This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

A Comprehensive Benchmark and Empirical Study of Trace Anomaly Detection

Yongqian Sun, *Member, IEEE*, Minyi Shao, Xiaohui Nie *Member, IEEE*, Kaiwen Yang, Xingda Li, Bowen Hao, Shenglin Zhang, *Member, IEEE*, Changhua Pei, Dongbiao He, Yanbiao Li, Dan Pei, *Senior Member, IEEE*

***Abstract*—The growing complexity of modern Internet applications and the widespread use of microservice architectures have amplified the need for efficient trace anomaly detection to maintain system stability. Despite the fact that many trace anomaly detection algorithms have been proposed to identify abnormal behaviors, a comprehensive evaluation of these methods is lacking, which makes it difficult for developers to choose the most suitable algorithm for real-world applications. To address this gap, we present *TADBench*, a comprehensive and extensible benchmark for trace anomaly detection. *TADBench* consolidates diverse publicly available trace datasets and algorithms into a unified repository, standardizes data formats, and incorporates manual anomaly labels. To ensure reproducibility and fair comparisons, we propose a modular evaluation framework supporting end-to-end model assessment. Additionally, we provide practical guidance for algorithm selection based on specific data attributes by evaluating their performance across datasets with different characteristics, thereby effectively bridging the gap between academic research and industrial deployment. To the best of our knowledge, this is the first comprehensive empirical study of trace anomaly detection algorithms. Our findings aim to facilitate the adoption of these methods in production environments, offering actionable insights for developers and researchers.**

***Index Terms*—Trace anomaly detection, Empirical study.**

I. Introduction

In recent years, the rapid development of modern Internet applications, accompanied by the expansion of business operations, has led to a significant increase in system complexity. Microservice architecture [1] has emerged as the preferred architectural choice for many Internet applications due to its well-documented advantages in scalability, flexibility, maintainability, and efficient resource use [2] [3]. This architecture enables developers and engineers to independently develop, deploy and update services, greatly boosting development efficiency and system responsiveness [1]. However, the highly modular and distributed structure also poses challenges in system monitoring and management.

To ensure the smooth operation and efficient management of microservice systems, trace anomaly detection serves as an indispensable tool for maintaining service quality and enhancing user experience. It enables the real-time identification of issues within the service interaction network, facilitating the prompt localization and resolution of faults and thereby preventing service interruptions or performance degradation [4]. Moreover, tracing request propagation paths is essential for enhancing system observability and maintainability. Developers can gain a comprehensive understanding of service interaction patterns by monitoring the dependencies and sequences of service calls, which allows them to optimize service performance and improve system stability [5] [6]. Furthermore, historical trace analysis can reveal long-term performance trends and abnormal patterns, offering crucial insights for system optimization and future architectural improvements [7]. Thus, effective trace anomaly detection technologies are vital for comprehensively understanding system states and ensuring the stable operation and efficient management of microservice systems.

Nowadays, many trace anomaly detection algorithms have been developed to swiftly identify potential abnormal behaviors. Despite extensive research in this field, comprehensive public benchmarks for evaluating these algorithms remain largely unavailable. This gap can be attributed to several challenges:

(1) Data availability. Although many anomaly detection algorithms and trace datasets have been open-sourced, previous efforts have failed to systematically consolidate these valuable resources. This lack of integration makes it difficult for researchers to access datasets and tools efficiently, as well as to conduct meaningful comparisons and validations of existing methods. Besides, existing trace datasets generally lack well-defined anomaly labels, limiting their utility and the accuracy of anomaly detection models. Inconsistency in data formats also makes data processing and analysis more difficult, further complicating subsequent research efforts.

(2) Absence of standardized evaluation framework. Inconsistent data preprocessing methods, inaccurate algorithm implementations, and the lack of unified testing pipelines contribute to reproducibility problems. In addition to making it challenging to compare algorithms equitably, these drawbacks impede innovation by directing researchers' focus away from developing novel approaches and into redundant implementation tasks.

(3) Difficulty in algorithm adoption for different applications. While several algorithms may perform better in different scenarios, it is still difficult for operations personnel to quickly identify the optimal algorithm for real-world production environments. Limited empirical analysis and recommendation strategies further exacerbate the difficulties in selecting and

Yongqian Sun, Minyi Shao, Kaiwen Yang, Xingda Li, Bowen Hao, and Shenglin Zhang are with the College of Software, Nankai University, Tianjin 300071, China (e-mail: sunyongqian@nankai.edu.cn, {2120230754, 2112122, 2210352, 2120230749}@mail.nankai.edu.cn, zhangsl@nankai.edu.cn).

Xiaohui Nie, Changhua Pei, Dongbiao He, and Yanbiao Li are with Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), Beijing 100083, China (e-mail: xhnie, chpei@cnic.cn, herbertt12@gmail.com, lybmath@cnic.cn). *(corresponding author: Xiaohui Nie.)*

Dan Pei is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: peidan@tsinghua.edu.cn.)

applying anomaly detection algorithms, which may lower the effectiveness of troubleshooting.

To tackle these challenges, we propose *TADBench*, a comprehensive and extensible benchmark for trace anomaly detection. *TADBench* makes diverse publicly available trace datasets and algorithms into a unified repository with standard data formats and accurate anomaly labels. **For Challenge 1:** We collect, organize, and make a publicly available repository containing trace datasets and anomaly detection algorithms. This repository provides researchers with the necessary resources for in-depth study and comprehensive comparisons. In addition, we standardize the formats of available trace datasets and provide anomaly labels based on human feedback. These enhancements significantly improve the datasets' usability and ensure greater accuracy for future analysis. **For Challenge 2:** In response to the lack of a standardized evaluation framework, we develop a comprehensive framework that includes trace collection, trace preprocessing, model adaptation, and model evaluation. This enables thorough empirical assessments of diverse algorithms across multiple datasets, ensuring reproducibility and promoting robust comparative studies. **For Challenge 3:** Based on extensive experimental results, we propose tailored recommendation strategies for selecting anomaly detection algorithms. These strategies account for the unique characteristics of various trace datasets and provide practical guidance for selecting the most appropriate algorithm for specific scenarios.

The key contributions of this paper can be summarized as follows:

(1) **Public Benchmark.** We establish an open-source repository containing trace data and algorithms and offer a unified and standardized benchmark for evaluating trace anomaly detection algorithms. The open-source data repository has a total size of 3.6 GB, containing approximately 1.04 million traces. Our code is available at https://github.com/nkalgo/TADBench.git.

(2) **Accurate Data Labeling.** We standardize available trace datasets and provide manually labeled anomaly annotations, improving the datasets' quality and applicability. Around 210,000 traces are labeled as structural anomalies or latency anomalies.

(3) **Empirical Study.** In response to the question of whether a universally effective trace anomaly detection algorithm exists, we conduct a comprehensive and systematic empirical study of trace anomaly detection algorithms, analyzing their performance across several datasets and identifying their strengths and limitations. To the best of our knowledge, this is the first comprehensive empirical study in this area.

(4) **Algorithm Recommendation Strategies.** We propose data-driven recommendation strategies for selecting the most suitable anomaly detection algorithms based on trace characteristics and operational requirements.

II. Background

A. Trace Structure

Microservice architecture has fundamentally transformed the way modern applications are developed and operated, enabling systems to be more modular, scalable, flexible, and durable. In this architecture, an application is composed of a collection of loosely coupled services, each responsible for a specific business function. These services communicate with each other through lightweight protocols and are independently deployable, allowing for rapid iteration and fault isolation. However, as the number of services grows, tracking and managing their interactions becomes increasingly complex. This is where distributed tracing plays a crucial role. As illustrated in Fig. 1, a microservice system typically consists of three layers: the service layer, where microservices like *ts-travel-service* and *ts-ticketinfo-service* function; the container layer, which provides runtime environments for these services, ensuring resource isolation and scalability; and the server layer, where the containers are hosted on physical or virtual machines.

Distributed tracing records the flow of requests as they propagate through different system components, giving insight into how microservices interact with one another. By tracing each request's path across different services, it becomes possible to understand not only the performance attributes of distinct services but also their dependencies and interactions with one another. Common distributed tracing systems mainly include Jaeger [8], Zipkin [9], and SkyWalking [10], all of which adhere to the OpenTracing [11] specification. OpenTracing defines two core components: traces and spans. Traces serve as essential tools for describing the execution paths of requests and play a pivotal role in comprehending and evaluating system behavior. Conceptually, a trace can be represented as a directed acyclic graph (DAG) that encapsulates all the services and operations involved in processing a request, along with their temporal dependencies, thereby providing a holistic view of system behavior.

An example of trace is shown in Fig. 1. The illustrated trace comprises seven spans, with each span representing a specific service call. Each trace is uniquely identified by its own trace ID, providing a global reference for tracking the entire execution flow. Meanwhile, each individual span in the trace captures detailed call-related information, including the trace ID, a unique span ID, and the parent span ID. The parent span ID establishes hierarchical relationships among spans, thus constructing the structure of the entire trace. Furthermore, spans typically record time-related information such as the start time and duration, as well as the service name, specific operation name, and status code.

B. Trace Anomalies

Trace data is primarily utilized for system operation and maintenance, supporting tasks such as fault diagnosis, root cause analysis, and system visualization. Among these tasks, trace anomaly detection is of particular importance. Generally speaking, trace anomalies can be categorized into two primary types:

Latency Anomalies. Latency anomalies occur when the execution time of an operation exceeds its normal range, which can be detected when spans exhibit abnormally long durations. Typically, such anomalies not only impact the target

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

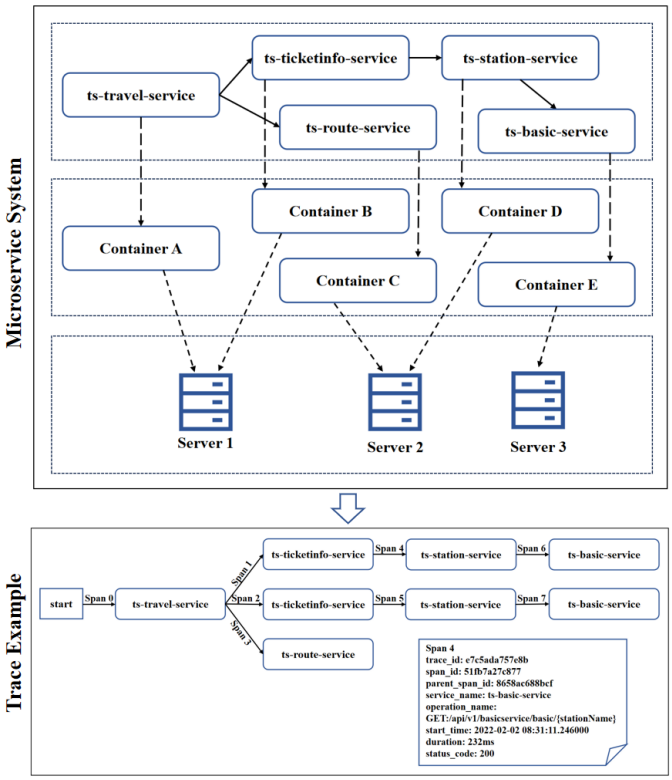


Fig. 1: Microservice System and Trace

service but can also have cascading effects on its upstream dependencies, which may impair overall system performance and have a negative impact on user experience. For instance, Spans 0, 1, and 2 have abnormally high latencies of 237 ms, 235 ms, and 230 ms, respectively, as illustrated in Fig. 2.

The reasons for causing latency anomalies may include performance bottlenecks, resource contention, and network delays. For example, a bottleneck within a service's processing (e.g., inefficient database queries or excessive CPU usage) directly extends the execution time of the corresponding span. Similarly, contention for limited resources (e.g., memory, disk I/O, or thread pool capacity) increases operation waiting times, thereby amplifying service latency. In addition, anomalous network conditions, including packet loss or congestion between microservices, introduce transmission delays in requests and responses, further prolongs trace latency beyond its normal statistical range. These issues are reflected in trace data as abnormally prolonged durations of specific spans, leading to service-level latency that exceeds the normal statistical range.

Structural Anomalies. Deviations from expected service invocation sequences inside a trace are known as structural anomalies. These deviations may include unexpected service calls, missing calls, and call order errors.

Unexpected service calls occur when a service or operation is invoked unexpectedly, as shown in Fig. 3, where Service B incorrectly invokes Service G instead of Service E. This type of anomaly may result from configuration drift (e.g., outdated service registries) leading to incorrect routing rules, fallback

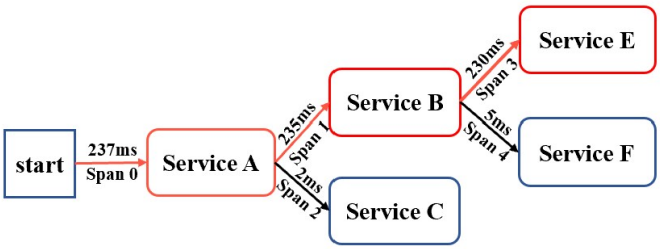


Fig. 2: Latency Anomaly

logic failures (e.g., circuit breakers redirecting traffic to unintended or non-redundant services), or dependency mismatches (e.g., API version upgradation without backward compatibility). These business-related configuration or deployment issues can inadvertently cause services to interact with incorrect or unexpected counterparts, thereby introducing anomalies in the expected trace.

Missing calls refer to cases where an expected service or operation is absent, like Service E is missing in Fig. 4. These anomalies may be caused not only by system failures or network latency, but more critically by business process incompleteness. For instance, a workflow step may be skipped due to unmet preconditions such as data validation failures, authorization denials, or business rule violations. Additionally, service outages, communication timeouts, or misconfigured routing (such as incorrect service addresses, unpublished APIs, or authentication failures) can also interrupt the normal flow, preventing certain calls from being made. Furthermore, operational decisions such as temporary service disruptions during deployments may lead to the absence of expected service interactions in the trace.

Moreover, call order errors occur when the sequence of service invocations deviates from the expected order. For example, in Fig. 5, Service F mistakenly invokes Service E, whereas it should have been Service B invoking Service E. Such errors often stem not merely from race conditions in asynchronous systems, but more specifically from concurrency control issues in event-driven architectures where events are processed in improper sequences, violating logical dependencies between services. Moreover, version incompatibilities between microservices can induce unintended temporal sequencing of operations, which often materializes as observable structural anomalies in the trace.

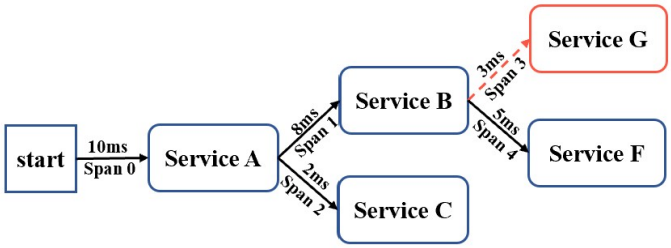


Fig. 3: Structural Anomaly (unexpected)

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

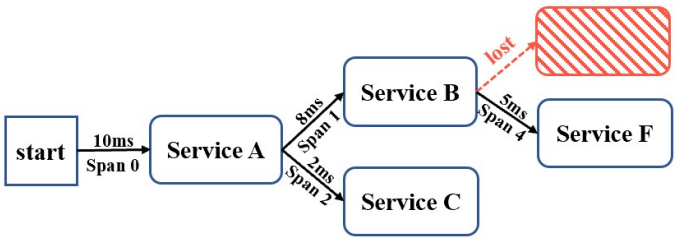


Fig. 4: Structural Anomaly (missing)

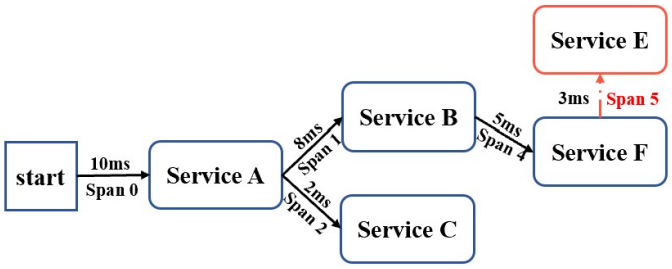


Fig. 5: Structural Anomaly (out-of-order)

C. Trace Anomaly Detection Algorithms

Many algorithms for trace anomaly detection exhibit unique design principles and robust performance under specific conditions. Traditionally, algorithms in this domain have been categorized into statistic-based and model-based approaches [12]. However, as deep learning has become more and more popular, an increasing number of algorithms now employ deep learning models, rendering the traditional classification inadequate for capturing the complexity and diversity of state-of-the-art trace anomaly detection techniques. To address this limitation, we propose a classification based on underlying architectures, providing a more precise and systematic framework. Accordingly, we divide these algorithms into three main categories: VAE-based, GNN-based, and LSTM-based algorithms, which represent the dominant design paradigms in the field. An overview of this classification is presented in Fig. 6. Except for PUTraceAD, which is a semi supervised learning algorithm, all others in Fig. 6 are unsupervised.

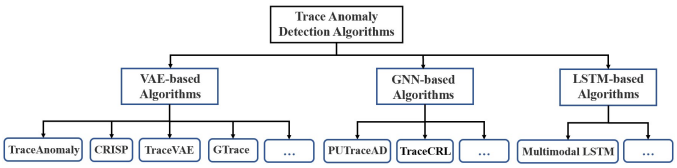


Fig. 6: Algorithm Overview

*1) **VAE-based Algorithms**:* Variational Autoencoders (VAEs) [13] serve as the core model for this category. These methods aim to reconstruct normal patterns in trace data, leveraging the learned latent representations to identify anomalies.

TraceAnomaly. TraceAnomaly [14] employs deep variational Bayesian networks with posterior flow [15] to model normal trace patterns. It uses a Service Trace Vector (STV) to represent data, with each dimension denoting a distinct call path and the corresponding response time indicated by its value.

CRISP. Adopting the same anomaly detection model as TraceAnomaly, CRISP [16] encodes traces by extracting the critical path using a recursive algorithm. It constructs a Service Critical Path Vector (SCPV) by subtracting the durations of child spans when recording spans on the critical path.

TraceVAE. Using Graph-VAE architectures [17]–[19], TraceVAE [20] separately models structural and temporal features via a structure VAE and a time VAE. Additionally, TraceVAE incorporates Graph Attention Networks (GATs) [21] to capture correlations among nodes in the trace graphs.

GTrace. By combining graph-wise and node-wise VAE models, GTrace [12] reconstructs both structural and latency features of trace graphs. To enhance detection efficiency, it groups traces with the same substructures. Moreover, GTrace adopts Tree-LSTM [22] to generate shared encodings for identical sub-tree structures, further supporting its substructure-based grouping strategy.

*2) **GNN-based Algorithms**:* In this category, Graph Neural Networks (GNNs) [23], [24] serve as the foundation. These models excel at capturing structural and relational information in trace graphs.

PUTraceAD. As a semi-supervised anomaly detection method, PUTraceAD [25] trains a model based on GATs and the nnPU [26] algorithm using a limited amount of labeled abnormal traces. Notably, when generating graph representations for traces, PUTraceAD uses WordPiece [27] and a pre-trained BERT model [28] to generate 768-dimensional embeddings for service and operation names while creating graph representations for traces.

TraceCRL. Following the GraphCL (Graph Contrastive Learning with Augmentations) framework [29], TraceCRL [30] leverages contrastive learning and graph neural networks to learn effective trace representations. Initially, it utilizes DeepWalk [31] to generate vector representations for each operation. Compared to other algorithms that only use duration as the temporal feature for each invocation, TraceCRL incorporates more detailed temporal features, such as the quantile of the invocation's start time and the proportion of the local execution time. By employing One-Class SVM [32] as a subsequent detection model, TraceCRL is widely used for anomaly detection tasks.

*3) **LSTM-based Algorithms**:* Long Short-Term Memory (LSTM) networks [33] are used in this category to model sequential dependencies and detect both latency and structural anomalies in traces.

Multimodal LSTM. Multimodal LSTM [34] combines temporal and structural features into a joint representation. Each span is encoded using a vector with one-hot call path encoding and normalized duration. Rare calls are excluded to reduce noise and maintain consistency. LSTM networks are applied to model the sequential relationships.

In summary, a wide range of trace anomaly detection algorithms have been developed in recent years, each with unique design models. Selecting an appropriate detection algo-

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

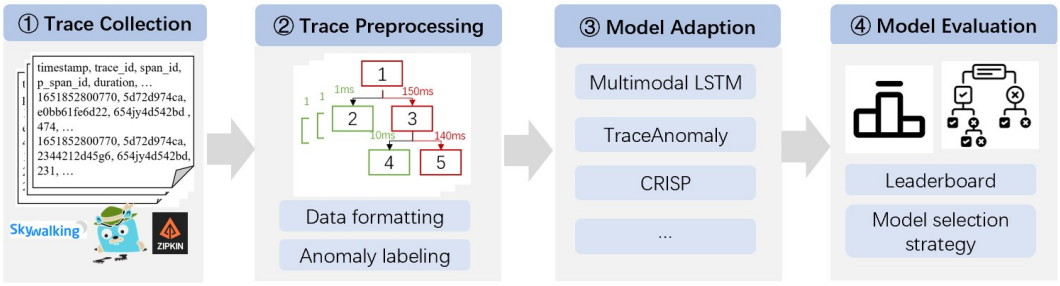


Fig. 7: Evaluation Framework

rithm for real-world applications remains a critical challenge. This paper aims to address this issue by conducting a comprehensive evaluation of various algorithms under different conditions. Through extensive experiments, we systematically analyze their performance, highlight their advantages and drawbacks, and provide recommendations for the most effective selection strategy.

III. *TADBench* Design

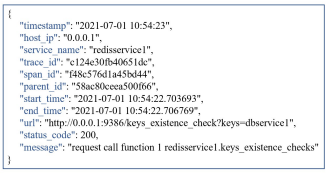
To systematically evaluate the performance of trace anomaly detection algorithms, we propose an evaluation framework depicted in Fig. 7, which consists of four main stages: trace collection, trace preprocessing, model adaptation, and model evaluation. First, we collect the available trace data, followed by preprocessing to standardize the data format. Next, the algorithms needs to ensure proper data format adaptation to receive and process the pre-processed data for evaluation. Finally, the performance of different algorithms is evaluated based on the experimental results.

A. Dataset Overview and Unified Format

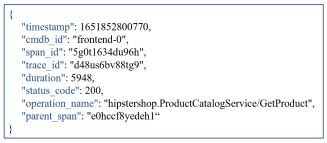
In this study, we conduct an in-depth analysis of multiple datasets, including TrainTicket [35], [36], GAIA (Generic AIOps Atlas) [37], AIOps2020 [38], AIOps2022 [39] and AIOps2023. These datasets are generally based on the OpenTracing format mentioned in Section II-A.

- **TrainTicket**: The TrainTicket dataset, sourced from the open-source data in the PUTraceAD [25] paper, is generated through fault injection in the microservice system TrainTicket.
- **GAIA**: GAIA, dedicated to anomaly detection, log analysis, and fault localization, is derived from CloudWise microservice simulation system.
- **AIOps2020, AIOps2022 and AIOps2023**: The AIOps2020, AIOps2022, and AIOps2023 datasets originate from the CCF International AIOps Challenge held in 2020, 2022, and 2023 respectively. AIOps2020 originates from a real-world production microservice system, while AIOps2022 and AIOps2023 simulate anomalies through fault injection.

However, the specific details in different datasets may vary, as exemplified by the GAIA and AIOps2022 datasets shown in Fig. 8a and Fig. 8b, respectively. These differences encompass various aspects, including the meaning of fields, field names, the number of fields, and the units of values. In previous experiments, researchers often had to preprocess each dataset separately to meet the format requirements of different algorithms. When new datasets emerged, adapting them to various algorithms was time-consuming and severely hindered experimental progress. Furthermore, there were significant difficulties in comparing the features of various datasets or combining them.



(a) Data Format for GAIA



(b) Data Format for AIOps2022

Fig. 8: Different Trace Formats

To address these challenges, we propose a standardized format consisting of two primary classes: Trace and Span, as demonstrated in Fig. 9.

- **Trace Class.** Each trace is uniquely identified by its *trace_id* and contains fields such as *root_span* (denoting the start of the trace) and *span_count* (denoting the total number of spans). The *anomaly_type* field categorizes anomalies into four types: 0 (normal), 1 (only latency anomaly), 2 (only structural anomaly), and 3 (both latency and structural anomalies). Additionally, the *source* field denotes the data source.
- **Span Class.** Each span records detailed call information, including *trace_id*, *span_id*, *parent_span_id*, and *children_span_id*, which collectively construct the trace tree. Each span also contains time information (*start_time* and *duration*), service and operation details (*service_name* and *operation_name*), anomaly labels (*anomaly*) and status information (*status_code*). The *latency* and *structure* fields further specify latency and structural anomaly, while the *extra* field captures additional contextual details.

This unified format eliminates the need to preprocess raw data for each algorithm separately. Instead, we merely convert the data into this unified format, and then adapt the format to various algorithms. Consequently, each new dataset in any format can be compatible with any algorithm without the need for extra data preprocessing once it has been converted to

TABLE I: Dataset Characteristics

	Average Trace Depth	Average Span Numbers per Trace	Granularity	Service Scale	Average Service Number per Trace	Operation Scale	Average Operation Number per Trace
TrainTicket	3.3	39.0	Operation	29	5.9	64	7.6
GAIA	4.7	9.3	Service	10	6.4	-	-
AIOps2020	5.5	23.1	Service	10	7.8	-	-
AIOps2022	4.2	21.7	Operation	40	6.9	29	11.6
AIOps2023	3.8	14.5	Service	37	1.3	-	-

Trace:
{
"trace_id": "Unique identifier for each trace.",
"root_span": "The starting span of the trace.",
"span_count": "The total number of spans in the trace.",
"anomaly_type": "Anomaly type: 0 (normal), 1 (latency anomaly only), 2 (structural anomaly only), 3 (both latency and structural anomalies).",
"source": "The data source of the trace."
}

Span:
{
"trace_id": "Identifies the trace this span belongs to.",
"span_id": "Unique identifier for the span.",
"parent_span_id": "The identifier of the parent span in the trace tree.",
"children_span_id": "Identifiers of the child spans in the trace tree.",
"start_time": "The start time of the span.",
"duration": "The duration of the span.",
"service_name": "The name of the service associated with the span.",
"operation_name": "The operation name performed in the span.",
"anomaly": "Indicates whether the span is anomalous.",
"status_code": "The status information for the span.",
"latency": "Specifies latency anomalies for the span.",
"structure": "Specifies structural anomalies for the span.",
"extra": "Captures additional contextual details about the span."
}

Fig. 9: Unified Format

the unified format. Besides, this unified format supports a systematic comparison of datasets, enabling operators to gain deeper insight into system behavior, and facilitating efficient anomaly diagnosis and resolution.

To evaluate the differences between the five datasets, we examine their respective characteristics, with the results summarized in Table I. Key features are systematically compared in the table, with particular attention paid to differences in average trace depth (ranging from 3.3 to 5.5), average span numbers per trace (ranging from 9.3 to 39.0), monitoring granularity (service/operation-level), service scale (ranging from 10 to 40), average service numbers per trace (ranging from 1.3 to 7.8), operation scale (ranging from 29 to 64), and average operation numbers per trace (ranging from 7.6 to 11.6). Notably, TrainTicket has the highest operation scale (64), while AIOps2022 contains the most services (40). Additionally, GAIA, AIOps2020 and AIOps2023 lack operation-level metrics.

Table II summarizes the anomaly ratios for the five datasets across three categories: total, structure, and latency. The total anomaly ratios range from 21.8% (AIOps2020) to 53.6% (AIOps2023). For structural anomalies, the ratios range from 3.5% (AIOps2020) and 39.0% (AIOps2023). For latency anomalies, the ratios range from 21.5% (AIOps2020) to 35.9% (AIOps2022).

TABLE II: Dataset Anomaly Ratio

	Total	Structure	Latency
TrainTicket	32.9%	26.7%	28.5%
GAIA	49.9%	21.8%	29.9%
AIOps2020	21.8%	3.5%	21.5%
AIOps2022	36.3%	4.0%	35.9%
AIOps2023	53.6%	39.0%	24.0%

B. Dataset Labeling

Despite being publicly accessible, the five datasets are generally collected systematically without specific labels. Consequently, researchers cannot directly distinguish between normal and abnormal traces in these datasets, posing challenges for further analysis and application. Without proper labeling, it is impossible to successfully validate the performance of anomaly detection algorithms. Therefore, recognizing that trace anomalies are generally categorized into latency and structural anomalies, we develop the labeling process shown in Fig. 10.

These datasets usually contain fault microservice information and fault occurrence time. Using this critical data, we divide the traces into two categories: normal traces and fault-injected traces. For latency anomaly detection, we assess whether the latency of each service in a fault-injected trace falls within the normal range. As discussed in Section II-B, some business-related anomalies—such as performance bottlenecks, resource contention, and network delays—produce detectable latency distribution shifts in trace data. To enhance accuracy, we group services based on their shared calling paths from the root service, rather than relying solely on service names. Next, we use a Gaussian distribution to model the service latency distribution in normal traces. Specifically, let μ and σ denote the mean and standard deviation of the latency distribution, respectively. A service latency L in a fault-injected trace is considered abnormal if it satisfies:

$$L \notin [\mu - 3\sigma, \mu + 3\sigma] \tag{1}$$

following the 3-sigma rule, which assumes that 99.73% of normally distributed data falls within three standard deviations of the mean. We employ the 3-sigma rule because it is a statistically robust and widely accepted method for detecting rare, extreme deviations from an established normal pattern [14]. If such an anomaly is detected, both the service and the trace are labeled as latency anomalies.

For structural anomaly detection, we compare the structural differences between faulty and normal traces. Initially, we collect all patterns and their frequencies from normal traces. Then, for each fault-injected trace, we examine if its structure exists in the normal dataset. If the structure is rare or absent,

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

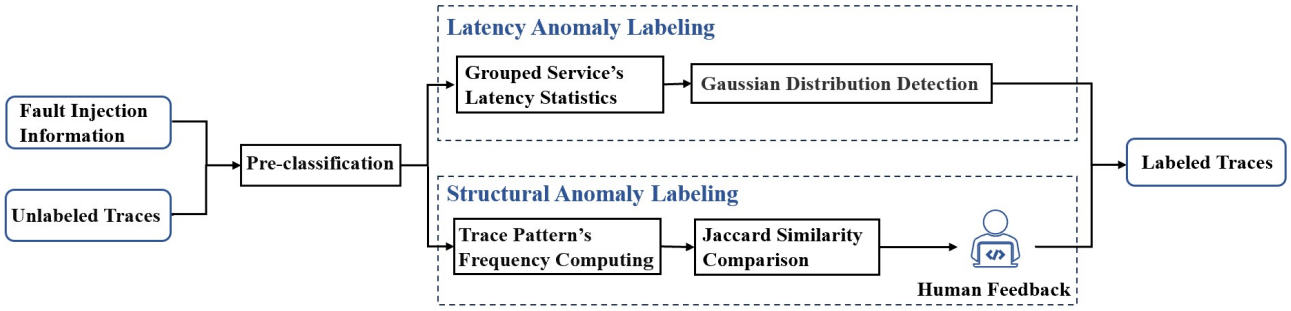


Fig. 10: Trace Labeling Process

we apply the Jaccard similarity to identify the most similar pattern from the normal traces. The Jaccard similarity between two traces T and P is defined as:

$$J(T,P) = \frac{|\mathcal{S}(T) \cap \mathcal{S}(P)|}{|\mathcal{S}(T) \cup \mathcal{S}(P)|} \tag{2}$$

where $\mathcal{S}(T)$ and $\mathcal{S}(P)$ denote the set of services in traces T and P respectively. We then select the most similar normal pattern through:

$$P^* = \arg\max_{P_i \in \mathcal{P}} J(T, P_i) \tag{3}$$

where $\mathcal{P}$ is the collection of normal trace patterns. Subsequently, we perform a detailed examination by manually comparing the faulty trace T with the most similar normal pattern P^*. The trace is finally labeled as a structural anomaly if this comparison confirms the presence of deviations such as missing calls, unexpected service calls, or call order errors.

C. Algorithm Adaption

In general, different algorithms often call for specific trace formats in their architecture. In prior studies, researchers who intended to use open-source algorithm code had to implement additional data preprocessing steps to meet the algorithm requirements, which incurred considerable time overhead. To address this, our work have standardized the data format, allowing us to simply add code to each algorithm's implementation that converts the unified format into the specific format required by the algorithm. As a result, when a new dataset needs to be applied to these algorithms for detection, it only needs to be converted to the standardized format now, which will then be automatically compatible with all algorithms, eliminating the need for cumbersome format adjustments. The open-source code we have released already incorporates this functionality.

In addition, to ensure the standardization, modularity, and extensibility of algorithm evaluation, we construct an algorithm Software Development Kit (SDK), as illustrated in the Fig. 11. By requiring each algorithm to inherit from the common abstract base class, we standardize the format of inputs and outputs across different algorithms. This design not only simplifies the integration of new algorithms into the evaluation pipeline but also reduces the effort required for extra adaptation or modification. More importantly, it facilitates fair and consistent comparisons. The SDK also serves as a scalable and extensible foundation for future research, enabling rapid prototyping, algorithm benchmarking, and collaborative development in the field of trace anomaly detection.

```python
class TADTemplate(ABC):
    def __init__(self, dataset_name=None, data_path=None):
        self.dataset_name = dataset_name
        self.data_path = data_path

    @abstractmethod
    def preprocess_data(self):
        """Preprocess raw trace data (feature engineering, normalization, etc.)"""
        pass

    @abstractmethod
    def train(self):
        """Train model and save trained model"""
        pass

    @abstractmethod
    def test(self):
        """Evaluate model by different datasets"""
        pass
```

Fig. 11: Algorithm SDK

D. Evaluation Metrics

In this study, we utilize several evaluation metrics to measure the performance of the anomaly detection models, including **Precision**, **Recall**, **F1-score**, **Accuracy**, and **Time Consumption**. These metrics evaluate both the effectiveness and efficiency of the trace anomaly detection algorithms. They are defined as follows, where TP refers to the number of anomalous traces accurately detected as anomalies, FP refers to the number of normal traces mistakenly detected as anomalies, FN refers to the number of anomalous traces that fail to be identified as such, and TN refers to the number of normal traces correctly detected as normal.

- **Precision**: the proportion of real anomalous traces over all the predicted anomalous traces, which can be calculated using the following formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

- **Recall**: the proportion of predicted anomalous traces over all the real anomalous traces, which can be calculated using the following formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

- **F1-score**: the harmonic mean of the precision and recall, which can be calculated using the following formula:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

- **Accuracy**: the proportion of correctly predicted traces over all the traces, which can be calculated using the following formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{7}$$

- **Time Consumption**: A key metric for assessing model efficiency, particularly in large-scale or real-time applications. This is quantified by the time required for the training and testing phases.

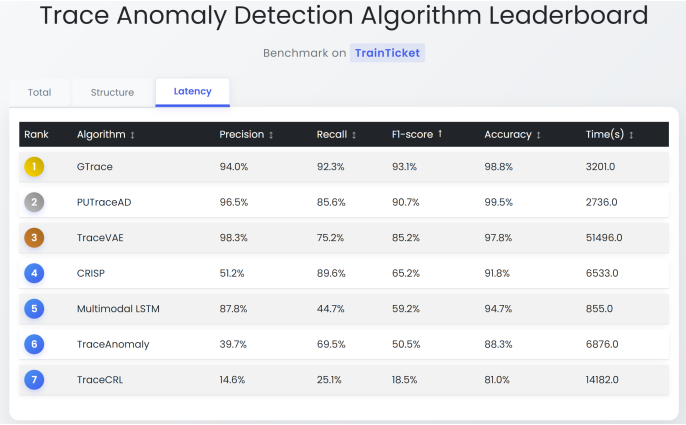


Fig. 12: Trace Anomaly Detection Algorithm Leaderboard

E. Algorithm Leaderboard

To provide a clearer and more comprehensive comparison of trace anomaly detection algorithms, we design and implement a dedicated leaderboard, as illustrated in Fig. 12. This leaderboard systematically displays the performance of different algorithms on different datasets. Specifically, it reports a set of key evaluation metrics — including precision, recall, F1-score, accuracy, and time consumption — across three representative anomaly types: total anomalies, structural anomalies, and latency anomalies. Moreover, the leaderboard supports interactive functionality, allowing users to dynamically sort results based on any chosen metric. This design enables researchers to explore algorithms' strengths and weaknesses from multiple dimensions. By offering a unified and visually accessible platform for comprehensive performance comparison, this leaderboard not only facilitates the identification of state-of-the-art methods but also encourages further improvement in trace anomaly detection research.

IV. Evaluation

In this section, we focus on the evaluation to address the following research questions:

- **RQ1**: Is there an algorithm that consistently outperforms others across all datasets?
- **RQ2**: How well do the algorithms perform across various dataset conditions?
- **RQ3**: How can we choose the algorithm that works best for a certain dataset?

A. Experimental Setup

After performing anomaly labels on each of the five datasets including TrainTicket, GAIA, AIOps2020, AIOps2022 and AIOps2023, we evaluated the performance of the trace-based anomaly detection algorithms on these datasets separately. For each dataset, we divided the normal traces into a training set and a test set in a 2:1 ratio, while all abnormal traces were included in the test set.

During experiments, we generally use default hyperparameters from the corresponding anomaly detection algorithm papers. If the training or detection performance on a particular dataset is suboptimal, hyperparameters are adjusted to optimize the results. All experiments are performed on a server configured with two Intel(R) Xeon(R) Gold 5416S CPUs, 376 GB of RAM, and seven NVIDIA RTX A6000 GPUs, each with 48 GB of GPU memory.

B. Overall Performance (RQ1)

Table III summarizes the overall performance of various algorithms across datasets, and Fig. 13 presents a radar chart comparing the F1-scores achieved by different algorithms on various datasets. **The findings reveal that no single algorithm consistently outperforms others across all datasets.** Specifically, GTrace [12] achieves optimal performance with F1-scores of 99.4% and 71.8% on the TrainTicket and AIOps2020 datasets respectively. TraceVAE [20] demonstrates leading performance on the GAIA and AIOps2022 datasets with F1-scores of 90.9% and 78.9% respectively, while PUTraceAD [25] achieves the highest F1-score of 74.7% on the AIOps2023 dataset. These results highlight that the effectiveness of each algorithm is influenced by dataset characteristics. GTrace's innovative strategy of "predicting latency with structure" proves effective in part datasets, while TraceVAE's integration of GNNs and VAEs addresses complex data characteristics. PUTraceAD's semi-supervised architecture demonstrates unique advantages on the AIOps2023 dataset containing 53.6% anomalies.

To evaluate the algorithms' performance in detecting different types of anomalies, we calculate the F1-scores for both structural and latency anomalies. This analysis does not take into account the dataset differences, as detailed in Table IV. TraceVAE achieves a structural anomaly detection F1-score of 96.8%, highlighting the exceptional performance of its Structure VAE module. The model's GNN encoder explicitly captures topological dependencies within trace data, significantly enhancing its structural modeling capabilities. In

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

TABLE III: Overall Performance (F1: F1-score; ACC: Accuracy)

	TrainTicket		GAIA		AIOps2020		AIOps2022		AIOps2023	
	F1 (%)	ACC (%)	F1 (%)	ACC (%)	F1 (%)	ACC (%)	F1 (%)	ACC (%)	F1 (%)	ACC (%)
Multimodal LSTM	68.5	73.7	89.8	70.7	54.5	41.6	59.2	51.4	64.4	80.5
TraceAnomaly	62.6	82.1	46.3	65.3	56.8	77.9	58.5	78.7	55.4	82.9
CRISP	62.5	82.0	44.5	64.1	57.0	78.0	58.1	78.5	66.4	87.2
PUTraceAD	95.4	98.9	68.6	84.8	48.3	78.6	68.1	88.1	**74.7**	95.5
TraceCRL	55.6	50.9	75.0	73.9	53.0	40.8	53.2	36.2	44.2	57.4
TraceVAE	97.3	98.2	**90.9**	91.1	57.1	72.7	**78.9**	86.7	74.0	90.0
GTrace	**99.4**	80.3	70.9	64.5	**71.8**	81.9	76.5	93.8	70.3	87.6

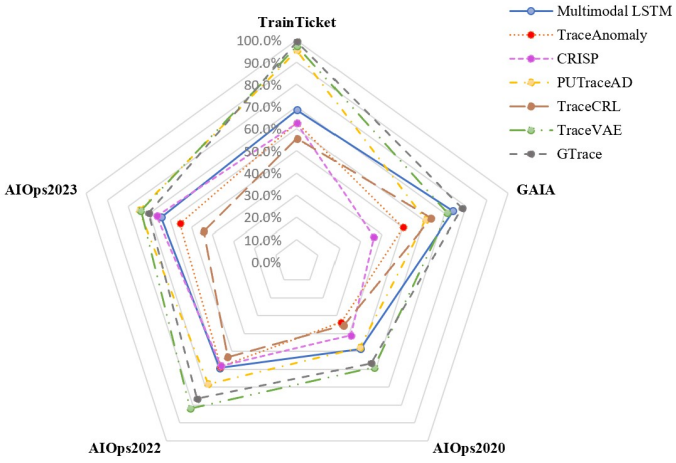


Fig. 13: F1-score Comparison Across Datasets

contrast, GTrace achieves a latency anomaly detection F1-score of 78.2%, primarily due to its focus on modeling latency features at the span level, unlike other algorithms that operate solely at the trace level. This span-level modeling allows GTrace to capture fine-grained latency variations within traces, leading to more precise anomaly detection that might be overlooked by trace-level approaches. Furthermore, this table indicates that current latency detection methods still have room for improvement, making it a promising direction for future research.

TABLE IV: Algorithm Performance under Different Anomaly Types

	Structure	Latency
Multimodal LSTM	76.5%	62.2%
TraceAnomaly	57.7%	56.0%
CRISP	55.8%	56.2%
PUTraceAD	89.1%	61.7%
TraceCRL	65.9%	53.7%
TraceVAE	**96.8%**	76.4%
GTrace	95.6%	**78.2%**

C. Performance on Various Data Characteristics (RQ2)

In order to thoroughly analyze the impact of data characteristics on the performance of trace anomaly detection algorithms, we first combine data from the five distinct datasets into a single, comprehensive dataset. This aggregated dataset serves as a unified foundation for further analysis. To better understand how different features affect algorithm performance, we then reclassify the combined data based on specific feature values, thereby establishing different subsets where data with similar characteristics are clustered together. These newly constructed datasets are subsequently applied to evaluate the performance of different algorithms. The results indicate that these data characteristics significantly influence algorithm performance:

Trace Depth. Fig. 14 clearly illustrates that different algorithms excel at different trace depths. Specifically, when trace depth is ≤ 3, TraceVAE achieves the highest F1-score of 92.2%, slightly outperforming PUTraceAD (91.6%) and GTrace (91.4%). Both GTrace and TraceVAE show distinct performance advantages as trace depth increases. **Notably, when the trace depth is more than 6, TraceVAE significantly outperforms other algorithms, reaching an F1-score of 82.3%**, while GTrace drops to 66.5%. This suggests that many anomaly detection methods are capable of effectively handling relatively simple traces with shallow invocation paths. However, when trace depth grows, the complexity of invocation logic and multi-layer service interactions increases, resulting in wider fault propagation and more noticeable variations in algorithm performance. In such scenarios, TraceVAE's dual-variable graph variational autoencoder [17]–[19] appears better equipped to capture long-range dependencies and contextual information. Meanwhile, GTrace shows its strength in moderate-depth situations, and PUTraceAD remains competitive for traces with small depth.

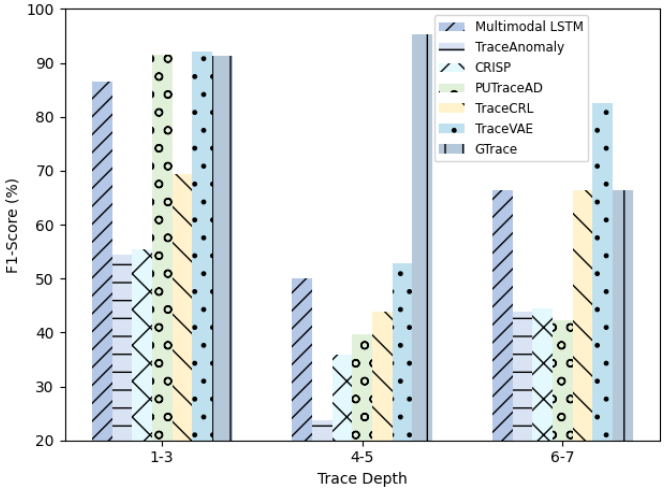


Fig. 14: Algorithm Performance under Different Trace Depths

Span Count. As shown in Fig. 15, **GTrace achieves**

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

the highest F1-score for traces with 1-5 spans (99.7%), 11-30 spans (70.7%), and more than 30 spans (60.8%), demonstrating its robust performance across both simple and highly complex traces. In contrast, TraceVAE excels at 6-10 spans, attaining an F1-score of 74.2%, which suggests that its architecture is well-suited to traces of moderate complexity. Notably, both TraceVAE (99.4%) and PUTraceAD (99.1%) also approach 99% at 1-5 spans, indicating that simple traces can be effectively handled by a variety of methods. Moreover, CRISP exhibits a unique performance trend: it improves steadily with increasing span count, starting from 24.3% for traces with 1–5 spans and reaching 59.3% when the span count exceeds 30. Meanwhile, other methods, such as Multimodal LSTM [34] and TraceCRL [30], exhibit moderate performance but do not outperform the optimal algorithm in any particular range. The span count of a trace reflects both the breadth and depth of its execution logic, encompassing horizontal service calls at the same level and vertical, multi-layered interactions across different services. Traces with more spans often involve more frequent service calls, denser data interactions, and greater opportunities for fault propagation. GTrace's Tree-LSTM [22], specifically designed for tree structures, effectively captures dependencies between parent-child nodes, aligning well with typical trace structures. However, although TraceVAE achieves outstanding results for traces with a medium or small number of spans, its detection capability may be limited by its fixed span count requirement, determined by a predefined *max_node_count*. A small *max_node_count* constrains the model's ability to deal with highly complex traces involving a large number of spans. Meanwhile, CRISP [16] becomes more competitive at higher span counts, suggesting that its focus on extracting critical paths is well-suited for scenarios requiring a lot of invocation. In contrast, methods like PUTraceAD and Multimodal LSTM maintain high accuracy for relatively simple traces but show diminished performance as trace complexity grows, due to their limited capacity for modeling long-range dependencies. For traces with high span counts, approaches capable of capturing extensive dependencies, like GTrace, are expected to yield better anomaly detection results.

Service Count. Fig. 16 illustrates that for traces with 1-4 services, TraceVAE achieves the highest F1-score (89.7%), followed by GTrace (83.5%) and Multimodal LSTM (78.3%). Similarly, TraceVAE maintains its dominance at 5-8 services with an F1-score of 67.9%, while GTrace and Multimodal LSTM are still competitive at 57.7% and 57.0%, respectively. However, **when the number of services exceeds 8, GTrace outperforms the other methods, reaching 68.1%**, whereas TraceAnomaly trails behind at 59.6%. As the number of services in a trace increases, fault diagnosis becomes more challenging. These results suggest that TraceVAE is particularly effective in relatively simple service topologies, where modeling key service-to-service interactions can capture most of the anomaly-relevant information. However, GTrace demonstrates better performance for traces with more services. This is primarily attributed to GTrace's Node-wise VAE encoder, which leverages the *service name* field to capture each service's impact on overall trace latency. In contrast,

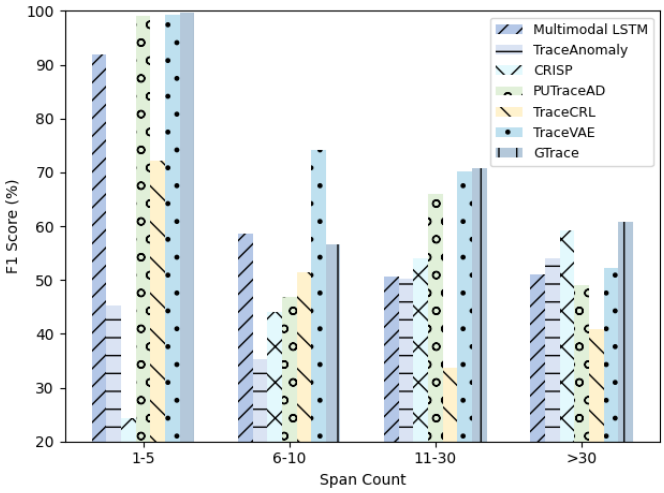


Fig. 15: Algorithm Performance under Different Span Counts

other anomaly detection methods such as PUTraceAD, CRISP, and TraceCRL show relatively limited adaptability, and their inability to consistently model detailed inter-service interactions hinders their performance, especially as system scale increases. These observations underscore the importance of capturing nuanced interactions across multiple services, which is critical for accurate anomaly detection in large-scale trace data.

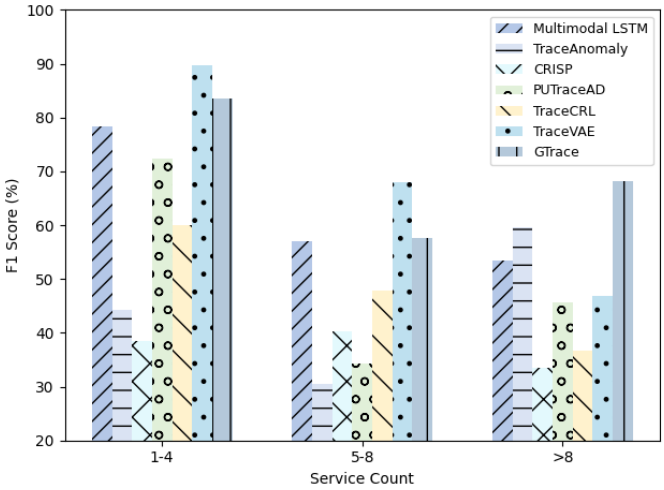


Fig. 16: Algorithm Performance under Different Service Counts

Anomaly Ratio. According to Fig. 17, GTrace achieves the highest F1-score (68.7%) when the anomaly ratio is 0%, whereas TraceVAE leads at 0.5% (69.1%). For 1%, GTrace again outperforms other methods at 61.8%, but **when the anomaly ratio reaches 3%, TraceVAE stands out with 56.6%**. Other algorithms, including Multimodal LSTM, TraceAnomaly, and CRISP, exhibit moderate performance across all anomaly levels. Notably, PUTraceAD loses detection capability for 0% or 0.5% but achieves 41.9% at 1% and 55.0% at 3%. This is consistent with its positive-unlabeled

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

learning paradigm, which relies on the presence of a sufficient number of positive (abnormal) samples to perform effective training. PUTraceAD's detection performance deteriorates in the absence of sufficient anomalies to direct the optimization process. Additionally, Since abnormal traces account for approximately 1% of total traces [40] in real-world production environments, GTrace's robustness at this ratio is particularly noteworthy. However, as the anomaly ratio further increases, unlike methods such as GTrace that use negative log-likelihood (NLL) directly as anomaly scores, TraceVAE's techniques, including Bernoulli & Categorical Scaling, Node Count Normalization, and Gaussian Std-Limit [20], reduce entropy gaps effectively , mitigate the negative impact of noise, and enable more accurate differentiation between normal and anomalous traces. This findings underscore the importance of designing models that can handle increased noise effectively, ensuring reliable anomaly identification even in highly contaminated datasets.

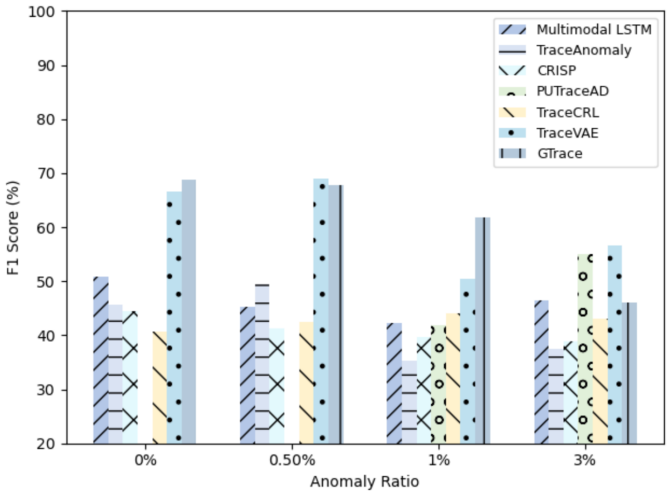


Fig. 17: Algorithm Performance under Different Anomaly Ratios

D. Efficiency analysis of different algorithms

To evaluate the detection efficiency of various algorithms, we measure the time consumption during the training and detection phases using the same dataset. Specifically, 51,386 traces are randomly sampled from the TrainTicket dataset as the training set, while the test set comprises 51,309 normal traces and 25,197 anomalous traces. The time consumption details are presented in Table V.

TABLE V: Training Time and Detecting Speeding of Different Algorithms

Algorithm	Training Time	Detecting Speed
Multimodal LSTM	419 s	4500 traces/s
TraceAnomaly	3422 s	107 traces/s
CRISP	3827 s	107 traces/s
TraceCRL	13176 s	700 traces/s
PUTraceAD	1715 s	548 traces/s
TraceVAE	23497 s	257 traces/s
GTrace	1641 s	10211 traces/s

The results reveal notable differences in time efficiency among the algorithms. Multimodal LSTM demonstrates the lowest training overhead, owing to its approach of independently modeling temporal and structural features with LSTMs. This architecture simplifies the learning process and leads to superior training efficiency compared to other models with more complex graph-based encoders or latent variable mechanisms. Conversely, TraceVAE requires a significant amount of training time, which reflects the computational complexity of its dual-variable graph variational autoencoder architecture, even if it has demonstrated outstanding detection performance in prior experiments. The model's training cost may hinder its real-time adaptability or deployment in resource-constrained environments. Similarly, TraceCRL exhibits considerable time consumption during trace representation generation, even with GPU acceleration. The model's reliance on contrastive learning frameworks, involving large numbers of positive and negative sample pairs, results in high computational complexity. Notably, GTrace achieves remarkable detection speed, primarily attributed to its caching techniques and trace grouping strategies, which allow shared representations and precomputed features to be reused across similar trace structures. By avoiding redundant computation, GTrace effectively minimizes time consumption during detection phases, making it suitable for real-time anomaly detection scenarios where response time is critical.

E. Recommended Algorithms (RQ3)

In previous experiments, we evaluated the performance of various algorithms on datasets characterized by distinct individual features, primarily focusing on the impact of single features on detection performance. However, real-world datasets typically exhibit multiple coexisting features, necessitating a more comprehensive evaluation. To bridge this gap and approximate more realistic application scenarios, we extend our analysis by employing a decision tree model [41] to integrate multiple dataset characteristics, as illustrated in Fig. 18. The results demonstrate that when the proportion of abnormal data in the dataset exceeds 10%, the semi-supervised anomaly detection method PUTraceAD is preferred due to its ability to effectively utilize anomaly information. For datasets with lower anomaly proportions, the choice of the algorithm depends on additional factors. If the number of spans in a trace is ≤ 5 or > 30, we recommend GTrace, benefiting from its Tree-LSTM architecture that handles both shallow and complex hierarchical structures efficiently. For traces with 6–10 spans, the selection depends on the trace depth: TraceVAE performs better for traces with a depth of 3 or less, whereas GTrace is more effective for deeper traces due to its ability to capture long-range dependencies. For traces with spans ranging from 11 to 30, TraceVAE is preferable when the anomaly proportion is $\leq 1\%$ or $> 3\%$; otherwise, GTrace remains the optimal choice.

This hierarchical decision-making framework provides a systematic and practical strategy for selecting the most suitable anomaly detection algorithm across diverse real-world scenarios, effectively accounting for the interplay of multiple dataset characteristics.

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2025.3622122

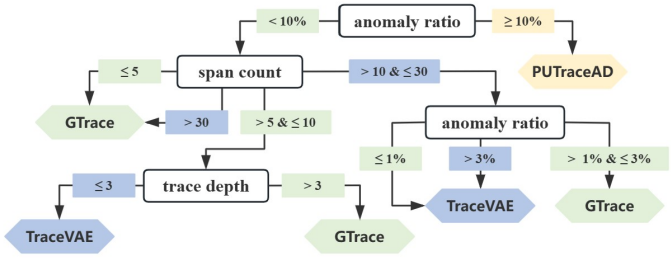


Fig. 18: A decision tree based on multiple datasets' characteristics

V. Threats to Validity

Algorithms. The primary threat to validity in this empirical study lies in the models' configuration. By default, we utilize the hyperparameter settings given in the original publications for the corresponding algorithm and conduct each experiment at least three times. However, when suboptimal results are observed on certain datasets, adjustments are made to improve detection performance. While using default hyperparameters is a practical approach, fine-tuning them to align with the unique characteristics of each dataset may result in better detection outcomes.

Datasets. The external threat to validity stems from the datasets used in the evaluation. Although five distinct datasets were employed to assess algorithm performance, their data features may not comprehensively reflect the wide range of real-world scenarios. Each dataset has unique characteristics and inherent limitations, potentially failing to capture the full diversity and complexity of data encountered in practical applications. To overcome this restriction, we will focus on collecting more datasets from a broader range of scenarios and domains in the future.

VI. Discussion and Future Directions

This section builds upon the empirical findings and provides a deeper exploration of the insights gained from benchmarking trace anomaly detection algorithms. We first discuss the data-driven strategy required for selecting an optimal existing algorithm. Following this, we address the current limitations observed in anomaly detection methodologies and outline promising research directions.

Insights for Existing Algorithm Selection. Our extensive evaluation demonstrates that the most architecturally complex or recently published method is not necessarily the most effective across all scenarios. The effectiveness of an algorithm is deeply intertwined with the characteristics of trace data. Therefore, to select the most appropriate method for a specific scenario, practitioners first need to conduct a thorough analysis of their production trace data, considering factors like span count, service count, trace depth, and anomaly ratio, and then follow our recommended strategy. For example, we recommend PUTraceAD for data with high anomaly ratios. For traces with a large number of spans or a large number of services, GTrace is the preferred choice. Conversely, for trace data with shallow depth, we suggest TraceVAE. Additionally, when higher detection efficiency is the primary goal, GTrace is the most suitable option.

Future Research Directions. Despite the progress made by existing trace anomaly detection algorithms, our empirical results reveal current limitations and future optimization directions.

- **Towards fine-grained span-level anomaly detection.** Many anomaly detection algorithms [14], [16], [20], [25], [34] operate primarily at the trace level, labeling an entire trace as normal or abnormal. This trace-centric approach is insufficient for detailed diagnosis. Future work should focus on optimizing the capability for span-level detection, which is essential for root cause analysis.
- **Building upon the strengths of prior methods.** Researchers should aim to adopt effective design elements from existing work while minimizing architectural constraints that may negatively impact model performance. For example, TraceVAE's use of a fixed span count per trace leads to performance degradation on traces with high span counts, which should be avoided in future designs. Building upon established methodologies while addressing their limitations, future anomaly detection methods can integrate the entropy gap reduction strategies pioneered in TraceVAE. Specifically, this involves implementing Bernoulli & Categorical Scaling for structural anomaly identification, Node Count Normalization for dimensional consistency, and Gaussian Std-Limit thresholding for latency anomalies. These techniques are to be synergistically combined with the hierarchical graph encoding architecture of GTrace, which separates global structure modeling from node-level feature processing through its innovative dispatching layer. Furthermore, the entire system can leverage GTrace's optimized caching strategy that utilizes dynamic programming and LRU-cached trees to enable batched processing of merged subgraphs, thereby enhancing overall efficiency and scalability.

VII. Conclusion

This paper presents a comprehensive empirical study on anomaly detection algorithms used in distributed tracing systems, addressing critical challenges in selecting and applying suitable models for real-world scenarios. By systematically evaluating multiple state-of-the-art algorithms across diverse datasets, we provide an in-depth analysis of their respective strengths and limitations under varying data characteristics, including trace depth, span count, service count, and anomaly ratio. The labeled datasets and algorithm implementation have been made publicly available. Future studies can seamlessly apply these models to new datasets by simply converting the data into our standardized format, eliminating the need for additional format-specific preprocessing for each model. This not only supports experiment reproducibility but also significantly facilitates future research in trace anomaly detection. Furthermore, we offer practical guidance for selecting suitable algorithms based on different data characteristics, bridging the gap between academic research and industrial applications.

To the best of our knowledge, this is the first comprehensive empirical study on trace anomaly detection, providing valuable insights to guide both research and practical deployment.

REFERENCES

[1] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *2015 10th Computing Colombian Conference (10CCC)*, 2015, pp. 583–590.

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham, Switzerland: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.48550/arXiv.1606.04036

[3] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

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

[5] J. Kaldor *et al.*, "Canopy: An end-to-end performance tracing and analysis system," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[6] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, and L. Su, "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1387–1397. [Online]. Available: https://doi.org/10.1145/3368089.3417066

[7] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 623–634.

[8] Jaegertracing.io, "Jaeger," [Online]. Available: https://www.jaegertracing.io/.

[9] Twitter, "Zipkin," [Online]. Available: https://zipkin.io/.

[10] Skywalking.apache.org, "Apache skywalking," [Online]. Available: http://skywalking.apache.org/.

[11] Opentracing.io, "Opentracing," [Online]. Available: http://opentracing.io/.

[12] Z. Xie, C. Pei, W. Li, H. Jiang, L. Su, J. Li, G. Xie, and D. Pei, "From point-wise to group-wise: A fast and accurate microservice trace anomaly detection approach," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1739–1749. [Online]. Available: https://doi.org/10.1145/3611643.3613861

[13] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

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

[15] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*. PMLR, 2015, pp. 1530–1538.

[16] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "CRISP: Critical path analysis of Large-Scale microservice architectures," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 655–672.

[17] Y. Kwon, J. Yoo, Y.-S. Choi, W.-J. Son, D. Lee, and S. Kang, "Efficient learning of non-autoregressive graph variational autoencoders for molecular graph generation," *Journal of Cheminformatics*, vol. 11, no. 1, p. 70, 2019. [Online]. Available: https://doi.org/10.1186/s13321-019-0396-x

[18] J. Mitton, H. M. Senn, K. Wynne, and R. Murray-Smith, "A graph vae and graph transformer approach to generating molecular graphs," *arXiv preprint arXiv:2104.04345*, 2021.

[19] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer, 2018, pp. 412–422.

[20] Z. Xie, H. Xu, W. Chen, W. Li, H. Jiang, L. Su, H. Wang, and D. Pei, "Unsupervised anomaly detection on microservice traces through graph vae," in *Proceedings of the ACM Web Conference 2023*, ser. WWW '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2874–2884. [Online]. Available: https://doi.org/10.1145/3543507.3583215

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[22] M. Ahmed, M. R. Samee, and R. E. Mercer, "Improving tree-lstm with tree attention," in *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, 2019, pp. 247–254.

[23] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1025–1035.

[24] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[25] K. Zhang, C. Zhang, X. Peng, and C. Sha, "Putracead: Trace anomaly detection with partial labels based on gnn and pu learning," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, 2022, pp. 239–250.

[26] R. Kiryo, G. Niu, M. C. du Plessis, and M. Sugiyama, "Positive-unlabeled learning with non-negative risk estimator," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1674–1684.

[27] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5149–5152.

[28] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[29] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

[30] C. Zhang, X. Peng, T. Zhou, C. Sha, Z. Yan, Y. Chen, and H. Yang, "Tracecrl: contrastive representation learning for microservice trace analysis," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1221–1232. [Online]. Available: https://doi.org/10.1145/3540250.3549146

[31] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 701–710. [Online]. Available: https://doi.org/10.1145/2623330.2623732

[32] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[34] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 179–186.

[35] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey,

benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2021.

[36] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Benchmarking microservice systems for software engineering research," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 323–324. [Online]. Available: https://doi.org/10.1145/3183440.3194991

[37] CloudWise, "GAIA," [Online]. Available: https://github.com/CloudWise-OpenSource/GAIA-DataSet, 2021.

[38] NetManAIOps, "AIOps Challenge 2020 Data," [Online]. Available: https://github.com/NetManAIOps/AIOps-Challenge-2020-Data, 2020.

[39] Google Cloud Platform, "Online Boutique," [Online]. Available: https://github.com/GoogleCloudPlatform/microservices-demo.

[40] S. Zhang, Z. Pan, H. Liu, P. Jin, Y. Sun, Q. Ouyang, J. Wang, X. Jia, Y. Zhang, H. Yang, Y. Zou, and D. Pei, "Efficient and robust trace anomaly detection for large-scale microservice systems," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023, pp. 69–79.

[41] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, 1st ed. New York: Chapman and Hall/CRC, 1984, eBook Published 19 October 2017.

Yongqian Sun received the B.S. degree in statistical specialty from Northwestern Polytechnical University, Xi'an, China, in 2012, and Ph.D. in computer science from Tsinghua University, Beijing, China, in 2018. He is currently an assistant professor in the College of Software, Nankai University, Tianjin, China. His research interests include anomaly detection and root cause localization in service management.

Minyi Shao received the B.E. degree in software engineering from Nankai University, Tianjin, China, in 2023. She is currently working toward the M.E. degree with the Department of Software Engineering, Nankai University. Her research interests include AIOps and anomaly detection in service management.

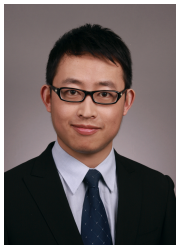
Xiaohui Nie received the B.E. degree in computer science and technology from Jilin University, Jilin, China, in 2013, and Ph.D. in computer science from Tsinghua University, Beijing, China, in 2019. He is currently an associate professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), Beijing, China. His research interests include AIOps and Internet Security.

Kaiwen Yang received the B.E. degree in software engineering from Nankai University, Tianjin, China, in 2025. She is currently working toward the M.E. degree with the Department of Software Engineering, Nankai University. Her research interests include AIOps and LLM.

Xingda Li He is currently working toward the B.E. degree with the Department of Software Engineering, Nankai University. His research interests include anomaly detection and root cause localization.

Bowen Hao received the B.E. degree in software engineering from Nankai University, Tianjin, China, in 2023. He is currently working toward M.E. degree with the Department of Software Engineering, Nankai University. His research interests include LLM for AIOps and software engineering.

Shenglin Zhang received B.S. in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012 and Ph.D. in computer science from Tsinghua University, Beijing, China, in 2017. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His current research interests include failure detection, diagnosis and prediction for service management.



Changhua Pei received the B.E. and Ph.D. in computer science from the Department of Computer Science and Technology, Tsinghua University in 2012 and 2017, respectively. He is currently an associate professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), Beijing, China. His research interests include AIOps and AI for Networking.

Dongbiao He received the PhD degree in computer science and technology from Tsinghua University, Beijing, China in 2019. His research interests include networking and edge computing systems. He received his B.E. degree from Jilin University, China, in 2013. He is an associate professor at the Computer Network Information Center of the Chinese Academy of Sciences.

Yanbiao Li received the B.S. degree in mathematics from Hunan University, Changsha, China, in 2009, and Ph.D. in computer science from Hunan University, Changsha, China, in 2016. He is currently a professor with the Computer Network Information Center (CNIC), Chinese Academy of Science (CAS), Beijing, China. His research interests include efficient and reliable routing, and satellite network.

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