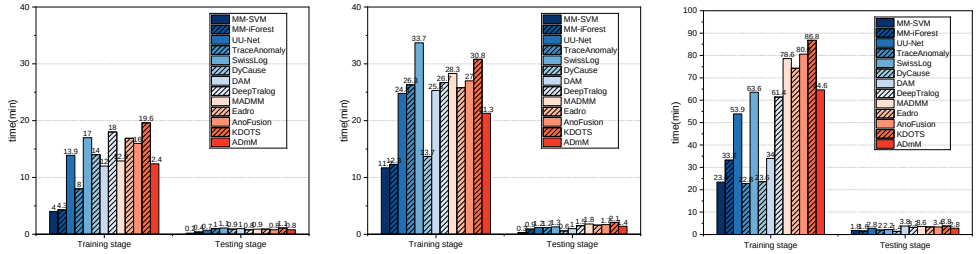
Fig. 15. t-SNE results under 10% incompleteness and 40% incompleteness.

Even when $\beta = 0$, where $\mathcal{D}_Z$ has no direct effect on the loss function, detection performance remains suboptimal. This highlights the necessity of incorporating $\mathcal{D}_Z$ to enhance feature discrimination.

To better illustrate the impact of $\mathcal{D}_Z$, we visualize the latent space using t-SNE [23]. Figure 15 demonstrates that under a 10% missing rate, incorporating $\mathcal{D}_Z$ enables the model to more effectively separate normal and anomalous samples compared to using only $\mathcal{Z}_1$ and $\mathcal{Z}_2$. However, when the missing rate increases to 40%, the latent space becomes less distinguishable. Notably, ADmM maintains a stronger separation than AnoFusion, highlighting its robustness under high missing rates. These results confirm that maximizing $\mathcal{D}_Z$ enhances anomaly detection by encouraging distinct latent representations.

4.6 Efficiency Analysis (RQ5)

The anomaly detection process consists of two stages: training and testing. In the training stage, high-dimensional data increases the complexity of the feature space, leading to higher computational costs and longer training times. During testing, the increased dimensionality also raises the computational load, resulting in slower inference times. As detailed in Table 2, while the SN and TT datasets have the same data dimensions, they differ in data volume. In contrast, the GAIA dataset has both larger data volume and higher dimensionality, making feature extraction and model initialization more complex. This increase leads to longer training and testing times, as shown in Figure 16. Furthermore, the performance of different methods varies across datasets. Specifically, methods involving log processing, such as Swisslog, DeepTralog, AnoFusion, and ADmM, consume more time. Additionally, since KDOTS involves training the teacher and student



(a) Time cost on the SN dataset. (b) Time cost on the TT dataset. (c) Time cost on the GAIA dataset.

Fig. 16. Training time and test time on three datasets.

models, it requires extra time compared to directly training a single model to generate high-quality outputs.

4.7 Threats to Validity

To assess the credibility and applicability of the research, we discuss the internal, external, and construct validities [44], as they represent different aspects of potential threats.

Internal validity primarily concerns whether the detected anomalies are caused by system failures rather than external confounding factors. A key threat to internal validity is model overfitting: Even with well-prepared datasets, overly complex reconstruction models may overfit the training data, failing to capture meaningful anomaly patterns. This results in high performance on seen data but poor accuracy on novel anomalies. To counteract this, we incorporate strategies such as graph regularization and early stopping during training, which help improve generalization and enhance the model's reliability.

External validity assesses whether the research findings can be generalized across diverse environments, ensuring the model's reliability and adaptability in different microservice systems. A well-generalized model must effectively handle varying system conditions and temporal changes. A key threat to external validity is dataset representativeness: A model trained on datasets with limited diversity may struggle to detect anomalies in unseen environments, limiting its applicability in real-world scenarios. To address this, we evaluate the model on three distinct datasets: two standard microservice benchmarks and a dataset containing multiple fault scenarios.

Construct validity focuses on whether the experimental design accurately measures the research objective, ensuring that the variables and methods reflect the research problem. In this study, the first threat to construct validity is the choice of evaluation metrics. We evaluate using multiple metrics, including Precision, Recall, and F1-score, to provide a comprehensive assessment of anomaly detection performance and reduce bias in evaluation. To ensure a realistic evaluation and avoid temporal leakage, each dataset was partitioned chronologically into training and test sets using a 20–80 split. This preserves the temporal order, ensuring that the model is evaluated on data occurring after the training period.

4.8 Limitations

Although ADmM demonstrates strong performance, it is affected by several factors:

(*i*) *Dependence on data correlations*: The performance of ADmM relies on the correlation between metrics, logs, and traces in microservice systems. However, in practical scenarios, such correlations may not always be stable. If the relationships among these modalities are weak or inconsistent, the model may struggle to learn effective feature representations, leading to degraded performance.

(*ii*) *Simplified experimental assumptions*: The missing metrics in our experiments were artificially generated by masking specific metric values, and the test data were assumed to follow the same distribution as the training data. While these settings simplify evaluation, they do not fully capture the complexity of production environments. In real-world systems, missing data patterns can be highly irregular due to network instability, container restarts, and adaptive sampling policies. Moreover, microservice environments are inherently dynamic, where new services, workloads, and failure modes may emerge over time, leading to patterns unseen during training.

In addition, ADmM also exhibits the following limitations:

(*i*) *Applicability under partial modality availability*: ADmM is designed to leverage multimodal observability data, yet real-world microservice systems often operate under partial observability due to cost, privacy, or infrastructure constraints. Based on the modality ablation results in Section 4.3, ADmM remains applicable when only a subset of modalities is available, with predictable degradation in detection performance. In particular, metrics alone provide a strong baseline signal, as they provide fine-grained temporal and quantitative indicators essential for assessing service-level health. The joint availability of metrics and traces further enhances performance by incorporating inter-service dependency information, while the absence of log data has a relatively limited impact. Although ADmM has not been explicitly optimized for scenarios where multiple modalities are simultaneously incomplete or intermittently unavailable, these observations offer practical guidance for observability prioritization in resource-constrained environments. Specifically, metrics emerge as the most essential modality for maintaining a reasonable detection baseline, traces provide a high-value complementary signal by capturing service dependencies, and logs can be treated as optional when collection or retention overhead is prohibitive.

(*ii*) *Sensitivity to incomplete and informative missing data*: Although ADmM can tolerate moderate levels of missing metrics, extremely high missing rates may still significantly degrade detection accuracy. In such cases, we recommend that operators carefully inspect the observability pipeline to mitigate further performance loss. More importantly, in real-world microservice systems, missing data may not only affect model performance but may itself be indicative of abnormal events. For example, service crashes, process hangs, or abrupt failures may simultaneously lead to system anomalies and the disappearance of abnormal metrics, traces, or logs. In the current framework, missing data are primarily treated as a source of incompleteness rather than as an explicit anomaly signal. As a result, ADmM has not been evaluated in scenarios where missingness itself carries semantic meaning, and may fail to fully capture anomaly patterns associated with such informative missing data. Explicitly modeling anomaly-aware missingness remains an important direction for future work.

5 Related Works

Based on the observability and capability to deal with missing data, the anomaly detection methods can be classified into the following categories:

5.1 Log-based Methods

Logs serve as a critical source of operational insights, which capture system events in a semi-structured textual format. When anomalies occur, sequential patterns in logs often deviate from typical structures, enabling anomaly detection by identifying these deviations from normal behavior. Typically, log-based methods consist of three main stages: log parsing, feature extraction, and anomaly detection [10, 17, 54].

DeepLog [10] leverages LSTM networks to model sequential patterns in system logs and detect anomalies by identifying deviations from learned normal patterns. Similarly, Yuan et al. [51] proposed an adaptive framework that builds LSTM models in real-time using an online

deep-learning approach for unsupervised anomaly detection. However, these methods often fail to capture critical component-level information in log messages, such as the specific sources of logs and the interactions between different system modules, which may provide important context for anomaly detection. To address this limitation, LogC [50] improves anomaly detection by transforming unstructured logs into structured templates and component sequences. LSTM models then process these structured logs for anomaly detection. Despite this advancement, the instability of log formats and the presence of noise in logs can undermine the model's performance. To enhance robustness in the face of these issues, LogRobust [54] takes a different approach by converting log events into semantic vectors. It then uses a **Bidirectional LSTM** (**Bi-LSTM**) combined with attention mechanisms, which help the model focus on the most relevant parts of the logs for anomaly detection. Traditional log-based anomaly detection methods typically struggle with both sequential anomalies (where the order of events is important) and quantitative anomalies (which involve deviations in numeric values). These methods often rely on template indexing, which can result in false positives. To address this limitation, Template2vec [31] integrates semantic features with LSTM models to improve detection accuracy and reduce false positives. SwissLog [26] takes a different route by building an ID relationship graph to link distributed logs and applying a dictionary-based parsing method to extract both semantic and temporal features from the logs. In scenarios where API logs are sparse, PLELog [48] introduces probabilistic label estimation, which enhances the training process by making better use of sparse labels. Unlike traditional methods, NeuralLog [22] bypasses the traditional log parsing step entirely. Instead, it directly processes raw logs using BERT-based models for semantic representation, significantly improving the model's ability to interpret the raw log data without requiring extensive preprocessing.

These methods rely on the sequential representation of logs, depending solely on the single modality, which limits their ability to detect complex anomalies resulting from interactions effectively.

5.2 Trace-based Methods

Traces consist of multiple spans, each representing a specific operation or step within a system. These spans capture critical details, such as the start and end times of an operation and their relationships to other spans.

Nedelkoski et al. [33] proposed an anomaly detection method based on trace data that leverages deep learning models. It learns the distribution of normal system behavior through the GRU model and sets a deviation threshold to flag anomalies using the VAE model. They further enhanced this method by integrating additional information, such as textual labels and response times extracted from traces [34]. TraceGra [6] takes a different approach by transforming trace data into graph structures, which are then analyzed using the GNN for topological insights. Additionally, the LSTM networks capture temporal patterns in the trace data. Similarly, TraceVAE [45] introduces a bivariate Graph Variational Autoencoder to jointly encode both structural and temporal features of microservice traces, efficiently identifying anomalous paths. Based on the above ideas, TraceAnomaly [29] introduces **Service Trace Vectors** (**STV**), which encapsulate call paths and response times into feature vectors. A deep Bayesian network then learns the normal patterns of these vectors and computes AS to detect deviations. TraceModel [5] uses a VAE-based approach to analyze request call chains via a SDG, capturing the relationships within the graph to detect anomalies.

While trace-based techniques are essential for real-time observability and proactive incident management, they primarily capture successfully routed requests. As a result, failures caused by unrouted or dropped requests may be overlooked. This limitation highlights the need for multimodal approaches that integrate metrics, logs, and traces to ensure comprehensive system reliability.

5.3 Metric-based Methods

Metrics provide essential insights into system health and operational efficiency, which can be broadly categorized into service-level metrics (e.g., throughput, request rate, and error rate) and system-level metrics (e.g., CPU utilization, memory availability, and disk I/O). In general, metric-based methods employ statistical methods, machine learning algorithms, or deep learning techniques to analyze time-series data. These methods then identify deviations from expected patterns to detect anomalies effectively.

DLA [40] detects anomalies by analyzing correlations between system metrics and dynamically estimating thresholds for individual metrics. However, it relies on user transaction counts, which limits its applicability. PDiagnose [19] integrates rule-based methods with p-value-based AS thresholds, providing a more statistically principled alternative to heuristic thresholding. Instead of manually defining static thresholds, it assesses anomalies based on the statistical significance of deviations. Unlike these threshold-based methods, Gulenko et al. [16] propose a cross-layer monitoring framework that applies Birch clustering to detect anomalies. By clustering similar system states and identifying outliers, this approach eliminates the need for manually predefined thresholds. However, such clustering methods often assume specific data distributions, which may not hold in the dynamic and heterogeneous environments of real-world systems.

Beyond statistical and rule-based models, generative approaches have been explored for capturing complex system behaviors. OmniAnomaly [41] combines VAE and GRU to learn temporal dependencies in system metrics. By modeling normal behavior patterns, it detects anomalies based on deviations in reconstruction probabilities. Similarly, Donut [46] improves anomaly detection by excluding anomalous samples during training, ensuring that the model focuses solely on learning normal system states. Anomalies are then flagged based on reconstruction errors. Bagel [27] further refines this approach by incorporating **Conditional Variational Autoencoders** (**CVAE**), where temporal dependencies serve as conditional inputs, enhancing the model's ability to capture contextual patterns. Meanwhile, TopoMAD [18] leverages both system topology and temporal information by combining Graph GNNs, LSTMs, and VAEs to model intricate spatiotemporal dependencies.

While metrics offer valuable numerical insights into system performance, they often lack the contextual depth required for comprehensive analysis. It is essential to integrate other modalities to improve the accuracy.

5.4 Multimodal-based Methods

Recent advances in anomaly detection leverage multimodal data integration to improve accuracy and robustness. Specifically, combining metrics, logs, and traces provides a more comprehensive view of system behaviors and reduces false positives in highly distributed environments.

Some approaches integrate two modalities. For instance, Some methods integrate logs and traces to enhance anomaly detection. DeepTralog [52] constructs a **Trace Event Graph** (**TEG**) by embedding log data into traces and employs a deep learning model (GGNN combined with SVDD) to learn the latent representations. This model defines a hyperspherical boundary to distinguish normal behaviors from anomalies. However, DeepTralog primarily focuses on service interactions and does not effectively detect anomalies at the instance level. To address this limitation, ServiceAnomaly [37] introduces a **Context Propagation Graph** (**CPG**) that models relationships among six key performance metrics: service execution time, request size, response size, service queue time, delay, and throughput. Comparing this graph with a reference model built under normal conditions identifies deviations that indicate anomalies. Other approaches combine logs and metrics to extract richer contextual information. For example, SCWarn [56] and DAM [7] parse logs into numerical data using Drain template matching and normalize them alongside metrics before feeding into a multimodal LSTM. This workflow enables the model to learn interactions

between logs and metrics. However, the template-based processing may lose critical information embedded in raw logs, limiting the model's ability to capture fine-grained anomalies. Hades [25] applies FastText and Transformer models to extract semantics and sequential dependencies in logs to overcome this limitation. Then, it uses a hierarchical encoder with causal convolution networks to model metric representations. However, like SCWarn and DAM, Hades does not explicitly model service dependencies, which are crucial for understanding anomaly propagation in distributed systems.

Methods that leverage all three modalities—metrics, traces, and logs—provide a more comprehensive view of system health but also impose higher demands in feature extraction. AnoFusion [55] serializes these modalities to construct a heterogeneous graph structure. It refines the graph using a **Graph Transformer Network** (**GTN**) followed by a Graph Attention NetworkGAT to highlight key features. Finally, it computes AS based on the deviation between predicted and observed values. Triple [39] also builds a heterogeneous graph but starts with trace data as the foundation. It applies the Drain method for log template matching and incorporates the average response time of each service instance into the trace node features. This integration improves the representation of system performance. Triple then uses a DeepSVDD model based on **Spatiotemporal Graph Convolutional Networks** (**STGCN**) to capture spatiotemporal dependencies more effectively. MSTGAD [20] further advances multimodal integration by replacing traditional GNNs with a Transformer-based architecture. By leveraging attention mechanisms, it extracts important features and significantly improves time-series modeling performance. Eadro [24] stematically unifies multi-source data (traces, logs, and KPIs) for integrated modeling, providing a multi-dimensional representation of intra-service behaviors and inter-service dependencies. By employing a multi-task learning paradigm, it enables end-to-end joint training and inference for anomaly detection and root cause localization. While Eadro effectively enhances troubleshooting accuracy through information complementarity and task synergy, it also significantly increases model complexity and computational overhead.

5.5 Anomaly Detection in the Face of Incomplete Data

In real-world microservice environments, data collection tools often encounter resource limitations or environmental fluctuations, leading to incomplete metrics data. This data loss can obscure critical system behaviors, reducing anomaly detection accuracy. Anomaly detection under missing data scenarios has received increasing attention in recent years [15, 30, 47, 47, 53]. Early studies primarily concentrated on modeling the missing patterns of data. For example, Ghahramani et al. [15] proposed a method based on **Expectation-Maximization** (**EM**) and probabilistic modeling, providing a theoretical framework for anomaly detection with missing data. Later, research shifted towards deep learning-based approaches. Yang et al. [47] introduced missing rate modeling and data reconstruction techniques to improve anomaly detection performance when data is incomplete. Zhou et al. [49] proposed TAME, which uses graph structure modeling and representation learning to enhance anomaly detection under missing data. Zhang et al. [53] introduced KDOTS, which applies dynamic kernel modeling in temporal scenarios to improve robustness against irregular sampling and missing values. Meanwhile, Feng et al. [13] addressed the more general problem of coexisting missing data and noise. They proposed a deep generative model that jointly performs data imputation and anomaly detection in an unsupervised manner. However, most existing methods rely on the assumption of single-source temporal data. They have not yet modeled the multi-source heterogeneity and complex dependencies of microservice systems, leaving a gap in systematic solutions for real-world operational environments.

These approaches have not been optimized for microservice systems, particularly in terms of modeling the correlations among the three types of observability data and the dependency

relationships between microservices. While prior research on microservice anomaly detection often considers such feature modeling, it has not addressed scenarios with missing metrics. ADmM fills this gap by providing a dedicated strategy for both imputing missing metrics and performing anomaly detection specifically in microservice systems.

6 Conclusion and Discussion

To tackle the challenge of incomplete metrics in microservice systems, we propose ADmM, a multimodal model for anomaly detection. The model first extracts both template-level and semantic-level features from multimodal inputs. Next, we apply a multi-scale-based AE module to impute incomplete metrics. For anomaly detection, ADmM maps the trace data into a graph structure and employs a GNN to learn generative patterns of normal samples. By comparing deviations between the observed and estimated values, the model assigns AS to detect anomalies. Extensive experiments on three real-world multimodal datasets demonstrate that ADmM outperforms the state-of-the-art methods.

Future work will focus on several directions to further enhance the robustness and applicability of ADmM.

First, we plan to strengthen the robustness of ADmM under incomplete and noisy multimodal observations. Although ADmM effectively handles partially missing metrics, real-world microservice systems often exhibit incomplete logs and traces, along with noisy signals introduced by transient workload spikes, measurement errors, or unstable monitoring agents. In the current study, such transient noise is not explicitly labeled or separated from anomalies in the evaluated datasets and is implicitly treated as part of normal system behavior during training. Addressing these issues therefore requires not only accurate within-modality imputation but also cross-modal reasoning that exploits temporal alignments and causal dependencies across services. To further improve robustness against noise, we plan to explore noise-aware representation learning and robust optimization methods that introduce both structured and stochastic noise during training, combined with consistency regularization to promote stability in learned representations under corrupted or fluctuating inputs.

Second, current experiments assume a static SDG. Adapting to dynamic topologies presents a significant challenge due to changing node and edge configurations. We plan to explore incremental graph learning and continual adaptation methods, enabling ADmM to maintain accurate anomaly detection as microservice instances and dependencies evolve. This includes offline periodic retraining, which updates the model with newly observed data, and online incremental learning, which continuously adapts to evolving services and workloads.

Finally, additional topological relationships beyond service dependencies exist, such as co-location on the same physical server or shared network segments. Future work could explore constructing hypergraphs to capture these richer relationships, providing a more comprehensive representation of microservice system architecture.

References

[1] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A software architect's perspective.* Addison-Wesley Professional.
[2] Carlo Batini and Monica Scannapieco. 2006. *Data Quality: Concepts, Methodologies and Techniques.* Springer.
[3] Kristin Bennett and Ayhan Demiriz. 1998. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems*, M. Kearns, S. Solla, and D. Cohn (Eds.), Vol. 11. MIT Press.
[4] Anastasia Borovykh, Sander Bohte, and Cornelis W. Oosterlee. 2018. Conditional time series forecasting with convolutional neural networks. arXiv:1703.04691. Retrieved from https://arxiv.org/abs/1703.04691
[5] Yang Cai, Biao Han, Jinshu Su, and Xiaoyan Wang. 2021. TraceModel: An automatic anomaly detection and root cause localization framework for microservice systems. In *Proceedings of the 2021 17th International Conference on Mobility, Sensing and Networking (MSN).* 512–519.

[6] Jian Chen, Fagui Liu, Jun Jiang, Guoxiang Zhong, Dishi Xu, Zhuanglun Tan, and Shangsong Shi. 2023. TraceGra: A trace-based anomaly detection for microservice using graph deep learning. *Computer Communications* 204, C (apr 2023), 109–117.

[7] Yufu Chen, Meng Yan, Dan Yang, Xiaohong Zhang, and Ziliang Wang. 2022. Deep attentive anomaly detection for microservice systems with multimodal time-series data. In *Proceedings of the 2022 IEEE International Conference on Web Services (ICWS)*. 373–378.

[8] Marcello Cinque, Raffaele Della Corte, and Antonio Pecchia. 2022. Micro2vec: Anomaly detection in microservices systems by mining numeric representations of computer logs. *Journal of Network and Computer Applications* 208, C (dec 2022), 15 pages.

[9] Zhicheng Cui, Wenlin Chen, and Yixin Chen. 2016. Multi-scale convolutional neural networks for time series classification. arXiv:1603.06995. Retrieved from https://arxiv.org/abs/1603.06995 (2016).

[10] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS'17)*. Association for Computing Machinery, 1285–1298.

[11] Bradley Efron. 1994. Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association* 89, 426 (1994), 463–475.

[12] Facebook AI Research. 2024. FastText: Library for Efficient Text Classification and Representation Learning. Retrieved October 28, 2024 from https://fasttext.cc

[13] Yong Feng, Zijun Liu, Jinglong Chen, Haixin Lv, Jun Wang, and Xinwei Zhang. 2023. Unsupervised multimodal anomaly detection with missing sources for liquid rocket engine. *IEEE Transactions on Neural Networks and Learning Systems* 34, 12 (2023), 9966–9980.

[14] The Apache Software Foundation. 2012. Apache Thrift. Retrieved November 21, 2024 from https://thrift.apache.org

[15] Zoubin Ghahramani and Michael Jordan. 1993. Supervised learning from incomplete data via an EM approach. In *Proceedings of the Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector (Eds.), Vol. 6.

[16] Anton Gulenko, Florian Schmidt, Alexander Acker, Marcel Wallschläger, Odej Kao, and Feng Liu. 2018. Detecting anomalous behavior of black-box services modeled with distance-based online clustering. In *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 912–915.

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

[18] Zilong He, Pengfei Chen, Xiaoyun Li, Yongfeng Wang, Guangba Yu, Cailin Chen, Xinrui Li, and Zibin Zheng. 2023. A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems. *IEEE Transactions on Neural Networks and Learning Systems* 34, 4 (2023), 1705–1719.

[19] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021. Diagnosing performance issues in microservices with heterogeneous data source. In *Proceedings of the 2021 IEEE International Conference on Parallel*. 493–500.

[20] 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 *Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 66–78.

[21] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*. San Diega, CA, USA.

[22] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, Los Alamitos, CA, USA, 492–504.

[23] Scikit learn Developers. 2025. TSNE – scikit-learn 1.6.1 documentation. Accessed: 2025-03-20.

[24] 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 *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE'23)*. IEEE Press, 1750–1762.

[25] 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 *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE Press, 1724–1736.

[26] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2023. SwissLog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (jul 2023), 2762–2780.

[27] Zeyan Li, Wenxiao Chen, and Dan Pei. 2018. Robust and unsupervised KPI anomaly detection based on conditional variational autoencoder. In *Proceedings of the 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. 1–9.

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *Proceedings of the 2008 8th IEEE International Conference on Data Mining*. 413–422.

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

[30] Siyuan Liu, Lei Chen, and Lionel M. Ni. 2014. Anomaly detection from incomplete data. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 2 (2014), 1–22.

[31] 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 *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (Macao, China) *(IJCAI'19)*. AAAI Press, 4739–4745.

[32] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. 2016. *Microservice Architecture: Aligning Principles, Practices, and Culture*. " O'Reilly Media, Inc.".

[33] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly detection and classification using distributed tracing and deep learning. In *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2019), 241–250.

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

[35] Daniel A. Newman. 2014. Missing data: Five practical guidelines. *Organizational Research Methods* 17, 4 (2014), 372–411.

[36] João Nobre, E. J. Solteiro Pires, and Arsénio Reis. 2023. Anomaly detection in microservice-based systems. *Applied Sciences* 13, 13 (2023), 7891–7913. Retrieved from https://www.mdpi.com/2076-3417/13/13/7891

[37] Mahsa Panahandeh, Abdelwahab Hamou-Lhadj, Mohammad Hamdaqa, and James Miller. 2024. ServiceAnomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics. *Journal of Systems and Software* 209 (2024), 111917.

[38] M. N. Norazian Ramli, A. S. Yahaya, N. A. Ramli, N. F. F. M. Yusof, and M. M. A. Abdullah. 2013. Roles of imputation methods for filling the missing values: A review. *Advances in Environmental Biology* 7, 12 S2 (10 2013), 3861+.

[39] Rui Ren, Yang Wang, Fengrui Liu, Zhenyu Li, and Gaogang Xie. 2023. Triple: The interpretable deep learning anomaly detection framework based on trace-metric-log of microservice. In *Proceedings of the 2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. 1–10.

[40] Areeg Samir and Claus Pahl. 2019. DLA: Detecting and localizing anomalies in containerized microservice architectures using markov models. In *Proceedings of the 2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE Computer Society, Los Alamitos, CA, USA, 205–213.

[41] 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 (KDD)*. Association for Computing Machinery, 2828–2837.

[42] Peipeng Wang, Xiuguo Zhang, Zhiying Cao, and Zihan Chen. 2024. MADMM: Microservice system anomaly detection via multi-modal data and multi-feature extraction. *Neural Computing and Applications* 36, 25 (Sep 2024), 15739–15757.

[43] Tingting Wang and Guilin Qi. 2024. A comprehensive survey on root cause analysis in (micro) services: Methodologies, challenges, and trends. arXiv:2408.00803. Retrieved from https://arxiv.org/abs/2408.00803. (2024).

[44] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Vol. 236. Springer.

[45] Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. 2023. Unsupervised anomaly detection on microservice traces through graph VAE. In *Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) *(WWW'23)*. Association for Computing Machinery, New York, NY, USA, 2874–2884.

[46] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications. In *Proceedings of the 2018 World Wide Web Conference (WWW)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 187–196.

[47] Jingyu Yang, Zuogong Yue, and Ye Yuan. 2023. Deep probabilistic graphical modeling for robust multivariate time series anomaly detection with missing data. *Reliability Engineering & System Safety* 238 (2023), 109410.

[48] 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. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (2021), 1448–1460.

[49] Changchang Yin, Ruoqi Liu, Dongdong Zhang, and Ping Zhang. 2020. Identifying sepsis subphenotypes via time-aware multi-modal auto-encoder. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD'20)*. Association for Computing Machinery, New York, NY, USA, 862–872.

[50] Kun Yin, Meng Yan, Ling Xu, Zhou Xu, Zhao Li, Dan Yang, and Xiaohong Zhang. 2020. Improving log-based anomaly detection with component-aware analysis. In *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 667–671.

[51] Yali Yuan, Sripriya Srikant Adhatarao, Mingkai Lin, Yachao Yuan, Zheli Liu, and Xiaoming Fu. 2020. ADA: Adaptive deep log anomaly detector. In *Proceedings of the IEEE INFOCOM 2020 - IEEE Conference on Computer Communications* (Toronto, ON, Canada). IEEE Press, 2449–2458.

[52] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. 2022. DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning. In *Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 623–634.

[53] Xinwei Zhang, Yong Feng, Jinglong Chen, Zijun Liu, Jun Wang, and Hong Huang. 2024. Knowledge distillation-optimized two-stage anomaly detection for liquid rocket engine with missing multimodal data. *Reliability Engineering & System Safety* 241 (2024), 109676.

[54] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, 807–817.

[55] 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 2018 World Wide Web Conference (WWW)*. Association for Computing Machinery, 5639–5649.

[56] 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 (ESEC/FSE)*. Association for Computing Machinery, New York, NY, USA, 527–539.

Received 11 April 2025; revised 7 February 2026; accepted 23 February 2026