

# Twin Graph-based Anomaly Detection via Attentive Multi-Modal Learning for Microservice System

Jun Huang<sup>1</sup>, Yang Yang<sup>1</sup>, Hang Yu<sup>2\*</sup>, Jianguo Li<sup>2\*</sup>, Xiao Zheng<sup>1</sup>

<sup>1</sup> School of Computer Science and Technology, Anhui University of Technology, Maanshan, China

<sup>2</sup> Ant Group, Hangzhou, China

{huangjun.cs,young978,xzheng}@ahut.edu.cn, {hyu.hugo,lijg.zero}@antgroup.com

**Abstract**—Microservice architecture has sprung up over recent years for managing enterprise applications, due to its ability to independently deploy and scale services. Despite its benefits, ensuring the reliability and safety of a microservice system remains highly challenging. Existing anomaly detection algorithms based on a single data modality (i.e., metrics, logs, or traces) fail to fully account for the complex correlations and interactions between different modalities, leading to false negatives and false alarms, whereas incorporating more data modalities can offer opportunities for further performance gain. As a fresh attempt, we propose in this paper a semi-supervised graph-based anomaly detection method, MSTGAD, which seamlessly integrates all available data modalities via attentive multi-modal learning. First, we extract and normalize features from the three modalities, and further integrate them using a graph, namely MST (microservice system twin) graph, where each node represents a service instance and the edge indicates the scheduling relationship between different service instances. The MST graph provides a virtual representation of the status and scheduling relationships among service instances of a real-world microservice system. Second, we construct a transformer-based neural network with both spatial and temporal attention mechanisms to model the inter-correlations between different modalities and temporal dependencies between the data points. This enables us to detect anomalies automatically and accurately in real-time. Extensive experiments on two real-world datasets verify the effectiveness of our proposed MSTGAD method, achieving competitive performance against state-of-the-art approaches, with a 0.961  $F_1$ -score and an average increase of 4.85%. The source code of MSTGAD is publicly available at [https://github.com/alipay/microservice\\_system\\_twin\\_graph\\_based\\_anomaly\\_detection](https://github.com/alipay/microservice_system_twin_graph_based_anomaly_detection).

**Index Terms**—anomaly detection, multi-modal learning, system twin graph

## I. INTRODUCTION

The microservice architecture has gained popularity in recent years as a method of developing applications. This methodology involves breaking down a single application into a suite of small services, each running in its own process and communicating via lightweight mechanisms [1]. One of the key advantages of this architecture is that services can be independently deployed and scaled. As a result, many industries have adopted this architecture to manage their enterprise applications. However, ensuring the reliability and safety of a Microservice system can be challenging. Traditional approaches rely on manual inspection, which is impractical for

large-scale distributed systems consisting of numerous services running on different machines.

Fortunately, various types of data can be monitored in a microservice system, including service/machine metrics, logs, traces, etc. These data play a crucial role in ensuring the reliability and safety of the system. For example, metrics are real-valued time-series providing information on the status of services or machines and the associated requirements that need to be met during runtime operations. Logs are semi-structured text messages printed by logging statements to record the system's run-time status. Traces are hierarchical descriptions of the modules and services called upon to fulfill a user request. They are usually recorded with the service name or category and the time duration of each module. By leveraging the monitored data of a microservice system, many works [2], [3] have successfully replaced manual inspection with automated anomaly detection algorithms, such as the log-based approaches [4]–[8], metric-based approaches [9]–[15] and trace-based approaches [16]–[22].

Despite their effectiveness, they are often based on a single data modality, which can incur both false negatives and false alarms. Practical abnormal patterns vary across data modalities, with some only being evident in specific modalities. Failing to account for these modalities can lead to false negatives, which may further cause system outage and substantial financial loss. For instance, consider an anomaly related to an API version, where a service request does not receive a timely response or is not properly parsed. Such an anomaly may not be detectable through metrics data alone, as there might not be any noticeable increase in resource consumption for the executed service container. However, it is feasible to detect this anomaly by leveraging trace data. On the other hand, different modalities can complement each other to filter out false alarms. For example, when configuring new applications, which is a normal change, metric-based approaches may interpret increased CPU and memory usage as an anomaly, resulting in false alarms. However, by incorporating logs and traces, such false alarms can be reduced. It is essential to acknowledge that reducing the number of false alarms is of significant importance in real-world scenarios, as a high volume of false alarms can obscure true positive detections. In light of these challenges associated with single-modal anomaly detection methods, several works have proposed approaches based on

\* Corresponding authors

This work was sponsored by CCF-AFSG research fund.

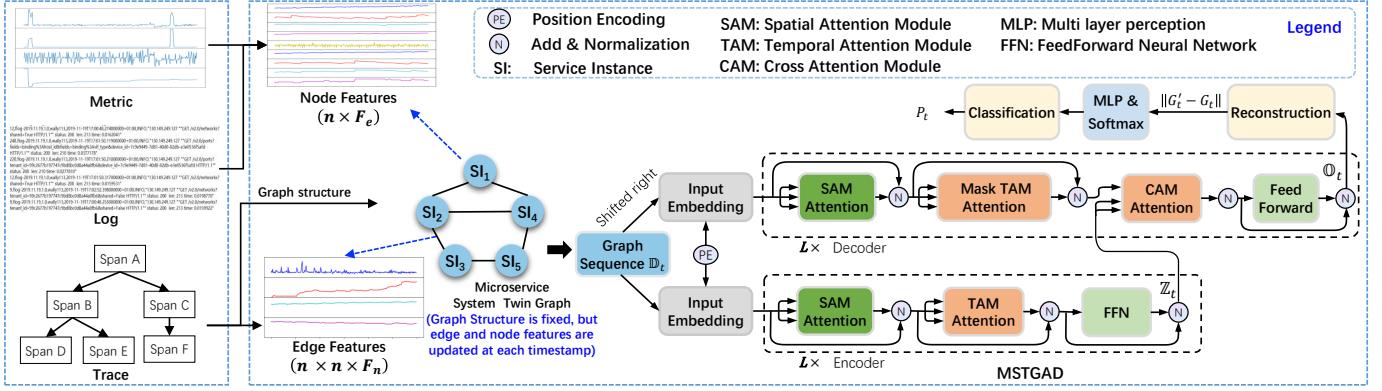


Fig. 1. Framework of the proposed method MSTGAD

multi-modal data, such as DeepTraLog [1], SCWarn [23], and HADES [24]. However, they use at most two modalities and do not fully consider the complex correlations between different data modalities.

Another problem with the existing works is that they are typically constructed in an unsupervised way and fail to exploit the labeling information. However, in real-world scenarios, anomaly information is usually available [25], particularly when the anomaly detection algorithm fails to identify a system failure and further causes a system outage. Utilizing this labeling information can further enhance the algorithm's performance, but it cannot be incorporated into unsupervised learning methods. On the other hand, unlabeled data is more abundant and readily available. It is beneficial to consider a semi-supervised approach that combines both labeled and unlabeled data.

The key to remedying the above two issues lies in the extraction of useful information from heterogeneous multi-source data involved in Microservice systems and the construction of a semi-supervised model that can accurately identify anomalies. To move forward to this goal, two major challenges must be overcome.

1) *How to unite and represent heterogeneous multi-source data and model the complex correlations and interactions between them?* In a Microservice system, service instances are configured in containers and called upon by user requests. Concurrently, traces are generated, and logs and metrics are monitored and recorded for these service instances and machines by timestamp. Therefore, it is natural to represent the multi-modal data using a graph to reflect the state of a Microservice system. The nodes represent service instances, and the edges correspond to their scheduling relationships. Since diverse modalities are integrated into a graph, graph machine learning techniques can be utilized to model complex correlations and interactions between them, such as the message-passing mechanism.

2) *How to construct a semi-supervised anomaly detection model?* In the IT industry, apart from the abundant unlabeled data, some supervised information can be collected, as some anomalies can be labeled and reported by users. Therefore, the

supervised information can be utilized to create an efficient semi-supervised anomaly detection model.

In this paper, we present a novel approach to address the challenges of anomaly detection in microservice systems. Our proposed method, MSTGAD, is a microservice system twin (MST) graph-based anomaly detection method that uses attentive multi-modal learning. It aims to automatically and accurately detect anomalies by leveraging metrics, logs, and traces simultaneously. The framework of MSTGAD is presented in Fig. 1. We first extract and normalize the features of metrics, logs, and traces. Then, we propose to integrate the three modalities together via a graph structure, where each node represents a service instance, and the edge indicates the scheduling relationship between different service instances obtained from the traces. Since logs and metrics are recorded according to service instances, we use the features of metrics and logs to represent the node features. Traces describe the scheduling relationships between service instances within a microservice system, which are extracted and used to represent the features of edges. Similar to a digital twin system, for each timestamp, the information about a microservice system can be reflected by the graph which can be considered a twin of the physical microservice system and virtually represent the status and scheduling relationships among service instances of a real-world microservice system. Therefore, we name it a microservice system twin graph, i.e., MST graph. It can be used to design effective anomaly detection models to improve the reliability and safety of its physical counterpart.

Next, we construct a transformer-based neural network that incorporates both spatial and temporal attention mechanisms based on the MST graphs. The spatial attention module (SAM) is used to model the inter-correlation between different modalities, while the temporal attention module (TAM) is used to model the temporal dependency between data points in a sliding window. Our proposed method is evaluated on two real-world datasets, and the results demonstrate its effectiveness in anomaly detection. To sum up, our contributions include:

- We perform the first empirical study of semi-supervised anomaly detection via multi-modal learning from metrics, logs, and traces simultaneously.

- We propose to integrate and represent the three types of data together via a microservice system twin (MST) graph which can be taken as a virtual representation of a real-world microservice system.
- We develop a transformer-based neural network with both spatial and temporal attention mechanisms based on the MST graphs to model the inter-correlation between different modalities and the temporal dependency among the data points (i.e., the MST graphs) of a sliding window.
- Extensive experiments on two real-world datasets show that the proposed MSTGAD model achieves competitive performance against all compared approaches, achieving 0.961  $F_1$ -score with an average increase of 4.85%. Additionally, ablation studies further validate the effectiveness of each design in our model.

## II. RELATED WORK

In the past decades, various techniques [2], [3] have been proposed for anomaly detection based on metrics (Key Performance Indicators), logs, and traces.

Log data records the system state and significant events at various critical timestamps, which is an important and valuable resource for online monitoring, anomaly detection, and root cause analysis. There have been a lot of studies on log-based anomaly detection. These logs-based approaches first parse the unstructured logs into structured representations through log parsing approaches including Drain [26], AEL [27], IPLoM [28] and Spell [29], and SemParser [30], then build the anomaly detection models. Deeplog [4] adopts LSTM to model a system log as a natural language sequence, which can automatically learn log patterns from normal execution and detect anomalies when log patterns deviate from the model trained from log data under normal execution, and also it can be incrementally updated in an online manner. LogAnomaly [5] is an end-to-end framework also using the LSTM network to automatically detect sequential and quantitative anomalies simultaneously. To solve log data containing previously unseen log events or log sequences, LogRobust [31] is proposed to extract semantic information of log events and represent them as semantic vectors. It then detects anomalies by utilizing an attention-based Bi-LSTM model, which has the ability to capture the contextual information in the log sequences and automatically learn the importance of different log events. PLELog [7] is a semi-supervised log-based anomaly detection approach via probabilistic label estimation based on an attention-based GRU neural network. To overcome the errors caused by log parsing, NeuralLog [6] is proposed to extract the semantic meaning of raw log messages and represent them as semantic vectors. Existing log anomaly detection approaches treat a log as a sequence of events and cannot handle microservice logs that are distributed in a large number of services with complex interactions.

Inspired by kernel-based one-class classification [9], Deep SVDD [10] is proposed to train a neural network by minimizing the volume of a hypersphere that encloses the network

representations of the data. Tuli *et al.* propose a transformer-based anomaly detection model TranAD [15] which uses self-conditioning and an adversarial training process. The DAGMM [11] method uses a deep autoencoding Gaussian mixture model for dimension reduction in the feature space and recurrent networks for temporal modeling. The USAD [12] method uses an autoencoder with two decoders with an adversarial game-style training framework. This is one of the first works that focus on low overheads by using a simple autoencoder and can achieve a several-fold reduction in training times compared to the prior art. The GDN [32] approach learns a graph of relationships between data modes and uses attention-based forecasting and deviation scoring to output anomaly scores. In microservice systems, traces are widely used in anomaly detection and root cause analysis. Seer [17] is an online cloud performance debugging system that leverages deep learning and a massive amount of tracing data to learn spatial and temporal patterns to detect QoS violations. TraceAnomaly [18] is an unsupervised anomaly detection approach of microservice through Service-Level Deep Bayesian Networks. GMTA [19] is a graph-based approach for architecture understanding and problem diagnosis of microservice trace analysis. Liu *et al.* propose a high-efficient root cause localization approach MicroHECL [20] which dynamically constructs a service call graph and analyzes possible anomaly propagation chains by traversing the graph along anomalous service calls.

Although the above approaches have achieved great performance in anomaly detection. While they are mainly constructed based on single-modal data, such as logs, metrics, or traces respectively. However, as a single source of information is often insufficient to depict the status of a microservice system precisely, existing methods could produce many false alarms and may omit some true positives [24]. Recently, several studies have been done based on multi-modal data to further improve the performance of anomaly detection.

DeepTraLog [1] utilizes a unified graph representation to describe the complex structure of a trace together with log events embedded in the structure, and trains a GGNNs-based deep SVDD [10] model and detects anomalies in new traces and the corresponding logs. SCWarn [23] is proposed for online service systems to identify the bad software changes via multimodal learning from heterogeneous multi-source data. In SCWarn, the temporal dependency in each time series is captured by the LSTM model, and the inter-correlations among multi-source data are encoded via multimodal fusion. Similar to SCWarn, Liu *et al.* propose HADES [24] which employs a hierarchical architecture to learn a global representation of the system status by fusing log semantics and metric patterns, and a novel cross-modal attention module to capture discriminative features and meaningful interactions from multi-modal data.

In parallel to our research, several other endeavors have emerged to explore the combined utilization of logs, metrics, and traces as modalities, such as [25], [33], [34]. Specifically, Eadro [25] presents an end-to-end framework that seamlessly integrates anomaly detection and root cause localization based

on multi-source data, specifically designed for troubleshooting purposes. This framework adeptly models both intra-service behaviors and inter-service interactions, exploiting shared knowledge through multi-task learning. AnoFusion [33] offers an unsupervised failure detection approach that efficiently identifies instance failures through multimodal data. It employs a Graph Transformer Network (GTN) to effectively capture correlations within the heterogeneous multimodal data. Additionally, it seamlessly integrates a Graph Attention Network (GAT) with a Gated Recurrent Unit (GRU) to address the challenges introduced by dynamically changing multimodal data. DiagFusion [34], on the other hand, employs embedding techniques and data augmentation to accurately represent multimodal data of service instances. By combining deployment data and traces, it constructs a dependency graph and subsequently employs a graph neural network to localize the root cause instance and determine the type of failure. It is important to note that all the aforementioned approaches are either fully supervised or unsupervised. In contrast, our proposed method is semi-supervised, which effectively leverages both labeled and unlabeled data to enhance model performance. Furthermore, the existing works consider all three modalities as node features in their graph neural networks for capturing spatial dependence, but our approach treats traces as edge features. Additionally, Eadro [25] and AnoFusion [33] employs CNN and GRU to capture the temporal dependence, whereas our proposed method harnesses the power of Transformers. Transformers have been proven to outperform CNNs and GRUs for time series modeling [35], [36]. Lastly, it is worth mentioning that AnoFusion primarily focuses on anomaly detection, DiagFusion on root cause analysis, while Eadro tackles both anomaly detection and root cause localization. In line with our work, our objective here is anomaly detection, akin to AnoFusion, but distinct from DiagFusion and Eadro.

### III. PRELIMINARIES

In our proposed model, MSTGAD, we use various attention mechanisms to capture the dependencies between different parts of the input data. Before delving into the details of our model, we first introduce different attention mechanisms used in the following sections.

#### A. Scaled Dot-Product Attention

The fundamental concept of attention is to determine a series of weights that express the significance or pertinence of various elements within the input information, depending on a particular inquiry. These weights can be utilized to calculate a weighted sum of the input data, which may then be utilized as an input for the subsequent layer of the model. Specifically, given the query  $\mathbf{Q} \in \mathbb{R}^{L_Q \times d_k}$ , key  $\mathbf{K} \in \mathbb{R}^{L_K \times d_k}$ , and value  $\mathbf{V} \in \mathbb{R}^{L_V \times d_v}$ , a scaled-dot product attention [37] of the three matrices can be defined as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}, \quad (1)$$

where  $\text{Softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d_k}) \in \mathbb{R}^{L_Q \times L_K}$  can be considered as a similarity or attention score matrix, and  $d_k$  is the number of

features. By selectively focusing on different parts of the input data, attention can help the model to better capture long-term dependencies and improve its overall performance.

#### B. Multi-head Attention

Multihead attention is an extension of the above standard attention mechanism that allows the model to jointly attend to information from different representation subspaces. The basic idea behind multi-head attention is to split the input queries, keys, and values into multiple heads, and to compute separate attention scores for each head. The outputs from each head are then concatenated and passed through a linear projection to generate the final output. By using multiple heads, the model can learn to attend to different aspects of the input data, and the final output is a combination of the outputs from multiple heads. Mathematically, the multi-head attention function can be expressed as follows:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(H_1, \dots, H_h) \mathbf{W}^O, \quad (2)$$

where each  $H_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$  is single-head attention, and  $h$  is the number heads. The projections  $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$ ,  $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$ ,  $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$  and  $\mathbf{W}^O \in \mathbb{R}^{hdv \times d_{model}}$  are learnable parameters.

#### C. Graph Attention

Graph attention is also an extension of the scaled dot-product attention mechanism that can be used to process graph-structured data. Unlike traditional attention mechanisms where the input is a sequence of vectors, in graph attention, the input is a graph where each node represents a vector. Each node is associated with “query”, “key”, and “value” vectors, which are used to compute attention scores between pairs of nodes. Specifically, For a graph  $G = (V, A, E)$ ,  $V$  is the set of nodes,  $E$  is the set of edges and  $A$  is the adjacency matrix. A Graph Attention Network (GAT) [38]–[40] layer updates each node representation by aggregating the representations of its neighboring nodes. Each node  $v_i$  can be updated by

$$v'_i = \sum_{u \in \mathbb{N}(i)} \alpha_{i,u} \mathbf{W} v_u, \quad (3)$$

where  $\mathbf{W} \in \mathbb{R}^{d_{model} \times F_v}$  is a learnable parameter, and the graph attention weight  $\alpha_{ij}$  is defined as

$$\alpha_{i,j} = \frac{e(v_i, v_j, e_{ij})}{\sum_{u \in \mathbb{N}(i)} e(v_i, v_u, e_{iu})}, \quad (4)$$

where  $v_i \in \mathbb{R}^{F_v}$  and  $v_j \in \mathbb{R}^{F_v}$  are the  $i$ -th and  $j$ -th nodes’ feature representation, and  $e_{ij}$  is the edge representation between the two nodes. Different from standard GAT, the edges are also used to calculate the attention weight.  $\mathbb{N}(i)$  includes the  $i$ -th node and its directed connected neighbors.  $e(v_i, v_j, e_{ij})$  indicates the importance of the features of node  $j$  to node  $i$ , which is defined as

$$e(v_i, v_j, e_{ij}) = \beta^\top \text{LeakyReLU}(\mathbf{W}[v_i \| v_j \| e_{ij}]), \quad (5)$$

where  $\beta \in \mathbb{R}^{3d_{model}}$  and  $\mathbf{W} \in \mathbb{R}^{3d_{model} \times (2F_v + F_e)}$  are learnable parameters, and  $\|$  denotes vector concatenation.

Equivalently, the above formulations can be written in the matrix form as

$$\text{SpatialAtt}(\mathbf{V}, \mathbf{V}, \mathbf{E}) = \text{Softmax}(e(\mathbf{V}, \mathbf{V}, \mathbf{E}))\mathbf{V}\mathbf{W}^\top, \quad (6)$$

where  $e(\mathbf{V}, \mathbf{V}, \mathbf{E}) = \beta^\top \text{LeakyReLU}(\mathbf{W}[\mathbf{V}\|\mathbf{V}\|\mathbf{E}])$ .

#### IV. APPROACH

The proposed method, MSTGAD, aims to automatically and accurately detect anomalies by leveraging multi-modal learning from metrics, logs, and traces simultaneously. The approach takes all three modalities as input and trains a graph-based deep learning model based on an enhanced transformer structure. The framework of MSTGAD is presented in Fig. 1, which is mainly composed of two stages, i.e., multi-modal data fusion and representation, and anomaly detection. In the first stage, the features of metrics, logs, and traces are extracted and integrated using an MST graph. The graph represents service instances as nodes and scheduling relationships between different instances as edges. In the second stage, a transformer-based neural network is constructed with spatial and temporal attention based on the MST graphs, for the sake of anomaly detection. The network can model the inter-correlation between different modalities and the temporal dependency of data points in a sliding window. Next, we elaborate on these two stages.

##### A. Pre-Processing and MST graph construction

1) *Metric Pre-processing*: Metrics provide valuable information regarding the status of services and machines, such as response time, the number of threads, and CPU and memory usage. In this paper, we focus on multivariate time-series data, denoted as  $\mathbb{T}_{\text{Metric}} = \{\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_T\}$ , which represents a sequence of timestamped observations of size  $T$ . Each data point  $\mathbf{M}_t \in \mathbb{R}^{N \times F_m}$  is collected at a specific timestamp  $t$  for  $N$  service instances and machines, where  $F_m$  is the number of metrics. We further normalize the data and convert it to time-series windows for both training and testing purposes. Specifically, each data point  $\mathbf{M}_t$  is normalized using min-max normalization, as follow:

$$\mathbf{M}_t = \frac{\mathbf{M}_t - \min(\mathbb{T}_{\text{Metric}})}{\max(\mathbb{T}_{\text{Metric}}) - \min(\mathbb{T}_{\text{Metric}}) + \epsilon}, \quad (7)$$

where  $\max(\mathbb{T}_{\text{Metric}}) \in \mathbb{R}^{F_m}$  and  $\min(\mathbb{T}_{\text{Metric}}) \in \mathbb{R}^{F_m}$  are the maximum and minimum vectors in the training time-series, and  $\epsilon$  is a small constant vector used to avoid zero-division.

In practice, the number of metrics might be huge. To cope with the challenge of real-time data collection, we can remove those metrics whose variances are very low, since anomalies are reflected by metrics with fluctuation.

2) *Log Parsing*: Logs are text messages that are semi-structured and record system states and significant events at critical timestamps. In this step, we aim to convert unstructured log messages into structured log events. To achieve this, we use the widely adopted parser, Drain3 [26], similar to other studies on anomaly detection [1], [23], [24]. Drain3 can parse logs in a streaming and timely manner with high parsing accuracy and efficiency. After parsing, timestamp, service instance ID,

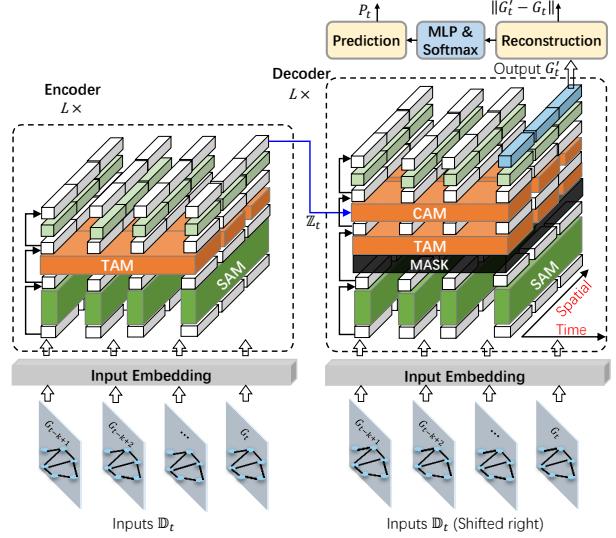


Fig. 2. Framework of MSTGAD

and log template index are attached to each log event for further analysis. To enable combined analysis with metrics, we count the occurrences of each template in each timestamp, creating log representations consistent with the formulation of metrics. This yields a timestamped log sequence, denoted as  $\mathbb{T}_{\text{Log}} = \{\mathbf{L}_1, \mathbf{L}_2, \dots, \mathbf{L}_T\}$  of observation of size  $T$ , where each data point  $\mathbf{L}_t \in \mathbb{R}^{N \times F_l}$ , and  $F_l$  is the number log templates.

Similar to the normalization of metrics, each data point  $\mathbf{L}_t$  is also normalized via the min-max normalization as

$$\mathbf{L}_t = \frac{\mathbf{L}_t - \min(\mathbb{T}_{\text{Log}})}{\max(\mathbb{T}_{\text{Log}}) - \min(\mathbb{T}_{\text{Log}}) + \epsilon}, \quad (8)$$

where  $\max(\mathbb{T}_{\text{Log}}) \in \mathbb{R}^{F_l}$  and  $\min(\mathbb{T}_{\text{Log}}) \in \mathbb{R}^{F_l}$  are the maximum and minimum vectors in the training time-series.

Serializing logs can potentially introduce challenges associated with long-tail distributions. However, it is possible to mitigate this issue to some extent by leveraging the complementary information from other modalities. For instance, although the occurrence of out-of-memory anomalies is rare and may be affected by the long-tail problem, such anomalies can also be detected through the analysis of metric data. Furthermore, the performance of our method can be further enhanced by incorporating more advanced log parsing techniques [41], including semantic-based approaches [6], [30].

3) *Trace Parsing*: Each trace in a microservice system depicts the execution processes of user requests. These processes are known as spans, and they possess valuable attributes such as trace ID, request type, span ID, father span ID, service instance ID, start time, and duration time. In the spirit of TraceAnomaly [18], we represent spans as time series. Within a sliding window, we aggregate the total duration time of spans sharing the same request type, service instance launcher, and service instance receiver for each timestamp. It is important to note that unfinished spans are omitted from the window. Consequently, we obtain a timestamped trace sequence  $\mathbb{T}_{\text{Trace}} = \{\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_T\}$  of size  $T$ , and each data

point  $\mathbf{S}_t \in \mathbb{R}^{N \times N \times F_s}$ , where  $N$  indicates the number of service instances of a microservice system, and  $F_s$  indicates the dimension of the extracted features of traces. Since not all service instances are interconnected, we use an approximate normalization technique to differentiate between small spans and no spans. Specifically, we define it as:

$$\mathbf{S}_t = \frac{\mathbf{s}_t}{\text{mean}(\mathbb{T}_{\text{Trace}})}, \quad (9)$$

where  $\text{mean}(\mathbb{T}_{\text{Trace}}) \in \mathbb{R}^{F_s}$  is the average of  $\mathbb{T}_{\text{Trace}}$

In reality, if tremendous traces are generated, state-of-the-art head-based and tail-based sampling techniques can be applied to reduce the processing and storage costs.

4) *Graph Construction*: After obtaining  $\mathbb{T}_{\text{Metrics}}$ ,  $\mathbb{T}_{\text{Log}}$ , and  $\mathbb{T}_{\text{Trace}}$ , we can integrate these three modalities together by constructing an MST graph to virtually represent the status and scheduling relationships among service instances of a real-world microservice system. In an MST graph, nodes represent service instances, and edges represent scheduling relationships between different instances.

Formally, for each timestamp  $t$ , the MST graph is defined as  $G_t = \langle V_t, A_t, E_t \rangle$ , where  $V_t$  is the set of nodes and the corresponding node features are defined as  $\mathbf{V}_t = \mathbf{M}_t \parallel \mathbf{L}_t$ , i.e., the concatenation of metric and log features, and  $\parallel$  indicates the concatenation operator.  $E_t$  is the set of edges, and  $\mathbf{E}_t = \mathbf{S}_t$  is used to represent the features of edges, and  $A_t$  is the adjacent matrix. Each element  $a_{ij}$  of  $A_t$  is defined as

$$a_{ij} = \begin{cases} 1, & \text{if their is scheduling between the } i\text{-th} \\ & \text{and } j\text{-th service instances} \\ 0, & \text{otherwise} \end{cases}.$$

In reality, microservice systems undergo constant changes, such as services scaling up or down. These changes can be accommodated by updating the adjacency matrix when services are added or removed. However, it is important to note that the parameters of the Graph ATtention network (GAT) used in the proposed model, which will be introduced in the following sections, can remain unchanged. This is because the GAT operates on the graph structure itself, rather than being dependent on the specific services present in the system. As a result, the model can effectively adapt to changes in the microservice system without the need for retraining or modifying the GAT parameters. Furthermore, by considering the preprocessed metrics and logs as node features, and the traces as edge features, we can characterize the dynamic behavior of traces by adjusting the edge features accordingly, without modifying the underlying graph structure. This property allows for efficient representation of changes in the scheduling relationships among service instances. Consequently, when a service is removed from the system, we can alternatively set the corresponding node and edge features to zero without removing the service from the graph.

In summary, the real-time status and scheduling relationships among service instances of a real-world microservice system can be virtually represented by the MST graph, which can be used to design an effective anomaly detection model to improve the reliability and safety of its physical counterpart.

To model the temporal dependence of a data point at timestamp  $t$ , we consider a sliding window of length  $k$  as

$$\mathbb{D}_t = \{G_{t-k+1}, \dots, G_t\}. \quad (10)$$

For each sliding window  $\mathbb{D}_t$ ,  $y_t \in \mathbb{R}^{1 \times N}$  is the ground truth label vector, and  $y_{ti} \in \{-1, 0, 1\}$  is the ground truth label for the  $i$ -th service instance in  $G_t$ . The numbers 1 and 0 indicate abnormal and normal data respectively, and  $-1$  indicates the corresponding label is unknown for the unlabelled data.

### B. Transformer with Spatial and Temporal Attention

MSTGAD is constructed based on the transformer [37] architecture for anomaly detection via attentive multi-modal learning. Fig. 2 shows the architecture of the neural network used in MSTGAD, which consists of encoding (left side) and decoding (right side) steps with several attention modules.

In the encoding step, the encoder takes the input sequence  $\mathbb{D}_t$  with  $k$  MST graphs (cf. Eq (10)) and converts it to a hidden state  $\mathbb{Z}_t$ . To accomplish this,  $\mathbb{D}_t$  is first subjected to the Input Embedding operation, a linear layer that maps multi-source features to appropriate dimensions for multi-head attention. We also add position encoding [37] of each data point on the time dimension to provide MSTGAD with local position information. In order to model the inter-correlation between different modalities, we propose the spatial attention module (SAM), which is based on GAT [39]. Moreover, we propose the temporal attention module (TAM) to model the temporal dependency between different data points in the sliding window  $\mathbb{D}_t$ . Specifically, the workflow of each layer of the encoder can be defined by a series of operations:

$$\begin{aligned} EI_1 &= \text{Norm}(EI_0 + \text{SAM}(EI_0)), \\ EI_2 &= \text{Norm}(EI_1 + \text{TAM}(EI_1)), \\ \mathbb{Z}_t &= \text{Norm}(EI_2 + \text{FFN}(EI_2)), \end{aligned}$$

where  $\text{Norm}(\cdot)$  is the normalization operation, and FFN is the Feed-forward neural network.

In the decoding step,  $\mathbb{D}_{t-1}$  will be used to predict the representation of  $\mathbb{D}_t$ . To achieve this goal,  $\mathbb{D}_t$  is shifted one timestamp to the right and padded with zero at the first timestamp as the input of decoding. After that, the input sequence  $\mathbb{D}_{t-1}$  is also converted to  $DI_0$  via the Input Embedding operation. The spatial attention module (SAM) is then applied to model the inter-correlation between different modalities. Unlike the encoding step, TAM is run with a causal mask to ensure that each  $G_t$  is updated or predicted only based on its former data points. In addition, we propose a cross-attention module (CAM) to update  $\mathbb{D}_t$  with the guidance of  $\mathbb{Z}_t$ . Correspondingly, the workflow of each layer of the decoder can be defined as the following operations:

$$\begin{aligned} DI_1 &= \text{Norm}(DI_0 + \text{SAM}(DI_0)), \\ DI_2 &= \text{Norm}(DI_1 + \text{Mask}(\text{TAM}(DI_1))), \\ DI_3 &= \text{Norm}(DI_2 + \text{CAM}(DI_2, \mathbb{Z}_t)), \\ \mathbb{O}_t &= \text{Norm}(DI_3 + \text{FFN}(DI_3)). \end{aligned}$$

Finally, we detect anomalies based on the reconstruction errors between  $\mathbb{D}_t$  and  $\mathbb{O}_t$ . The following subsections provide

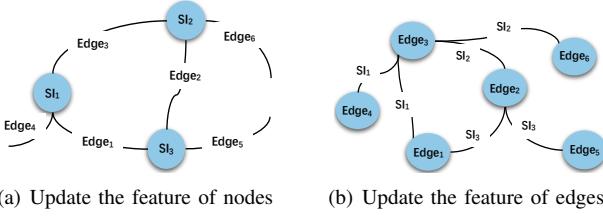


Fig. 3. Spatial Attention

further details about the SAM, TAM, and CAM attention modules.  $\mathbb{D}_t$  is the input of MSTGAD, and  $\mathbf{M}_{t-k+1:t}$ ,  $\mathbf{L}_{t-k+1:t}$ , and  $\mathbf{S}_{t-k+1:t}$  are used to represent the features of metrics, logs, and traces of  $\mathbb{D}_t$  respectively for convenience.

1) *Spatial Attention Module*: As mentioned in the introduction, metrics, logs, and traces are different ways of viewing the status of a microservice system. By exploiting the complementary information from these different modalities, we can boost the generalization ability of anomaly detection algorithms. To achieve this, we propose a spatial attention module (SAM) that models the inter-correlation between modalities using two layers of Graph Attention Network (GAT) [39].

SAM allows for the updating of node and edge features by incorporating information from their neighbors through message passing. Fig. 3 provides a toy example, where SI stands for service instance. In Fig. 3(a), when updating the representation of a node, the information from its directed neighbors will be passed to it with an attention weight along the edges. Similarly, the feature of edges can be updated by exchanging the roles of nodes and edges in an MST graph (see Fig. 3(b)). As a result, the representation of each modality can be updated and improved by absorbing the complementary information from other modalities.

In this paper, for an input MST graph  $G_t = (V_t, A_t, E_t)$ , we apply multi-head attention (see Eq.(2)), and then the features of nodes of service instances of  $G_t$  can be updated by

$$\mathbf{V}'_t = \text{MultiHead}(\mathbf{V}_t, \mathbf{V}_t, \mathbf{E}_t), \quad (11)$$

where each head  $H_i = \text{SpatialAtt}(\mathbf{V}_t, \mathbf{V}_t, \mathbf{E}_t)$  (see Eq.(6)). Unlike standard GAT [39], our approach incorporates edges in the calculation of the attention weight. The reasoning behind this is that edge features represent the features of traces, and excluding them would result in the loss of valuable trace information. By exchanging the roles of nodes and edges in an MST graph, the features  $\mathbf{E}_t$  of edges of service scheduling in each  $G_t$  can be updated in the same manner as

$$\mathbf{E}'_t = \text{MultiHead}(\mathbf{E}_t, \mathbf{E}_t, \mathbf{V}'_t), \quad (12)$$

where each head  $H_i = \text{SpatialAtt}(\mathbf{E}_t, \mathbf{E}_t, \mathbf{V}'_t)$ .

The SAM module consists of two layers of GAT which alternate in updating the features of nodes and edges. This enables the module to efficiently model the complex inter-correlation among metrics, logs, and traces by learning the representation of each MST graph  $G_t$  in a sliding window  $\mathbb{D}_t$ . Through the thorough integration and fusion of complementary information among the three modalities, the SAM

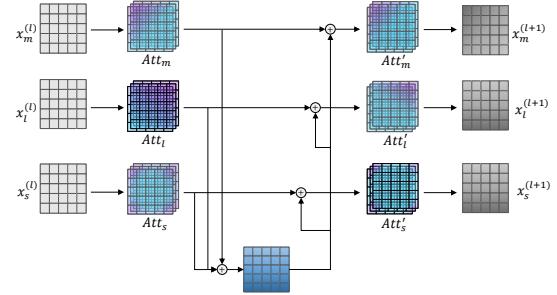


Fig. 4. Framework of the Temporal Attention

module produces an enriched representation  $G'_t$ . Thus, for a sliding window, we will update all the MST graphs to get a new representation of  $\mathbb{D}_t$  as

$$\mathbb{D}'_t = (G'_{t-k+1}, \dots, G'_t) = \text{SAM}(\mathbb{D}_t).$$

2) *Temporal Attention Module*: The above SAM module is designed to model the complex inter-correlation among different modalities at the local graph level across space. In this section, we further propose the temporal attention module (TAM) to capture the complex inter-correlation among different modalities at the global sequence level across time. Notice that in a microservice system, multi-source monitored data are generated sequentially and exhibit temporal trends. To improve anomaly detection performance, it is crucial to model temporal dependencies within these sequences. Therefore, we adopt the TAM module to capture time dependencies for data points in a sliding window. As a first step, we extend the attention mechanism in Eq. (1) as

$$\text{TemporalAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{C}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{C}\right)\mathbf{V}, \quad (13)$$

where  $\mathbf{C}$  is the average of attention scores of multi-source data. This is used to capture the common temporal dependencies shared by the multi-source data and fuse the complementary information among them.

Next, we apply the multi-head self-attention across time to a sequence of the spatial representation of the MST graph given by the SAM. The structure of TAM is shown in Fig. 4. By capturing the temporal dependency, the feature of metrics  $\mathbf{M}_{t-k+1:t}$  can be updated as

$$\mathbf{M}'_{t-k+1:t} = \text{Multihead}(\mathbf{M}_{t-k+1:t}, \mathbf{M}_{t-k+1:t}, \mathbf{M}_{t-k+1:t}, \mathbf{C}),$$

where each  $H_i = \text{TemporalAtt}(\mathbf{M}_{t-d+1:t} \mathbf{W}_i^Q, \mathbf{M}_{t-d+1:t} \mathbf{W}_i^K, \mathbf{M}_{t-d+1:t} \mathbf{W}_i^V, \mathbf{C})$ , and the average of attention score  $\mathbf{C}$  is defined as

$$\left( \frac{\mathbf{M}_{t-k+1:t} \mathbf{M}_{t-k+1:t}^\top}{\sqrt{F_m}} + \frac{\mathbf{L}_{t-k+1:t} \mathbf{L}_{t-k+1:t}^\top}{\sqrt{F_l}} + \frac{\mathbf{S}_{t-k+1:t} \mathbf{S}_{t-k+1:t}^\top}{\sqrt{F_s}} \right) / 3.$$

Similarly, the feature of logs and traces can be updated as

$$\mathbf{L}'_{t-k+1:t} = \text{Multihead}(\mathbf{L}_{t-k+1:t}, \mathbf{L}_{t-k+1:t}, \mathbf{L}_{t-k+1:t}, \mathbf{C}),$$

$$\mathbf{S}'_{t-k+1:t} = \text{Multihead}(\mathbf{S}_{t-k+1:t}, \mathbf{S}_{t-k+1:t}, \mathbf{S}_{t-k+1:t}, \mathbf{C}).$$

In summary, we will update all modalities within  $\mathbb{D}_t$  as

$$\mathbb{D}'_t = \text{TAM}(\text{Norm}(\mathbb{D}_t + \mathbb{D}^{\text{SAM}}_t))$$

$$= (\mathbf{M}_{t-k+1:t}^{'}, \mathbf{L}_{t-k+1:t}^{'}, \mathbf{S}_{t-k+1:t}^{'}).$$

3) *Cross Attention Module*: The cross-attention module CAM is proposed to perform decoding using the hidden state  $\mathbb{Z}_t$  given by the encoder. This module assists the decoder in obtaining guidance information from  $\mathbb{Z}_t$ . To achieve this goal, we adopt the multi-head attention (see Eq.(2)). To distinguish features between  $\mathbb{D}_t$  and  $\mathbb{Z}_t$ , we add a superscript, e.g.  $\mathbf{M}_{t-k+1:t}^{\mathbb{D}}$  and  $\mathbf{M}_{t-k+1:t}^{\mathbb{D}}$ , to indicate the feature of metrics in the encoding and decoding stages respectively. In CAM, features of the three modalities are updated by

$$\begin{aligned}\mathbf{M}_{t-k+1:t}^{\mathbb{D}} &= \text{MultiHead}(\mathbf{M}_{t-k+1:t}^{\mathbb{D}}, \mathbf{M}_{t-k+1:t}^{\mathbb{Z}}, \mathbf{M}_{t-k+1:t}^{\mathbb{Z}}), \\ \mathbf{L}_{t-k+1:t}^{\mathbb{D}} &= \text{MultiHead}(\mathbf{L}_{t-k+1:t}^{\mathbb{D}}, \mathbf{L}_{t-k+1:t}^{\mathbb{Z}}, \mathbf{L}_{t-k+1:t}^{\mathbb{Z}}), \\ \mathbf{S}_{t-k+1:t}^{\mathbb{D}} &= \text{MultiHead}(\mathbf{S}_{t-k+1:t}^{\mathbb{D}}, \mathbf{S}_{t-k+1:t}^{\mathbb{Z}}, \mathbf{S}_{t-k+1:t}^{\mathbb{Z}}).\end{aligned}$$

Similarly to TAM, we can update all modalities by  $\mathbb{D}_t$  and  $\mathbb{Z}_t$  to obtain a new representation  $\mathbb{D}_t^{\text{CAM}}$  of the sliding window.

$$\begin{aligned}\mathbb{D}_t^{\text{CAM}} &= \text{CAM}(\text{Norm}(DI_1 + \text{Mask}(\text{TAM}(DI_1))), \mathbb{Z}_t) \\ &= (\mathbf{M}_{t-k+1:t}^{\mathbb{D}}, \mathbf{L}_{t-k+1:t}^{\mathbb{D}}, \mathbf{S}_{t-k+1:t}^{\mathbb{D}}).\end{aligned}$$

where  $DI_1 = \text{Norm}(\mathbb{D}_t + \mathbb{D}_t^{\text{SAM}})$ . This updated representation is then passed through the feed-forward and normalization layers to generate the final reconstruction representation.

4) *Loss Function*: For an anomaly algorithm deployed to production, given a sliding window  $\mathbb{D}_t = \{G_{t-k+1}, \dots, G_t\}$  with  $k$  MST graphs, our primary concern is whether the service instances in the last graph  $G_t$  is abnormal or not. Therefore, we only calculate the distance between the input  $G_t$  and the output  $G'_t$  as

$$\|G_t - G'_t\| = \mathbf{R} \circ \mathbf{R}$$

where  $\circ$  indicates the element-wise product,  $\mathbf{R} = \text{Concat}((\mathbf{M}_t - \mathbf{M}'_t), (\mathbf{L}_t - \mathbf{L}'_t), \mathbf{W}(\mathbf{S}_t - \mathbf{S}'_t)) \in \mathbb{R}^{N \times (F_m+F_l+F_s)}$ , and  $\mathbf{W}$  is a learnable parameter that maps the third item to the same dimension as the first two items. Using this distance, we can make predictions for the service instances within an MST graph via supervised learning (i.e., the probability of being abnormal  $P_t \in \mathbb{R}^{N \times 2}$ ) and unsupervised learning (i.e., the reconstruction error  $RE_t \in \mathbb{R}^N$ ), as follows:

$$\begin{aligned}RE_t &= \text{sum}(\|G_t - G'_t\|, 2), \\ P_t &= \text{Softmax}(\text{MLP}(\|G_t - G'_t\|)),\end{aligned}$$

where  $\text{sum}(\cdot, 2)$  indicates the summation by columns.

MSTGAD aims to train an efficient anomaly detection model based on limited labeled data and a substantial amount of unlabelled data. To this end, we adopt a semi-supervised reconstruction loss and a supervised classification loss, which can be combined as follows:

$$\text{Loss} = \frac{1}{\text{epoch}} \mathcal{L}_1 + (1 - \frac{1}{\text{epoch}}) \mathcal{L}_2, \quad (14)$$

where  $\mathcal{L}_1$  is the loss for data reconstruction,  $\mathcal{L}_2$  is the loss for classification, and  $\text{epoch}$  represents the current epoch number. More concretely, the semi-supervised loss  $\mathcal{L}_1$  is defined as:

$$\mathcal{L}_1 = \frac{1}{m+n} (\eta \sum_{i=1}^m RE_i + \sum_{i=1}^{n_a} \frac{1}{RE_i} + \sum_{i=1}^{n_n} RE_i), \quad (15)$$

---

#### Algorithm 1: The MSTGAD training algorithm

---

```

Input: Dataset used for training  $\{\mathbb{D}_t\}_{t=1}^T$ , Encoder  $E$ , Decoder  $D$ , Iteration limit  $M$ ;
Output: Model Coefficients
1 Initialization: Initialize the weights of  $E$  and  $D$ ;
2  $i \leftarrow 1$ ;
3 repeat
4   for  $t = 1$  to  $T$  do
5      $EI_0^t, DI_0^t \leftarrow \text{InputEmbedding}(\mathbb{D}_t)$ ;
6      $\mathbb{Z}_t \leftarrow E(EI_0^t)$  ;
7      $G'_t \leftarrow D(DI_0^t, \mathbb{Z}_t)$ ;
8     Calculate  $RE_t$  and  $P_t$ ;
9      $\mathcal{L} = \frac{1}{i} \mathcal{L}_1 + (1 - \frac{1}{i}) \mathcal{L}_2$ ;
10    Update the weights of  $E$  and  $D$  using  $\mathcal{L}$ ;
11    $i \leftarrow i + 1$ ;
12 until  $i > M$ ;
```

---

where  $m$  and  $n$  are respectively the numbers of unlabeled and labeled service instances in the training data,  $RE_i$  indicates the reconstruction error for the  $i$ -th service instance,  $\eta$  is a hyperparameter that balances the weight between labeled and unlabeled data,  $n_a$  and  $n_n$  are the numbers of abnormal and normal service instances in the training data, respectively, and  $n = n_a + n_n$ . This semi-supervised reconstruction loss helps the model learn the underlying patterns in the data and detect anomalies based on deviations from these patterns.

The supervised classification loss, on the other hand, helps the model to classify the service instances as normal or abnormal based on the learned patterns. Specifically, we employ the binary cross entropy loss:

$$\mathcal{L}_2 = -\frac{1}{n} \sum_{i=1}^n [\frac{n_n}{n_a} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)], \quad (16)$$

where  $\hat{y}_i$  denotes the predicted probability for the  $i$ -th service instance,  $\frac{n_n}{n_a}$  is introduced to control the trade-off between normal and abnormal data in the loss function. This loss function enables the model to learn to distinguish between normal and abnormal service instances more accurately, which leads to better performance in detecting anomalies.

Note that in Eq. (14) the weight of the two losses changes with the number of epochs to balance the relative importance of the reconstruction loss and the classification loss during the training process. In the early stages of training, the model relies more on the reconstruction loss to learn the underlying patterns in the data and reconstruct the input data accurately. At this stage, the importance of the classification loss is relatively smaller, as the model has not yet learned to distinguish between normal and abnormal service instances accurately. As the training progresses, the model becomes better at reconstructing the input data and learning the underlying patterns. At this stage, the importance of the classification loss increases, as the model needs to learn to distinguish between normal and abnormal service instances more accurately. Therefore, we adjust the weight of the two

losses with the number of epochs, starting with a higher weight for the reconstruction loss and gradually decreasing it while increasing the weight of the classification loss. This approach ensures that the model can learn the underlying patterns in the data effectively while also accurately identifying anomalies, ultimately leading to a more robust and accurate anomaly detection model. The overall training processes of MSTGAD are summarized in Algorithm 1.

## V. EVALUATION

### A. Datasets

To evaluate the effectiveness of our proposed method, we used two multi-modal anomaly detection datasets in our experiment. Both datasets contain metrics, logs, and traces.

1) *MSDS*: The multi-modal dataset MSDS<sup>1</sup> [42] is composed of distributed traces, application logs, and metrics from a complex distributed system (Openstack) that is used for AI-powered analytics. The metrics data contains information from 5 physical nodes in the infrastructure, each containing 7 metrics such as RAM and CPU usage. The log files are distributed across the infrastructure and recorded for each node, with a total of 23 features. The trace information encompasses various attributes such as host, event name, service name, span ID, parent ID, and trace ID. Notably, the MSDS also provides a JSON file containing the ground-truth information for the injected anomalies, including their start and end times, as well as their corresponding anomaly types. Hence, by leveraging this JSON file, the service instances' label information can be easily extracted and assigned within an MST graph. The ratio of normal and abnormal data is approximately 80:1.

2) *AIOps-Challenge*: The AIOps-Challenge<sup>2</sup> dataset serves as the foundation for the CCF International AIOps Challenge organized by CCF (China Computer Federation), Tsinghua University, and CCB (China Construction Bank) in 2022. This dataset is derived from a simulated e-commerce system operating on a microservice architecture, with 40 service instances deployed across 6 physical nodes. Each service instance records metrics, encompassing a total of 56 metrics, of which 25 are utilized in this study, including metrics related to RAM and CPU usage. Additionally, log files are recorded for each service instance, containing a collective set of 5 features, such as timestamps and original logs, among others. The traces within the dataset capture scheduling information among service instances, including timestamps, types, status codes, service instance names, span IDs, parent IDs, and trace IDs. Within the AIOps-Challenge dataset, three levels of anomalies are intentionally injected, specifically at the service, pod (service instance), and node levels. The injected anomalies are accompanied by their start times, levels, service names, and types. Accordingly, the label information for service instances can be extracted based on the injected anomalies. In specific terms, if a service anomaly is injected, the corresponding service instances are labeled as abnormal. Similarly, if a node

anomaly is injected, the service instances configured within that node are all labeled as abnormal. It is crucial to note that the end time of an injected anomaly is not provided. To address this limitation, we manually set the end time with a maximum interval of two minutes. The ratio of normal data to abnormal data in this dataset is approximately 120:1.

### B. Experiment Settings

Unless otherwise specified, for all datasets, we allocated 60% of the data for training, 10% for validation, and the remaining 30% for testing. Since PLELog, HADES, and our approach MSTGAD are semi-supervised, we randomly selected 50% of the training data as unlabelled. To evaluate the performance of all approaches, we utilized precision (PR), recall (RC), F1-score, average precision (AP), and area under the receiver operating characteristic curve (ROC/AUC) as the criteria [1]–[3]. PR indicates how many of the anomalous events predicted are actual. RC denotes the percentage of predicted abnormal events versus all abnormal events. The F1-score is the harmonic mean of precision and recall. AP indicates the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight.

All experiments are run on a Linux server equipped with an Intel(R) Xeon(R) Gold 5318Y CPU, 64 GB RAM, RTX 3090 with 24GB GPU memory, and Ubuntu 18.04.6 OS. Our implementation of MSTGAD is created using Python 3.8.13, PyTorch 1.12.0 [43] with CUDA 11.3, and PyTorch Geometric Library 2.2.0 [44]. The encoder and decoder layers were set to two, the batch size was 50, the window size was 10, and the Dropout was set to 0.2. We used the AdaBelief [45] optimizer with an initial learning rate of 0.001 and a step-scheduler with a step size of 0.9 for training. We trained models for up to 300 epochs and utilized early stopping with a patience of 15.

### C. Benchmark Algorithms

To verify the effectiveness of our proposed method MSTGAD, we conducted a comparative analysis with state-of-the-art models for multivariate time-series anomaly detection, including TraceAnomaly [18], PLELog [7], TranAD [15], USAD [12], SCWarn [23] and HADES [24]. Table I presents an overview of the distinguishing features of these models. We utilized the open-source codes provided by their respective publications for comparisons. For these approaches, the window size is 10, the batch size is 50, and the other hyperparameters of them are tuned by grid searching based on the validation set.

### D. Experimental Result

1) *Results of Anomaly Detection*: The average experimental results for the benchmark algorithms on the MSDS and AIOps-Challenge datasets are presented in Table II and III respectively. Based on the outcomes of the experiments, we can make the following observations:

MSTGAD, our proposed method, achieves the best performance among all compared approaches and outperforms

<sup>1</sup><https://zenodo.org/record/3549604>

<sup>2</sup><https://aiops-challenge.com/>

TABLE I  
CHARACTERISTICS OF THE COMPARING MODELS

Method	Data Used	Supervised Type	Online
TraceAnomaly	Trace	Unsupervised	✗
TranAD	Metric	Unsupervised	✓
USAD	Metric	Unsupervised	✓
SCWarn	Metric & Log	Unsupervised	✓
PLELog	Log	Semi-supervised	✓
HADES	Metric & Log	Semi-supervised	✓
MSTGAD	Metric&Log& Trace	Semi-supervised	✓

TABLE II  
EXPERIMENTAL RESULTS ON MSDS DATASET

Method	PR	RC	AUC	AP	F1
TraceAnomaly	0.903	<b>0.989</b>	0.986	0.894	0.944
PLELog	0.753	0.663	0.826	0.516	0.705
TranAD	0.772	0.815	0.907	0.815	0.793
USAD	0.481	0.463	0.730	0.611	0.472
SCWarn	0.440	0.371	0.679	0.581	0.402
HADES	0.908	0.895	0.947	0.814	0.901
MSTGAD	<b>0.946</b>	0.969	<b>0.996</b>	<b>0.971</b>	<b>0.957</b>

all baselines by a significant margin, attaining  $F_1$ -score of 0.957 and 0.965 on the two datasets. The high scores of MSTGAD indicate that there are few instances of missed anomalies or false alarms, which highlights the effectiveness of our approach for anomaly detection.

Moreover, compared with single-modal approaches, MSTGAD achieves superior performance. Previous studies have shown that metrics, logs, and traces can all reflect anomalies, and none of them are sufficient when acting alone. Thus, constructing the model based on a single data modality can omit important information hidden in the other data modalities, resulting in performance degradation. Additionally, the log-based approach PLELog’s performance may be influenced by the distribution of logs, i.e., the significant difference in the number of logs in different sliding windows.

On the other hand, compared with multi-modal based approaches SCWarn and HADES, MSTGAD outperforms them across all measurements on average. The possible reasons are summarized from three aspects: 1) SCWarn and HADES only use two types of data modalities, i.e., metrics and logs. 2) SCWarn is an unsupervised method and does not use supervised information to guide the training process. Furthermore, it detects anomalies on each modality individually and fails to identify the hidden correlations among the multi-modal data. 3) HADES is designed with an attention-based module, which fuses information between metrics and logs. However, it only detects anomalies for a single service instance and ignores the interaction between multiple service instances.

In contrast, MSTGAD detects anomalies via multi-modal learning based on metrics, logs, and traces simultaneously. The three modalities are effectively integrated via an MST graph, while spatial and temporal attention modules are proposed to model the interactions between different data modalities, and the complementary information among them and specific information are thoroughly fused and effectively integrated.

In conclusion, MSTGAD effectively detects abnormal pat-

TABLE III  
EXPERIMENTAL RESULTS ON AIOPS-CHALLENGE DATASET

Method	PR	RC	AUC	AP	F1
TraceAnomaly	0.857	0.239	0.619	0.234	0.374
PLELog	0.750	0.173	0.586	0.142	0.281
TranAD	0.661	0.789	0.827	0.738	0.719
USAD	0.667	0.750	0.813	0.728	0.706
SCWarn	0.878	0.625	0.798	0.762	0.730
HADES	0.911	<b>0.937</b>	0.958	0.865	0.924
MSTGAD	<b>1</b>	0.933	<b>0.974</b>	<b>0.977</b>	<b>0.965</b>

TABLE IV  
EXPERIMENTAL RESULTS OF ABLATION STUDY ON MSDS DATASET

Method	PR	RC	AUC	AP	F1
MSTGAD-Metric	0.920	0.957	<b>0.996</b>	0.960	0.937
MSTGAD-Log	0.758	0.917	0.986	0.949	0.830
MSTGAD-Trace	0.720	0.920	0.977	0.876	0.807
MSTGAD-TAM	0.932	0.917	0.960	0.948	0.921
MSTGAD-SAM	<b>0.965</b>	0.929	0.975	0.967	0.947
MSTGAD-C	0.939	0.929	0.967	0.955	0.933
MSTGAD	0.946	<b>0.969</b>	<b>0.996</b>	<b>0.971</b>	<b>0.957</b>

terns in all datasets, significantly proving its superiority over all baselines in terms of every evaluation criterion.

2) *Ablation Study*: To comprehensively assess the efficacy of different modules of MSTGAD, we conducted an ablation study on the MSDS dataset, and the experimental results are listed in Table IV, where “MSTGAD-X” indicates that MSTGAD is executed without the module or data modal “X”. For example, “MSTGAD-Metric” means MSTGAD is executed without the metrics data. It can be seen from Table IV that all modules and data modalities contribute significantly to improving the performance of MSTGAD. Concretely, MSTGAD exhibits a significant improvement over MSTGAD-Metric, MSTGAD-Log, and MSTGAD-Trace with an impressive margin of 3.933% in terms of  $F_1$ -score on average, showcasing its exceptional ability to effectively utilize multi-modal data for anomaly detection. Notably, MSTGAD-Trace performs worse than MSTGAD-Metric and MSTGAD-Log. Recall that in MSTGAD, both metrics and logs serve as node features, while only traces are utilized as edge features. When trace data is removed, the SAM module can only utilize the node features, compromising the effectiveness of updating and enhancing the graph representation. Consequently, the complementary information from different modalities cannot be effectively integrated, resulting in suboptimal performance. Remarkably, similar observations have been reported in Eadro [25], further emphasizing the fundamental role of trace data in combining and exploiting information from the other two modalities.

In the meantime, MSTGAD demonstrates statistically significant or, at the very least, comparable performance compared to MSTGAD-TAM and MSTGAD-SAM. The impressive performance of MSTGAD validates the effectiveness of incorporating both spatial and temporal attention mechanisms in anomaly detection. Moreover, MSTGAD-SAM outperforms MSTGAD-TAM, indicating that modeling temporal dependencies is crucial and highly effective in addressing anomaly detection challenges.

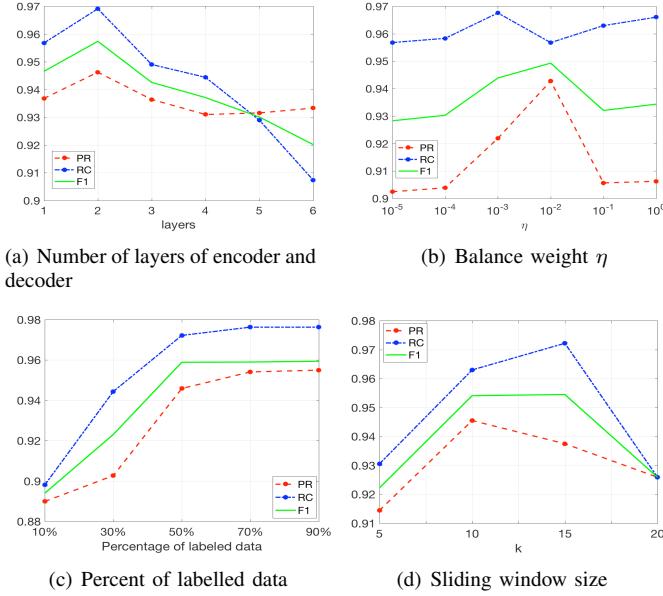


Fig. 5. Parameter Sensitivity Analysis

MSTGAD-C indicates that MSTGAD runs without modeling the common temporal dependency shared by different modalities, i.e., the matrix  $\mathbf{C}$  is removed from Eq.(13). As shown in Table IV, MSTGAD achieves a better performance than MSTGAD-C. It demonstrates the effectiveness of capturing the common temporal dependencies shared by the multi-source data and fusing the complementary information among them.

*3) Parameter Sensitivity Analysis:* The proposed method, MSTGAD, incorporates several crucial hyperparameters, including the sliding window size  $k$ , the number of encoder and decoder layers, the balance weight  $\eta$ , and the percentage of labeled data. To evaluate the sensitivity of these parameters, we conducted experiments on the MSDS dataset, and the results are displayed in Fig. 5.

Fig. 5(a) demonstrates the impact of the number of encoding and decoding layers on the performance of MSTGAD. We observed that a smaller or larger number of layers typically leads to lower performance, with the optimal range being between two and six layers. The performance of MSTGAD decreased with an increase in the number of hidden layers, probably due to the over-smoothing that makes the features indistinguishable, leading to reduced classification accuracy.

Fig. 5(b) illustrates the impact of the balance weight  $\eta$  on the precision, recall, and  $F_1$ -score of MSTGAD. As described in Section IV-B,  $\eta$  controls the trade-off between labeled and unlabeled data in the reconstruction loss. The results indicate that both precision and  $F_1$ -score decrease with an increase in  $\eta$ , primarily due to the model giving more attention to the noisy unlabeled data points.  $\eta$  is typically set between  $10^{-4}$  and  $10^{-1}$ , based on the preference for precision and recall.

The impact of the percentage of labeled data for the training stage on the performance of MSTGAD is depicted in Fig. 5(c).

TABLE V  
STATISTICAL CHARACTERISTICS OF THE EXPERIMENTAL DATASETS AND THE PROCESSING TIME OF MSTGAD

Dataset	Interval	# Metrics (instance*metric)	# Logs	# Spans
MSDS	1s	5*3	11	152
AIOps	1min	40*25	9260	6253
		Running Time		
Dataset	Preprocessing	Constructing graph	Detecting	Total
MSDS	0.014s	0.0088s	0.0035s	0.0263s
AIOps	6.7946s	0.173s	0.4648s	7.4324s

The results demonstrated that the performance of MSTGAD significantly improved with an increase in labeled data. It is worth noting that most of the labeled data are normal patterns that are easily obtainable in real-world applications.

Finally, Fig. 5(d) shows the impact of the sliding window size  $k$  on the precision, recall, and  $F_1$ -score of MSTGAD. Smaller or larger values of  $k$  typically lead to lower performance due to insufficient or unrelated features in the window. We observed that the best performance was achieved when  $k$  was set to 10, as shown in Fig. 5(d).

*4) Real-time Detection:* In the proposed method, the three modalities are first extracted and then integrated using an MST graph structure for anomaly detection. It is essential for the time consumption of this step to be smaller than the process interval of a microservice system, ensuring real-time detection capabilities. In this section, we present experimental results conducted on the MSDS and AIOps-Challenge datasets to demonstrate the efficiency of our method. The summarized results are presented in Table V, where # Metrics, # Logs, and # Spans correspond to the average number of metrics, logs, and spans observed within each processing interval. It is important to note that all experiments were conducted exclusively using CPU resources, without utilizing GPU or multithreading techniques for acceleration. The results unequivocally illustrate the applicability of the proposed method to real-world applications, exhibiting its capability to perform real-time anomaly detection.

## VI. CONCLUSION

In this paper, we propose a novel semi-supervised graph-based anomaly detection method MSTGAD that utilizes attentive multi-modal learning. We fuse the multi-modal data and represent them via an MST graph, where each node corresponds to a service instance, and the edge indicates the scheduling relationship between different service instances. Based on the MST graph sequences, we construct a transformer-based neural network with both spatial and temporal attention mechanisms to automatically and accurately detect anomalies in a timely manner. Experimental results demonstrate that MSTGAD achieves superior performance compared to state-of-the-art approaches. Furthermore, our results verify that effectively modeling the correlation between different data modalities and the time dependency within each data modality can further improve the performance of anomaly detection.

## REFERENCES

- [1] 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 *Proceedings of the IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, p. 623–634.
- [2] J. Soldani and A. Brogi, “Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey,” *ACM Comput. Surv.*, vol. 55, no. 3, feb 2022. [Online]. Available: <https://doi.org/10.1145/3501297>
- [3] P. Notaro, J. Cardoso, and M. Gerndt, “A survey of aiops methods for failure management,” *ACM Trans. Intell. Syst. Technol.*, vol. 12, no. 6, nov 2021. [Online]. Available: <https://doi.org/10.1145/3483424>
- [4] M. Du, F. Li, G. Zheng, and V. Srikanth, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, October 2017, p. 1285–1298.
- [5] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, “Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. AAAI Press, August 2019, p. 4739–4745.
- [6] V.-H. Le and H. Zhang, “Log-based anomaly detection without log parsing,” in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’21. IEEE Press, 2022, p. 492–504. [Online]. Available: <https://doi.org/10.1109/ASE51524.2021.9678773>
- [7] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, “Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Madrid, ser. ICSE ’21. IEEE Press, 2021, p. 230–231. [Online]. Available: <https://doi.org/10.1109/ICSE-Companion52605.2021.00106>
- [8] V.-H. Le and H. Zhang, “Log-based anomaly detection with deep learning: How far are we?” in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1356–1367. [Online]. Available: <https://doi.org/10.1145/3510003.3510155>
- [9] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.
- [10] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, “Deep one-class classification,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 4393–4402. [Online]. Available: <https://proceedings.mlr.press/v80/ruff18a.html>
- [11] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, “Deep autoencoding gaussian mixture model for unsupervised anomaly detection,” in *Proceedings of the International Conference on Learning Representations*, 2018. [Online]. Available: <https://openreview.net/forum?id=BJJLHbb0->
- [12] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, “Usad: Unsupervised anomaly detection on multivariate time series,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3395–3404. [Online]. Available: <https://doi.org/10.1145/3394486.3403392>
- [13] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, “Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3220–3230. [Online]. Available: <https://doi.org/10.1145/3447548.3467075>
- [14] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, “A transformer-based framework for multivariate time series representation learning,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2114–2124. [Online]. Available: <https://doi.org/10.1145/3447548.3467401>
- [15] S. Tuli, G. Casale, and N. R. Jennings, “Tranad: Deep transformer networks for anomaly detection in multivariate time series data,” *Proc. VLDB Endow.*, vol. 15, no. 6, p. 1201–1214, jun 2022. [Online]. Available: <https://doi.org/10.14778/3514061.3514067>
- [16] Z. Ren, C. Liu, X. Xiao, H. Jiang, and T. Xie, “Root cause localization for unreproducible builds via causality analysis over system call tracing,” in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’19. IEEE Press, 2020, p. 527–538. [Online]. Available: <https://doi.org/10.1109/ASE.2019.00056>
- [17] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, “Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 19–33. [Online]. Available: <https://doi.org/10.1145/3297858.3304004>
- [18] 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 *Proceedings of the IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 48–58.
- [19] 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>
- [20] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, “Microhecl: High-efficient root cause localization in large-scale microservice systems,” in *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP ’21. IEEE Press, 2021, p. 338–347. [Online]. Available: <https://doi.org/10.1109/ICSE-SEIP52600.2021.00043>
- [21] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, “Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments,” in *Proceedings of the Web Conference 2021*, ser. WWW ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3087–3098. [Online]. Available: <https://doi.org/10.1145/3442381.3449905>
- [22] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, and D. Pei, “Practical root cause localization for microservice systems via trace analysis,” in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.
- [23] N. Zhao, J. Chen, Z. Yu, H. Wang, J. Li, B. Qiu, H. Xu, W. Zhang, K. Sui, and D. Pei, “Identifying bad software changes via multimodal anomaly detection for online service systems,” in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 527–539. [Online]. Available: <https://doi.org/10.1145/3468264.3468543>
- [24] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, “Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 1724–1736.
- [25] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, “Eadro: An end-to-end troubleshooting framework for microservices on multi-source data,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 1750–1762.
- [26] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in *Proceedings of the 2017 IEEE International Conference on Web Services*, 2017, p. 33–40. [Online]. Available: <https://doi.org/10.1109/ICWS.2017.13>
- [27] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, “Abstracting execution logs to execution events for enterprise applications (short paper),” in *Proceedings of the 2008 The Eighth International Conference on Quality Software*, 2008, pp. 181–186.
- [28] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, “Clustering event logs using iterative partitioning,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1255–1264. [Online]. Available: <https://doi.org/10.1145/1557019.1557154>

- [29] M. Du and F. Li, “Spell: Streaming parsing of system event logs,” in *Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 859–864.
- [30] Y. Huo, Y. Su, C. Lee, and M. R. Lyu, “Semparser: A semantic parser for log analytics,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 881–893.
- [31] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, “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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817.
- [32] A. Deng and B. Hooi, “Graph neural network-based anomaly detection in multivariate time series,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4027–4035. [Online]. Available: <https://doi.org/10.1609/aaai.v35i5.16523>
- [33] C. Zhao, M. Ma, Z. Zhong, S. Zhang, Z. Tan, X. Xiong, L. Yu, J. Feng, Y. Sun, Y. Zhang, D. Pei, Q. Lin, and D. Zhang, “Robust multimodal failure detection for microservice systems,” 2023.
- [34] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin, D. Zhang, Z. Zhu, and D. Pei, “Robust failure diagnosis of microservice system through multimodal data,” *IEEE Transactions on Services Computing*, pp. 1–14, 2023.
- [35] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, “Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting,” *Advances in neural information processing systems*, vol. 32, 2019.
- [36] S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar, “Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting,” in *International conference on learning representations*, 2021.
- [37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008.
- [38] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: <https://openreview.net/forum?id=rJXMpikCZ>
- [39] S. Brody, U. Alon, and E. Yahav, “How attentive are graph attention networks?” in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.
- [40] Z. Wang, J. Chen, and H. Chen, “Egat: Edge-featured graph attention network,” in *Artificial Neural Networks and Machine Learning – ICANN 2021*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds. Cham: Springer International Publishing, 2021, pp. 253–264.
- [41] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, “System log parsing: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 8, pp. 8596–8614, 2023.
- [42] S. Nedelkoski, J. Bogatinovski, A. K. Mandapati, S. Becker, J. Cardoso, and O. Kao, “Multi-source distributed system data for ai-powered analytics,” in *Service-Oriented and Cloud Computing*, A. Brogi, W. Zimmermann, and K. Kritikos, Eds. Cham: Springer International Publishing, 2020, pp. 161–176.
- [43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, December 8-14, Vancouver, BC, Canada, 2019, pp. 8024–8035.
- [44] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *CoRR*, vol. abs/1903.02428, 2019, arXiv: 1903.02428. [Online]. Available: <http://arxiv.org/abs/1903.02428>
- [45] J. Zhuang, T. Tang, Y. Ding, S. Tatikonda, N. C. Dvornek, X. Papademetris, and J. S. Duncan, “AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients,” in *Advances in Neural Information Processing Systems*, December 6-12, virtual, 2020.