Robust Multimodal Failure Detection for Microservice Systems

Chenyu Zhao*
Nankai University
Tianjin, China

Minghua Ma*
Microsoft
Beijing, China

Zhenyu Zhong
Shenglin Zhang†
Nankai University
Tianjin, China

Zhiyuan Tan
Xiao Xiong
Nankai University
Tianjin, China

LuLu Yu
Jiayi Feng
Nankai University
Tianjin, China

Yongqian Sun
Yuzhi Zhang
Nankai University
Tianjin, China

Dan Pei
Tsinghua University
Beijing, China

Qingwei Lin
Dongmei Zhang
Microsoft
Beijing, China

ABSTRACT

Proactive failure detection of instances is vitally essential to microservice systems because an instance failure can propagate to the whole system and degrade the system's performance. Over the years, many single-modal (i.e., metrics, logs, or traces) data-based anomaly detection methods have been proposed. However, they tend to miss a large number of failures and generate numerous false alarms because they ignore the correlation of multimodal data. In this work, we propose *AnoFusion*, an unsupervised failure detection approach, to proactively detect instance failures through multimodal data for microservice systems. It applies a Graph Transformer Network (GTN) to learn the correlation of the heterogeneous multimodal data and integrates a Graph Attention Network (GAT) with Gated Recurrent Unit (GRU) to address the challenges introduced by dynamically changing multimodal data. We evaluate the performance of *AnoFusion* through two datasets, demonstrating that it achieves the F_1-score of 0.857 and 0.922, respectively, outperforming the state-of-the-art failure detection approaches.

CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**; • **Computing methodologies** → **Failure Detection; Graph Neural Networks**.

KEYWORDS

Microservice System, Multimodal, Failure Detection

*Equal Contribution
†Corresponding Author

1 INTRODUCTION

As an increasing number of Internet applications migrate to the cloud, the microservice architecture, which allows each microservice to be independently developed, deployed, upgraded, and scaled, has attracted widespread attention recently [47]. A microservice system is typically a large-scale system with many *instances* (e.g., virtual machines or containers). Correlations among instances, e.g., service invocations and resource contention, are usually complex and dynamic [42]. When an instance fails, it may degrade the performance of the whole microservice system, impact user experience and even lead to revenue loss. For example, some failed instances resulted in a surge of connection activity that overwhelmed the networking devices between the internal network and the main AWS network in December 2021 [2]. Therefore, it is crucial to proactively detect instance failures to mitigate failures timely.

Operators continuously collect three types of monitoring data, including metrics, logs, and traces for proactively detecting instance failures [30]. The metrics include system-level metrics (e.g., CPU utilization, memory utilization, and network throughput) and user-perceived metrics (e.g., average response time, error rate, and page view count). A log records the hardware or software runtime information, including state changes, debugging output, and system alerts. For an API request, a trace records its invocation chain through instances, where each service invocation is called a span.

We adopt failure and anomaly to characterize the faulty behaviors of instances and monitoring data: 1) *an anomaly* is a deviation from the normal system state (often reflected in monitoring data), and 2) *a failure* is an event where the service delivered by an instance goes wrong, and user experience is degraded [23]. Table 1 lists some types of anomalies and failures. For example, when a "login failure" occurs, users cannot log into the system successfully. Anomalies in logs and traces can be detected when this failure happens: many "ERROR"s will be printed in logs, and some trace data will have significantly larger Response Time (RT). It is common to observe many anomalies during a service failure. However, an (intermittent) anomaly does not necessarily lead to a failure.

Over the years, a significant number of methods have been proposed for automatic metric/log/trace (from now on, we call them single-modal) *anomaly detection*. They try to proactively detect the single-modal data's anomalous behaviors and determine that the instance fails when the monitoring data becomes anomalous. However, after investigating hundreds of instance failures (see § 5.1.1), we conclude that previous methods do not work well for *instance*

C.Zhao, M.Ma, Z.Zhong, S.Zhang, Z.Tan, X.Xiong, L.Yu, J.Feng, Y.Sun, Y.Zhang, D.Pei, Q.Lin, D.Zhang

Table 1: The anomalies of multimodal data during service failures. "Mem" represents the memory utilization metric, "ERR" is an error log, and $RT_{S_x \to S_y}$ denotes the response time when service instance S_x calls S_y. "–" means no anomaly is found or data is lost in that data modality.

Failure Type	Metric	Log	Trace	# Failures
failed of QR code	Mem ↑	–	–	505
system stuck	Mem ↓	–	–	16
login failure	–	ERR	$RT_{S_1 \to S_2}$=11s	527
file not found	–	–	$RT_{S_2 \to S_3}$=1.5s	36
access denied	–	ERR	$RT_{S_2 \to S_4}$=1.1s	15

failure detection in microservice systems. Correlating metrics, logs, and traces (from now on, we call them multimodal) is crucial for instance failure detection. On the one hand, the single-modal data cannot reveal all types of failures, let alone itself can be missing or collected too slowly [23]. For example, in Table 1, when the failure "failed to generate QR code" happens, only the instance's metrics, i.e., memory utilization, increase dramatically and exhibit anomalous behaviors. If only conducting log or trace anomaly detection, this type of failure will be falsely ignored. On the other hand, simply combining the anomaly detection results of the single-modal anomaly detection methods may generate false alarms, which have been confirmed in our experiments (see Table 3). For instance, the (transient) anomalies detected by single-modal anomaly detection methods may not represent any instance failure. Suppose an instance's network throughput metric experiences an anomaly and an alarm is reported because the metric increases suddenly and returns to the normal level after a short period. However, the system still delivers normal service, because no trace data becomes anomalous, and user experience is not impacted. Hence, no instance failure should be reported.

To this end, we aim to correlate the multimodal data to detect instance failures for microservice systems, which face the following two challenges. (1) Modeling the *complex correlations* among multimodal data. When a failure occurs, one, two, or three modalities of data can become anomalous, and they are correlated with each other. Neglecting the correlations can degrade the failure detection accuracy. (2) Dealing with the *heterogeneous and dynamically changing multimodal data.* Specifically, metrics are usually in the form of multivariate time series, and logs are typically semi-structured text. Moreover, a trace consists of spans in a tree structure. Integrating such heterogeneous multimodal data is quite challenging. Additionally, an instance's multimodal data usually changes dynamically over time.

In this work, we propose *AnoFusion*, an unsupervised instance failure detection approach for microservice systems. To address the first challenge, we apply Graph Transformer Network (GTN) [16, 45] since it can embed multimodal data into a graph and learn the correlation of heterogeneous data through the effective representations of graph nodes and edges [40, 43]. To address the second challenge, we first serialize the data of each modality according to the modality's characteristics and construct the nodes and edges of heterogeneous graphs. After that, we adopt Graph Attention Network (GAT) [36], which assigns different weights to neighbor nodes
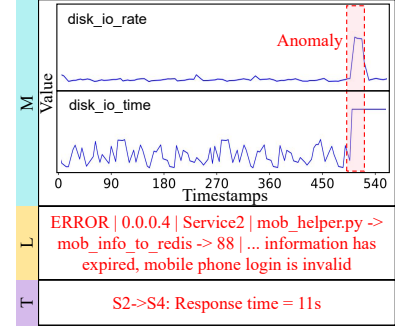

Figure 1: The multimodal monitoring data, i.e., Metrics, Logs, and Trace, during an instance failure

and learns the dynamic patterns of multimodal data, to optimize the graphs and filter significant node information. We then use a Gated Recurrent Unit (GRU) [19] to capture the temporal information and predict the multimodal data of the next moment. Finally, the similarity between the observation and prediction values is used to determine whether an instance fails.

The contributions of this paper are summarized as follows:

- To the best of our knowledge, we are among the first to identify the importance of exploring the correlation of multimodal monitoring data (i.e., metrics, logs, and traces), and correlate the multimodal data using GTN for instance failure detection.
- Our approach, *AnoFusion*, serializes the data of the three modalities according to each modality's trait. It combines GTN and GAT to detect anomalies in the dynamic multimodal data robustly. In addition, a GRU layer is used to capture the temporal information of the multimodal data.
- We adopt two microservice systems, consisting of 10 and 28 instances, respectively, to evaluate the performance of *AnoFusion*. The evaluation results show that *AnoFusion* detects instance failures with average F_1-score of 0.857 and 0.922, outperforming baseline methods by 0.278 and 0.480 on average, respectively.

Our source code and experimental data are available at **https://github.com/zcyyc/AnoFusion**.

2 BACKGROUND AND RELATED WORK

2.1 Single-modal Anomaly Detection

Generally, operators continuously collect three types of observable monitoring data: metrics, logs, and traces [30] to ensure the reliability of microservice systems. Figure 1 shows an example of anomalous multimodal data in a failure case.

Metric. A metric is defined as $\mathbf{x} = \{x_1, x_2, \ldots, x_T\}$, where T is the length of the metric, $x_t \in \mathbb{R}$ denotes the observation at time t. A microservice instance typically has a set of metrics that can be represented as a multivariate time series, monitoring various service metrics (e.g., page view) and system/hardware metrics (e.g., CPU usage). Figure 1 (M part) shows an example of metric data. Traditional statistic metric anomaly detection methods [24] do not need training data but can be less effective when facing intricate data. Supervised learning methods [17, 20] need operators to manually

label anomalies, which is impractical in many real-world scenarios. Thus, unsupervised methods [1, 25, 25, 35, 44] that do not require anomaly labels have become a hot research topic in recent years. For example, JumpStarter [25] applies a compressed sensing technique for anomaly detection. USAD [1] detects anomalies through adversarial training with high efficiency. A metric anomaly detection method can easily detect an instance failure if multiple metrics become anomalous soon after the failure. However, since metric anomaly detection methods only utilize metric data to detect anomalies and possible failures in a system, they will fail to alert operators when a failure does not manifest itself on metrics. *AnoFusion* analyzes metric data in an unsupervised way and reduces false alarms by using metric data together with other data modalities.

Log. Log data is semi-structured text output by instances at the application or system level. It is typically used to record the operational status of hardware or software. Generally, logs are generated with a predefined structure. As a result, extracting log templates and their parameters is a standard step in analyzing log data [7]. For example, Figure 1 (L part) lists a log. Traditional log anomaly detection methods are usually designed to identify keywords in logs like "ERROR" or "fail". However, negative keywords such as "fail" may appear in logs due to network jitters or operator login failure, and they do not imply an instance failure. Advanced approaches follow a similar workflow: log parsing, feature extraction, and anomaly detection [11]. Deep learning-based methods learn the log patterns (e.g., sequential feature, quantitative relationship) of normal executions and determine an anomaly when the pattern of a log sequence deviates from the learned normal patterns [5, 22, 41]. For example, LogAnomaly [27] applies template vectors to extract the hidden semantic information in the log templates and detects continuous and quantitative log anomalies at the same time. Deeplog [5] predicts the logs that may appear after a sliding window utilizing the LSTM model. *AnoFusion* requires neither labeling work nor domain knowledge when analyzing log data.

Trace. A trace is made up of spans, each of which corresponds to a service invocation [29]. Figure 1 (T part) shows an example of trace data. When the service processes a user's request, several instances will be invoked. The monitoring system records when a specific service is called and when it responds, and the difference between them is the Response Time (RT). Most trace anomaly detection methods detect anomalies according to whether the response time of each invocation increases dramatically and/or whether the invocation path behaves abnormally [9, 18, 21, 28, 29]. For instance, TraceAnomaly [21] learns the normal patterns of traces, and anomalies are detected when their patterns deviate from those of normal traces. However, on the one hand, a trace anomaly alone does not necessarily denotes an instance fails. On the other hand, an instance failure may not manifest itself in the trace data. Therefore, using trace anomaly detection methods alone can also lead to missed alerts or false alarms. *AnoFusion* can combine trace data with other modalities to boost anomaly detection performance.

2.2 Multimodal Anomaly Detection

Deep learning-based multimodal data fusion has witnessed great success in several research fields. For example, video subtitle generation [12], conversation behavior classification [32], and emotion

recognition [14]. Recent studies have started to tackle the anomaly detection problem based on multimodal data. Vijayanand et al. [37] propose an anomaly detection framework for cloud servers using multidimensional data, including different features such as network traffic sequence features, CPU usage, and memory usage from host logs. These extracted multidimensional features are fed to the detection model that identifies the anomalies and maximizes the detection accuracy. [6] performs correlation analysis on metrics and logs to discover the anomaly patterns in cloud operations. Additionally, SCWarn [46] combines metrics and logs for anomaly detection by serializing the metrics and logs separately and adopting LSTM to detect failures. However, traces, which are vital to instances, are missing in these works, and thus, they cannot achieve optimal performance when detecting anomalies in our scenario. To the best of our knowledge, we are among the first to focus on detecting instance failures using multimodal data.

2.3 Graph Neural Networks

GTN. GTN takes a heterogeneous graph as input and turns it into a new graph structure specified by meta-paths. Meta-paths are relational sequences that connect pairs of objects in heterogeneous graphs, which are commonly employed to extract structural information. By combining multiple GT layers with GCN, GTN learns node representations on the graph efficiently in an end-to-end way [43]. We apply GTN to learn the correlations among multimodal data in our scenario.

GAT. GCN is not good at analyzing dynamic graphs, and when the graph structure of training and test sets changes, GCN will no longer be suitable. In addition, GCN assigns the same weight to each neighbor node, which falls short of our expectations for future graph structure optimization. GAT solves the problems of GCN by allocating various weights to different nodes. It enables various nodes to be distinguished in terms of importance, so that *AnoFusion* can focus on more significant information in the graph structure [36]. Therefore, GAT is expected to achieve better performance in processing dynamically changing time series data, and thus we utilize GAT instead of GCN.

GRU. As we know, RNN [15] can represent time dependency by adopting deterministic hidden variables. However, RNN may be incapable of dealing with the long-term dependency problem in the time series, and LSTM [13] and GRU [19] are proposed as solutions. Generally, GRU is often comparable to LSTM, and the fewer parameters and more straightforward structure make it ideal for model training [35]. We thus apply GRU to capture the time dependency of the multimodal data.

3 MOTIVATION

To prove that single-modal data is insufficient to comprehensively capture the failure patterns of instances, we adopt two datasets (see Section 5.1 for more details) for an empirical study. These datasets contain the multimodal data (i.e., metrics, logs, and traces) collected from microservice systems. It also includes the records of all failure injections for a fair evaluation.

We perform a thorough internal data analysis to investigate the correlation of different modalities. Table 1 lists some monitoring

data collected from a microservice system. Many instance failures cannot be successfully captured using single-modal data. It also shows that when a failure occurs, data of different modalities may display anomalous patterns at the same time. Mining the correlation between multimodal data can provide more comprehensive and accurate information for failure detection tasks.

Moreover, we experiment to evaluate the failure detection performance of single-modal data-based anomaly detection methods. We apply five popular single-modal anomaly detection methods (Section 5.1), JumpStarter [25], USAD [1], LogAnomaly [27], Deeplog [5], TraceAnomaly [21], and the combination of JumpStarter, LogAnomaly, and TraceAnomaly, to conduct metric/log/trace anomaly detection, respectively. Table 3 lists the precision, recall, and F_1-score of these methods.

Metric anomaly detection. JumpStarter and USAD achieve low performance on the two datasets. JumpStarter extracts real-time normal patterns from metric data, but it does not consider the patterns of historical data. USAD is not very noise-resilient, which results in a significant number of false positives and false negatives. Furthermore, they do not take logs and traces into account, thus they lack essential information from logs and traces for failure detection tasks.

Log anomaly detection. LogAnomaly and Deeplog achieve relatively high F_1-score on D1. It is because the anomaly patterns of the log data in D1 are more obvious and straightforward to identify than those in D2. When a keyword like "ERROR" appears in logs, there is a high probability that it is an instance failure. However, only relying on log data results in a large number of false positives and false negatives in D2, for some failures do not manifest themselves obviously in logs, and some anomalous logs do not indicate an instance failure.

Trace anomaly detection. TraceAnomaly gets unsatisfactory F_1-score on both datasets. The precision of TraceAnomaly is low, indicating that there are a huge number of false positives. Because TraceAnomaly determines anomalies based on only response time. However, a larger response time quickly returning to normal status does not indicate an instance failure.

JLT. JLT aggregates the results of **J**umpStarter, **L**ogAnomaly, and **T**raceAnomaly directly, to form a multimodal baseline. The aggregation method is majority voting, which determines a failure if two or more modalities have anomalies at a certain moment. JLT suffers from low precision (recall) on D1 (D2), indicating a high false positive (negative) rate, because it ignores the correlation of the multimodal data. On the one hand, some failures only manifest themselves in a specific modality of data, making the majority voting strategy requiring two or more modalities to have anomalies for failure detection ineffective, which results in many false negatives on D2. On the other hand, since JumpStarter, LogAnomaly, and TraceAnomaly all suffer from low precision on D1, their combination, i.e., JLT, still experiences a high false positive rate.

In summary, single-modal anomaly detection approaches fail to detect failures robustly since they lack insights from other data modalities. Additionally, simply combining the anomaly detection results of the single-modal anomaly detection methods, instead of mining the correlations of the multimodal data, cannot guarantee

high accuracy. Therefore, we attempt to design a robust instance failure detection approach by correlating the multimodal data.

4 *ANOFUSION*

4.1 Overview

As shown in Figure 2, the workflow of *AnoFusion* is divided into the offline training stage and the online detection stage. To capture the heterogeneity and correlation among multimodal data, we employ GTN to update the heterogeneous graphs. Moreover, to improve the robustness of *AnoFusion*, we apply GAT after GTN, making it perform stably when the data patterns of the training set and test set are different. In addition, to achieve unsupervised failure detection and make *AnoFusion* more suitable for time series data, we use GRU to predict the multimodal data of the next moment. In the offline training stage, *AnoFusion* consists of four main steps:

- **Multimodal Data Serialization.** To prepare for the future graph structure's construction, *AnoFusion* converts multimodal data (i.e., metrics, logs, and traces) into time series using predefined processes and aligns their time. After serialization, *AnoFusion* treats each time series as a "data channel".
- **Graph Stream Construction.** To build the raw inputs for GTN, *AnoFusion* constructs a heterogeneous graph containing all the data channels based on their connections for each moment t. Then, all heterogeneous graphs form a graph stream, which will be input into GTN.
- **Feature Filtering.** GTN updates the graph stream by learning the representations of nodes in the heterogeneous graph and capturing the correlation among different data modalities. The updated graph stream is regarded as the features of the original data channels. Then, *AnoFusion* utilizes GAT to give attention scores to the nodes in the graph stream, identifying different patterns and achieving feature filtering.
- **Failure Detection.** GRU is applied to temporal sequences to predict the values at the next moment based on the previous inputs. We train the GRU network to predict the next graph based on the given graph stream as accurately as possible.

In the online detecting stage, for a given time t, multimodal data will be serialized according to the observations in $[t - \theta, t]$, where θ is the input window size. Then, we use the serialized data channels to construct a heterogeneous graph stream. The graph stream of this window will be fed into the trained model to obtain the prediction of the next graph. *AnoFusion* calculates the similarity between the predicted graph and the observed graph as the failure score and then determines whether it is a failure. Note that *AnoFusion* does not restrict the dimensions of any modality data.

4.2 Multimodal Data Serialization

Serialization of metric data. Metrics collected from instances are in the form of time series, which have a serialized structure. Therefore, it only requires regular preprocessing steps such as normalization. The normalization process is given by: $\hat{\mathbf{m}} \equiv \mathbf{m}/|\mathbf{m}|$ where $\mathbf{m}$ is the raw metric data and $\hat{\mathbf{m}}$ is the normalized data. It scales individual samples to have a unit norm, which can be useful when using a quadratic form such as the dot-product to quantify the similarity of any pair of samples.
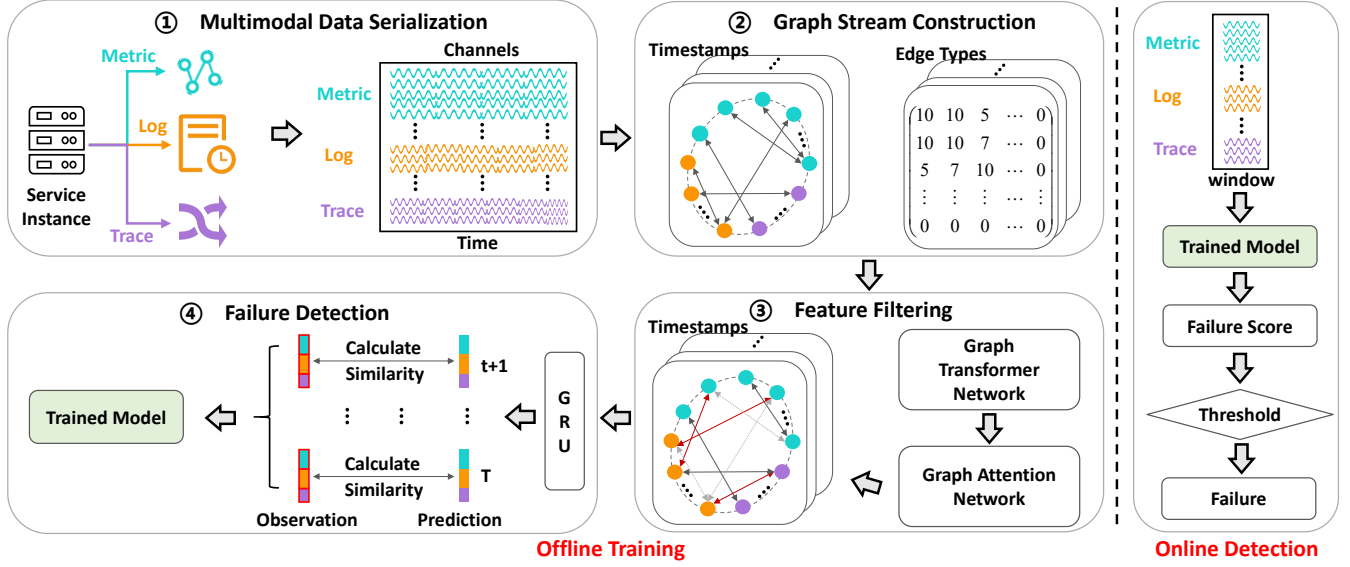
Figure 2: The framework of *AnoFusion*. It is an unsupervised learning approach without using labels in the offline training.

Serialization of log data. Parsing logs correctly and extracting log templates are the two essential steps of log serialization [39]. We adopt the advanced log parsing algorithm, Drain [10], which has shown its superiority and efficiency, to extract log templates in *AnoFusion*. The log serialization process is the following two steps: (1) **Clustering.** To deal with log changes caused by constantly updating code, adding new logs, and altering new logs in actual microservice systems, we first use a clustering algorithm for the log templates. By grouping similar log templates into clusters, on the one hand, the redundant information can be removed, and on the other hand, the types of log templates can be used to characterize log data. Once a new log template emerges due to a software update, the similarity between the new log template and the previous cluster centroids can be calculated, and it can be decided whether the new log template belongs in an existing cluster or should be regarded as a new cluster. Furthermore, through the empirical study of a large number of online service systems, we conclude that failures rarely occur in real-world scenarios [25]. Since *AnoFusion* is an unsupervised learning method based on the assumption that all training samples have normal patterns, removing anomalous log templates will improve the model's performance. Based on the analysis mentioned above, we finally utilize the "bert-base-uncased" model [4] to obtain sentence embedding vectors, and apply the DB-SCAN [33] algorithm to cluster these vectors. *AnoFusion* computes the centroid **c** of each cluster C by:

$$\mathbf{c} = \arg\min_{\mathbf{a} \in C} \sum_{\mathbf{b} \in C} |\mathbf{a} - \mathbf{b}| \tag{1}$$

(2) **Serialization.** The category of each log entry in the input can be determined by calculating the distance between its sentence embedding vector and that of the centroid of each cluster. After that, *AnoFusion* uses a sliding window to split the input log data into windows, each of which has the window length θ and the step size δ. We count the number of each category of logs as well as

the total number of logs in each window to form $M + 1$ time series, where M is the number of log template clusters. The horizontal axis $Timestamp$ corresponds to the input log's collection time.

Serialization of tracing data. *AnoFusion* uses the sliding window with length θ and step size δ to split tracing data. Each window contains tracing data (in the form of RT) of the invocations related to the instance. Then, for each window, *AnoFusion* computes the mean, median, range, and standard deviation of the invocations RT, producing four time series, respectively. If status code is available, *AnoFusion* may take them as the fifth time series. We treat each time series as a data channel, similar to the serialization of log data.

Clock synchronization. To build the graph structure more conveniently, *AnoFusion* synchronizes the clocks of the three modal data after serialization. The goal of *AnoFusion* is failure detection for a single instance and all the monitoring data acquired is within that instance. Therefore, the monitoring data clocks of the three modalities are relatively synchronized. The metric data is collected every minute. A log entry is generated when an event occurs in the instance. Moreover, a trace is recorded when a request is processed. Therefore, we obtained the features (e.g. the number of occurrences) of metrics, logs, and traces every minute in our scenario.

4.3 Graph Stream Construction

The data channels we get from the previous step can be described as $X = \left\{ \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \right\}$, where N is the number of data channels. *AnoFusion* constructs a heterogeneous graph G_t for each timestamp t using the extracted data channels. The node set of graph G_t, denoted by X_t, consists of the value of each data channel at time t, i.e., $X_t = \left\{ x_t^{(1)}, \ldots, x_t^{(N)} \right\}$. Since there are three modalities, there are also three types of nodes (i.e., metrics, logs, and traces). Thus, the number of edge types $K = 6$ (i.e., metrics-metrics, metrics-logs, metrics-traces, logs-logs, log-traces, traces-traces). The adjacency

matrix for each type of edge in graph G_t can now be expressed as $A^{(k)} \in \mathbb{R}^{N \times N}$, where $k = 1, \ldots, K$.

AnoFusion utilizes the mutual information (MI) [8] to calculate the adjacency matrix. For each data channel pair $\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right)$ with an edge type of k, the corresponding adjacency matrix value can be calculated as follows:

$$A_{i,j}^{(k)} = A_{j,i}^{(k)} = \sum_{a=1}^{\tau} \sum_{b=1}^{\tau} p\left(x_a^{(i)}, x_b^{(j)} \right) \log \frac{p\left(x_a^{(i)}, x_b^{(j)} \right)}{p\left(x_a^{(i)} \right) p\left(x_b^{(j)} \right)} \quad (2)$$

where τ is the number of timestamps (i.e., the length of each data channel), $p\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right)$ is the joint probability mass function of $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$, and $p\left(\mathbf{x}^{(i)} \right)$ and $p\left(\mathbf{x}^{(j)} \right)$ are the marginal probability mass functions of $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$, respectively. After calculating MI for all channel pairs, we now have the final adjacency matrix $A \in \mathbb{R}^{N \times N \times K}$. $G_t = (X_t, A)$ is defined to be the heterogeneous graph generated at time t. *AnoFusion* stacks the graphs of each moment together to form a graph stream $G = \{G_1, \ldots, G_\tau\}$.

4.4 Feature Filtering

AnoFusion performs feature filtering by updating the heterogeneous graph stream G with GTN and learning failure patterns by GAT.

Graph Transformer Network. GTN models the heterogeneity and correlation of multimodal channels using the adjacency matrix A. Graph Transformer layers (GT layers) are the main component of GTN. They learn the soft selection and composite relationship of edge types to produce useful multi-hop connections, also known as meta-path [43]. Specifically, considering the adjacency matrix A as the input, a GT layer has two steps: First, it softly constructs several graph structures from A by a 1×1 convolutional layer, which can be formulated as:

$$Q^{(k)} = \phi\left(A, W^{(k)} \right) = \sum_{i=1}^{K} w_i^{(k)} A^{(i)} \quad (3)$$

where $Q^{(k)}$ is the generated graph for edge type k, ϕ denotes the 1×1 convolution, $W^{(k)} \in \mathbb{R}^{1 \times 1 \times K}$ is the parameter of ϕ (for edge type k), K is the number of edge types. Second, it combines each $Q^{(k)}$ through matrix multiplication to generate a new graph structure A', a.k.a, meta-path:

$$A' = D^{-1} \prod_{k=1}^{K} Q^{(k)} \quad (4)$$

Note that we also normalize A' by D, which denotes the degree matrix of A, to ensure numerical stability. Stacking several GT layers in GTN aims to learn a high-level meta-path that is a useful relationship of multimodal data.

Graph Attention Network. With the meta-path matrix A' generated by stacking multiple GT layers, *AnoFusion* employs GAT on the heterogeneous graph stream to distinguish the significance of multimodal data channels and completes the feature filtering. The multi-head attention mechanism is utilized as well to stabilize this process. Specifically, for each channel pair $\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right)$, we first

compute a raw attention score for each attention head based on A':

$$\beta_{i,j}^{(h)} = LeakyRelu\left(A_{i,j}' \cdot concat\left(W\mathbf{x}^{(i)}, W\mathbf{x}^{(j)} \right) \right) \quad (5)$$

where $\beta^{(h)}$ is the attention score for the h-th attention head, W denotes the learnable parameter of a linear transformation. Then, *AnoFusion* normalizes the raw attention score with softmax and performs node feature aggregation for the i-th node $x^{(i)}$ by:

$$\widetilde{\beta}_{i,j}^{(h)\prime} = softmax\left(\beta_{i,j}^{(h)\prime} \right) = \frac{\exp\left(\beta_{i,j}^{(h)} \right)}{\sum_{l=1}^{N} \exp\left(\beta_{i,l}^{(h)} \right)}$$

$$\mathbf{x}^{(i)\prime} = W_H \cdot \underset{h=1}{\overset{H}{concat}} \left(\sum_{j=1}^{N} W^{(h)} \mathbf{x}^{(j)} \widetilde{\beta}_{i,j}^{(h)\prime} \right) \quad (6)$$

where H is the number of attention heads, $\beta_{i,l}^{(h)}$ denote the h-th head attention score between channel i and channel l, $W^{(h)}$ and W_H denote the linear transformation for each head and final output, respectively. The data channels are successively updated across the multi-layer Graph Attention Network.

4.5 Failure Detection

After feature filtering, we use the updated graph stream to train a failure detection model based on a recurrent neural network. Let $X' \in \mathbb{R}^{N \times \tau}$ denote the updated data channels, we can use GRU to capture its complex temporal dependence and predict the value of data channels at time τ. The GRU network can be formulated as:

$$\begin{aligned}
z_t &= \sigma\left(W_z X_t' + U_z h_{t-1} + b_z \right) \\
r_t &= \sigma\left(W_r X_t' + U_r h_{t-1} + b_r \right) \\
\hat{h}_t &= \tanh\left(W_h X_t' + U_h \left(h_{t-1} \odot r_t \right) + b_h \right) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t,
\end{aligned} \quad (7)$$

where σ denotes the sigmoid function, $\odot$ denotes the Hadamard product (i.e., element-wise product), X_t' is the input vector, h_{t-1} is the previous hidden state, $\hat{h}_t$ is the candidate activation vector, h_t is the hidden state and output vector of time t. z_t denotes the update gate, controlling how much information h_t needs to keep from h_{t-1}, and how much information needs to be received from $\hat{h}_t$. r_t denotes the reset gate, controlling whether the calculation of the candidate activation vector depends on the previous hidden state. W and U are trainable parameter matrices, and b is a trainable parameter vector. *AnoFusion* uses the final hidden state of GRU, h_t, to predict the value of data channels at time τ (i.e., the last moment in the graph stream):

$$\hat{X}_\tau' = \tanh\left(W_o h_{\tau-1} + b_o \right) \quad (8)$$

where W_o and b_o are the learnable parameters. *AnoFusion* adopts mean squared error (MSE) between the predicted value $\hat{X}_\tau'$ and the observation X_τ' as the loss function:

$$\mathcal{L} = \frac{1}{N} \|\hat{X}_\tau' - X_\tau'\|_2^2 \quad (9)$$

where N is the number of data channels. The GRU network is updated using this loss function iteratively.

4.6 Online Detection

In the online detection stage, for a new-coming multimodal monitoring data X_t, *AnoFusion* will first serialize the data using its previous historical observations, i.e., $X_{t-\theta+1:t-1}$, and construct the graph stream $G = \{G_{t-\theta+1}, \ldots, G_t\}$, where θ is the length of the window. Then, the graph stream is fed into the trained model to get a prediction $\hat{X}_t$ for X_t. We calculate the difference between the observed and predicted values for each data channel n [3]:

$$ERR_n = |\hat{X}_t^{(n)} - X_t^{(n)}| \tag{10}$$

Failures may only happen in part of the multimodal data, so we focus on the biggest error. *AnoFusion* utilizes the max function to aggregate $ERR_n(t), n \in [1, N]$:

$$ERR = \max_{n=1}^{N} \frac{ERR_n - \widetilde{\mu}}{\widetilde{\sigma}} \tag{11}$$

where ERR is the failure score at time t, $\widetilde{\mu}$ and $\widetilde{\sigma}$ are the median and inter-quartile range (IQR) of the set composed by ERR_n, respectively. We use median and IQR instead of mean and standard deviation as they are more robust.

AnoFusion uses a threshold to determine if a failure has occurred at a specific time t. However, using a static threshold is not effective since data distribution changes over time. To solve this problem, we employ the Extreme Value Theory (EVT) [34] to automatically and accurately determine the threshold. EVT is a statistical theory that identifies the occurrences of extreme values and doesn't make any assumptions about data distribution. EVT can be applied to estimate the likelihood of observing the extreme value for anomaly detection. EVT has been shown to accurately choose thresholds in previous failure detection methods [25, 26].

5 EVALUATION

In this section, we address the following research questions:

- **RQ1:** How well does *AnoFusion* perform in failure detection?
- **RQ2:** Does each component contribute to *AnoFusion*?
- **RQ3:** How do the major hyperparameters of *AnoFusion* influence its performance?

5.1 Experimental Design

5.1.1 Datasets. To evaluate the performance of *AnoFusion*, we conduct extensive experiments using two microservice systems (forming dataset 1 and 2, respectively). Table 2 lists the detailed information of the datasets. The second column indicates the number of microservices of each dataset. The third column indicates the number of instances of each dataset. The fourth column displays the average failure ratio ($\frac{The\ number\ of\ failure\ windows}{The\ total\ number\ of\ windows}$) of all instances. The fifth column lists every modality, and the last column shows the number of metrics, logs, or traces.

- **Dataset 1** (D1) is Generic AIOps Atlas (GAIA) dataset from CloudWise[1]. It contains the multimodal data collected from a system consisting of 10 instances, which is collected more than 0.7 million metrics, 87 million logs, and 28 million traces in two weeks. Real-world failures are injected, and Table 1 lists some

Table 2: The detailed information of the two datasets. #Micro and #Ins denote the number of microservices and instances, respectively.

	#Micro	#Ins	%Failures	Modality	#
D1	5	10	4.908	Metric	734,165
				Log	87,974,577
				Trace	28,681,438
D2	14	28	1.243	Metric	3,122,168
				Log	14,894,069
				Trace	9,473,763

typical symptoms of failure types, such as QR code generation failure, system stuck, file not found, and access denied, *etc.*

- **Dataset 2** (D2) is collected from a large-scale microservice system operated by a commercial bank. The system has 28 instances such as Web servers, application servers, databases, etc., and provides services for millions of users daily. Failures are injected into the system manually by professional operators and the multimodal monitoring data (i.e. metrics, logs, and traces) is collected. In general, these failures can be resource (CPU, memory, disk) failures, network failures (network packet loss and network latency), and application failures (VM failures). Due to the non-disclosure agreement, we cannot make this dataset publicly available.

5.1.2 Implementation. *AnoFusion* is implemented in PyTorch and all of the experiments are conducted on a Linux Server with two 16C32T Intel(R) Xeon(R) Gold 5218 CPU @ 2.30 GHz, two NVIDIA(R) Tesla(R) V100S, and 192 GB RAM. In the multimodal data serialization stage, we set the sliding window length $\theta = 60$ and step size $\delta = 1$ (more discussions can be found in Section 5.4). In the graph stream construction stage, we set the number of GT layers in GTN to 5, as suggested by [14]. For GAT, the total number of attention heads H is 6 (see Section 5.4 for more details). We split the multimodal monitoring datasets into a training set and a testing set, where the training set contains the front 60% data of each instance and the testing set contains the rest 40%.

5.1.3 Baseline Approaches. We utilize JumpStarter [25], USAD [1], LogAnomaly [27], Deeplog [5], TraceAnomaly [21], SCWarn [46], and JLT (see Section 3), which use single modality, two modalities, or three modalities of data for instance failure detection, as baselines. For all approaches, we use grid-search to set their parameters best for accuracy.

5.1.4 Evaluation Metrics. We adopt the widely-used True Positive (TP), False Positive (FP), and False Negative (FN), to label the failure detection results according to the ground truth [25, 26, 31]. A TP is a failure both confirmed by operators and detected by an approach. An FP is a normal window that is falsely identified as a failure by an approach. An FN is a missed failure that should have been detected. We calculate $precision = TP/(TP + FP)$, $recall = TP/(TP + FN)$, and F_1-score$= 2 \cdot precision \cdot recall/(precision + recall)$ to evaluate the overall performance of each approach.

5.2 RQ1: Effectiveness of *AnoFusion*

Table 3 lists the average precision, recall, and best F_1-score of *AnoFusion* and seven baseline approaches described above on the two

Table 3: The average precision, recall, and F_1-score of different approaches on the two datasets

Approach	Modality			D1			D2		
	Metric	Log	Trace	Precision	Recall	F_1-score	Precision	Recall	F_1-score
JumpStarter [25]	✓			0.466	0.785	0.584	0.533	0.413	0.465
USAD [1]	✓			0.459	0.825	0.590	0.837	0.341	0.484
LogAnomaly [27]		✓		0.486	0.957	0.644	0.126	0.344	0.184
Deeplog [5]		✓		0.506	0.812	0.623	0.105	0.275	0.151
TraceAnomaly [21]			✓	0.550	0.548	0.549	0.521	0.699	0.597
SCWarn [46]	✓	✓		0.547	0.425	0.447	0.633	0.891	0.734
JLT	✓	✓	✓	0.461	0.940	0.618	0.800	0.344	0.481
AnoFusion	✓	✓	✓	**0.795**	**0.945**	**0.857**	**0.863**	**0.991**	**0.922**

datasets. *AnoFusion* outperforms all baseline approaches on both datasets, with best F_1-score of 0.857 and 0.922, respectively.

Multimodal failure detection. SCWarn performs better than single-modal failure detection methods by simultaneously processing metrics and logs on D2. However, it ignores tracing data, which is crucial for detecting instance failures in microservice systems on D1. The correlation among each modality is ignored by JLT, yielding sub-optimal performance on both datasets.

AnoFusion. Our approach is effective to detect instance failures, with the average best F_1-score significantly higher than existing methods. Compared with SCWarn which combines two modalities, the average best F_1-score of *AnoFusion* outperforms it by 41.00% and 18.80% on both datasets, respectively. Compared with JLT, using a heterogeneous graph stream significantly improves the effectiveness of instance failure detection. *AnoFusion* outperforms JLT by 23.90% and 44.10% on both datasets, respectively.

Robustness comparison. Firstly, we use two datasets from different microservice systems, and the experiments show that *AnoFusion* achieves superior detection results on both datasets, outperforming all baselines. Secondly, each dataset contains many kinds of instances. We analyze the failure detection results of each instance for each dataset. The F_1-scores of *AnoFusion* for all instances on D1 range from 0.784 to 0.977, and from 0.805 to 0.986 on D2. We can see that *AnoFusion* performs well in both D1 and D2. Therefore, we believe these are testaments to the robustness of *AnoFusion*.

Efficiency comparison. We simulate the online detection environment and analyze the complexity of *AnoFusion* and other baselines by counting the detection time required for each sliding window. *AnoFusion* takes a window containing the data points as input and then calculates a failure score through the trained model. The average running time of *AnoFusion*'s online failure detection is 1.932×10^{-2}s. The prediction time of other baselines is 9.810×10^{-3}s for JumpStarter, 8.975×10^{-5}s for USAD, 1.707×10^{-3}s for LogAnomaly, 1.165×10^{-4}s for Deeplog, 1.102×10^{-2}s for TraceAnomaly, 3.331×10^{-3}s for SCWarn, and 9.958×10^{-3}s for JLT. Since operators perform failure detection every minute, every approach can satisfy this requirement. Furthermore, *AnoFusion* achieves satisfactory results by leveraging the three modalities, which is superior considering both effectiveness and performance.

Table 4: The average precision, recall, and F_1-score of *AnoFusion* and different model variants

	Approach	Precision	Recall	F_1-score
D1	w/o GTN	0.608	0.891	0.723
	use GCN	0.769	0.847	0.802
	w/o GAT	0.602	0.643	0.615
	AnoFusion	**0.795**	**0.945**	**0.857**
D2	w/o GTN	0.742	0.917	0.859
	use GCN	0.819	0.863	0.823
	w/o GAT	0.698	0.735	0.700
	AnoFusion	**0.863**	**0.991**	**0.922**

5.3 RQ2: Contributions of Components

To demonstrate the contribution and importance of each component of *AnoFusion*, we create three variants of *AnoFusion* and conduct a series of experiments to compare their performance. These variants are: 1) ***AnoFusion* w/o GTN.** To study the significance of GTN in modeling the heterogeneity and correlation of multimodal data, we remove GTN from *AnoFusion*. 2) ***AnoFusion* using GCN.** To show the importance of assigning different weights to neighbor nodes in the graph (attention mechanism), we use GCN instead of GAT in *AnoFusion*. 3) ***AnoFusion* w/o GAT.** To demonstrate how the graph attention mechanism improves *AnoFusion*'s performance, we remove the GAT from *AnoFusion*.

Table 4 lists the average precision, recall, and best F_1-score of the three variants discussed above on two datasets. When GTN is removed, both precision and recall are degraded. The decrease in precision is especially obvious. It shows that GTN can capture heterogeneity and correlation among multimodal data in heterogeneous graphs, thereby reducing false positives by comprehensively synthesizing the information of the three modalities. Both precision and recall decrease when GCN is substituted with GAT, demonstrating that GAT is more efficient than GCN for dynamically changing time series. Each node in the heterogeneous graph stream behaves differently. Treating all nodes equally like GCN introduces noise to the graph stream that can interfere with the model training process. Furthermore, when GAT is no longer present in *AnoFusion*, the precision and recall drop dramatically, which indicates that GAT can focus on the most relevant information in the graph.
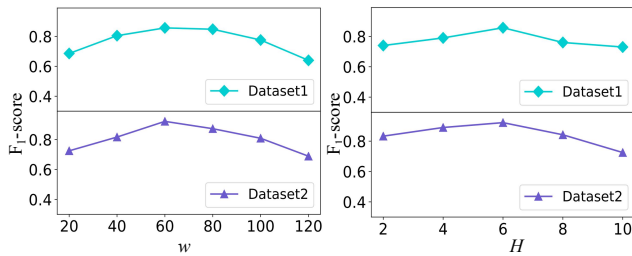
Figure 3: F_1-score of *AnoFusion* under different parameters

5.4 RQ3: Hyperparameters Sensitivity

We mainly discuss the effect of two hyperparameters in multimodal data serialization and graph stream construction on *AnoFusion*'s performance. Figure 3 shows how the average best F_1-score of *AnoFusion* changes with different values of hyperparameters. Specifically, we increase the size of the sliding window in data serialization, θ, from 10 to 120. From the experiment results, we can find that if θ is too large, it will contain too many seasonal variations and will struggle to reconstruct the current state; if θ is too small, the model will be unable to comprehensively learn the information from historical data, degrading *AnoFusion*'s performance. θ between 40 and 90 can lead to relatively good performance. Thus, we set $\theta = 60$.

We change the number of attention heads in GAT, H, from 2 to 10. An H of 6 yields the best performance. If H is too small, the performance of *AnoFusion* will slightly degrade due to the decrease in model size; if H is too large, more redundant information may be generated, interfering with the training of *AnoFusion* [38].

6 DISCUSSION

6.1 Lessons Learned

Collecting multimodal monitoring data in real-time. We utilize multimodal data to detect failures in instances. Ensuring the real time data quality of different modalities is essential for the deep learning models. From our real-world experience in Microsoft, we suggest leveraging the open-source monitoring systems or Azure Monitor[2] to build the data pipeline. For example, Prometheus[3] can be used to collect metrics. ELK (Elasticsearch, Logstash, and Kibana) Stack[4] are used to collect logs. Skywalking[5] can be used to collect traces. Additionally, 16 days of data are utilized for training. When the model training is completed, *AnoFusion* digests 10 minutes of data to perform real-time detection in the online detection stage, which is efficient and effective in practice.

Selection of evaluation metrics. In the online detection stage, *AnoFusion* adopts the EVT algorithm [34] to obtain the best F_1-score. In practice, however, operators may have varying preferences for precision and recall depending on the business type. For example, operators generally seek a high recall for the core services that provide online shopping services. They do not want to miss any potential failures that could negatively impact users' experience. Precision is often preferable in data analytic services. Operators

want to detect failures accurately rather than receive a large number of false alarms. Therefore, concentrating solely on F_1-score is not appropriate for all instances. In the future, we plan to provide an interface that allows operators to apply additional weights, valuing one of precision or recall more than the other.

6.2 Threat to Validity

Failure labeling. In our experiments, we use two datasets, one is public and another is collected from a real-world commercial bank. The ground truth labels are based on failure injection (D1) and manually checking failure reports by system operators (D2). Manually labeling anomalies may contain few noises. We believe that the labeling noises are minimal due to the extensive experience of operators. Furthermore, to reduce the impact of labeling noises, we adopt widely used evaluation metric to detect continuous failure segments instead of point-wise anomalies [26].

Granularity effect. The granularity of the monitoring data in our experiments is one minute, but this has no effect on the algorithm's effectiveness. We believe the algorithm can still work with finer or coarser-grained data without additional effort. The datasets in our experiments are still limited. We will experiment *AnoFusion* with a larger scale of system in the future.

Data modalities. Our work involves the utilization of three modalities of monitoring data. We believe that in a real-world scenario, as long as the modalities of the monitoring data are no less than two, the algorithm will function normally. Furthermore, if a failure manifests itself in only one type of monitoring data, *AnoFusion* will consider not only the correlation among historical multimodal data, but also the proportion of anomalous in all monitoring data to determine whether the instance fails.

7 CONCLUSION

Failure detection in the microservice systems is of great importance for service reliability. In this work, we propose *AnoFusion*, one of the first studies using multimodal data, *i.e.*, metrics, logs, and traces, to detect failures of instances in microservice systems robustly. Specifically, we first serialize the data of the three modalities and construct a heterogeneous graph structure. Then, GTN is utilized to update the heterogeneous graph, with GAT being used to capture significant features. Finally, we use GRU to predict the data pattern at the next moment. The deviation between the predicted and observed values is used as the failure scores. We apply *AnoFusion* on two microservice systems, which proves that it significantly improves the F_1-score for failure detection. We believe that the solution of applying multimodal data for failure detection will benefit more areas beyond microservice systems.

---

[2]https://azure.microsoft.com/en-us/products/monitor
[3]https://prometheus.io
[4]https://www.elastic.co/what-is/elk-stack
[5]https://skywalking.apache.org

REFERENCES

[1] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. 2020. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 3395–3404. https://doi.org/10.1145/3394486.3403392

[2] AWS. 2021. Summary of the AWS Service Event in the Northern Virginia (US-EAST-1) Region. (2021). https://aws.amazon.com/cn/message/12721/.

[3] Ailin Deng and Bryan Hooi. 2021. Graph Neural Network-Based Anomaly Detection in Multivariate Time Series. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 4027–4035. https://ojs.aaai.org/index.php/AAAI/article/view/16523

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[5] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1285–1298. https://doi.org/10.1145/3133956.3134015

[6] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. 2018. Metric selection and anomaly detection for cloud operations using log and metric correlation analysis. *J. Syst. Softw.* 137 (2018), 531–549. https://doi.org/10.1016/j.jss.2017.03.012

[7] Ying Fu, Meng Yan, Jian Xu, Jianguo Li, Zhongxin Liu, Xiaohong Zhang, and Dan Yang. 2022. Investigating and improving log parsing in practice. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1566–1577.

[8] Weihao Gao, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. 2017. Estimating Mutual Information for Discrete-Continuous Mixtures. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5986–5997. https://proceedings.neurips.cc/paper/2017/hash/ef72d53995bc4805684c9b61fa64a102-Abstract.html

[9] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. 2020. 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*. 1387–1397.

[10] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, Ilkay Altintas and Shiping Chen (Eds.). IEEE, 33–40. https://doi.org/10.1109/ICWS.2017.13

[11] Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, et al. 2022. An empirical study of log analysis at Microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1465–1476.

[12] Vladimir Iashin and Esa Rahtu. 2020. Multi-modal Dense Video Captioning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. Computer Vision Foundation / IEEE, 4117–4126. https://doi.org/10.1109/CVPRW50498.2020.00487

[13] Chen Jia and Yue Zhang. 2020. Multi-Cell Compositional LSTM for NER Domain Adaptation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 5906–5917. https://doi.org/10.18653/v1/2020.acl-main.524

[14] Ziyu Jia, Youfang Lin, Jing Wang, Zhiyang Feng, Xiangheng Xie, and Caijie Chen. 2021. HetEmotionNet: Two-Stream Heterogeneous Graph Recurrent Neural Network for Multi-modal Emotion Recognition. In *MM '21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, Heng Tao Shen, Yueting Zhuang, John R. Smith, Yang Yang, Pablo Cesar, Florian Metze, and Balakrishnan Prabhakaran (Eds.). ACM, 1047–1056. https://doi.org/10.1145/3474085.3475583

[15] Soopil Kim, Sion An, Philip Chikontwe, and Sang Hyun Park. 2021. Bidirectional RNN-based Few Shot Learning for 3D Medical Image Segmentation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 1808–1816. https://ojs.aaai.org/index.php/AAAI/article/view/16275

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[17] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and Scalable Framework for Automated Time-series Anomaly Detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM, 1939–1947. https://doi.org/10.1145/2783258.2788611

[18] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Zhekang Chen, Wenchi Zhang, Xiaohui Nie, Kaixin Sui, and Dan Pei. 2021. Practical Root Cause Localization for Microservice Systems via Trace Analysis. In *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*. IEEE, 1–10. https://doi.org/10.1109/IWQOS52092.2021.9521340

[19] Zhe Li, Peisong Wang, Hanqing Lu, and Jian Cheng. 2019. Reading selectively via Binary Input Gated Recurrent Unit. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 5074–5080. https://doi.org/10.24963/ijcai.2019/705

[20] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: Towards Practical and Automatic Anomaly Detection Through Machine Learning. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring (Eds.). ACM, 211–224. https://doi.org/10.1145/2815675.2815679

[21] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, Marco Vieira, Henrique Madeira, Nuno Antunes, and Zheng Zheng (Eds.). IEEE, 48–58. https://doi.org/10.1109/ISSRE5003.2020.00014

[22] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2022. UniParser: A Unified Log Parser for Heterogeneous Log Data. In *Proceedings of the Web Conference (WWW '22)*. ACM, 1893–1901. https://doi.org/10.1145/3485447.3511993

[23] Minghua Ma, Yudong Liu, Yuang Tong, Haozhe Li, Pu Zhao, Yong Xu, Hongyu Zhang, Shilin He, Lu Wang, Yingnong Dang, Saravanakumar Rajmohan, and Qingwei Lin. 2022. An Empirical Investigation of Missing Data Handling in Cloud Node Failure Prediction. In *Proceedings of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 1453 – 1464.

[24] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing Root Causes of Intermittent Slow Queries in Cloud Databases. In *PVLDB*, Vol. 13. 1176–1189.

[25] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. 2021. Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 413–426. https://www.usenix.org/conference/atc21/presentation/ma

[26] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. 2018. Robust and Rapid Adaption for Concept Drift in Software System Anomaly Detection. In *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018, Memphis, TN, USA, October 15-18, 2018*, Sudipto Ghosh, Roberto Natella, Bojan Cukic, Robin S. Poston, and Nuno Laranjeiro (Eds.). IEEE Computer Society, 13–24. https://doi.org/10.1109/ISSRE.2018.00013

[27] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 4739–4745. https://doi.org/10.24963/ijcai.2019/658

[28] Sasho Nedelkoski, Jorge S. Cardoso, and Odej Kao. 2019. Anomaly Detection and Classification using Distributed Tracing and Deep Learning. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019*. IEEE, 241–250. https://doi.org/10.1109/CCGRID.2019.00038

[29] Sasho Nedelkoski, Jorge S. Cardoso, and Odej Kao. 2019. Anomaly Detection from System Tracing Data Using Multimodal Deep Learning. In *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*, Elisa

Bertino, Carl K. Chang, Peter Chen, Ernesto Damiani, Michael Goul, and Katsunori Oyama (Eds.). IEEE, 179–186. https://doi.org/10.1109/CLOUD.2019.00038

[30] Rodolfo Picoreti, Alexandre Pereira do Carmo, Felippe Mendonça de Queiroz, Anilton Salles Garcia, Raquel Frizera Vassallo, and Dimitra Simeonidou. 2018. Multilevel Observability in Cloud Orchestration. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2018, Athens, Greece, August 12-15, 2018.* IEEE Computer Society, 776–784. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00134

[31] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019,* Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 3009–3017. https://doi.org/10.1145/3292500.3330680

[32] Tulika Saha, Aditya Prakash Patra, Sriparna Saha, and Pushpak Bhattacharyya. 2020. Towards Emotion-aided Multi-modal Dialogue Act Classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020,* Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 4361–4372. https://doi.org/10.18653/v1/2020.acl-main.402

[33] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 19 (jul 2017), 21 pages. https://doi.org/10.1145/3068335

[34] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouët. 2017. Anomaly Detection in Streams with Extreme Value Theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017.* ACM, 1067–1075. https://doi.org/10.1145/3097983.3098144

[35] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019,* Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 2828–2837. https://doi.org/10.1145/3292500.3330672

[36] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. 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. https://openreview.net/forum?id=rJXMpikCZ

[37] S. Vijayanand and S. Saravanan. 2022. A deep learning model based anomalous behavior detection for supporting verifiable access control scheme in cloud servers. *J. Intell. Fuzzy Syst.* 42, 6 (2022), 6171–6181. https://doi.org/10.3233/JIFS-212572

[38] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers,* Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 5797–5808. https://doi.org/10.18653/v1/p19-1580

[39] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-Cause Metric Location for Microservice Systems via Log Anomaly Detection. In *2020 IEEE International Conference on Web Services (ICWS).* 142–150. https://doi.org/10.1109/ICWS49710.2020.00026

[40] Lianghao Xia, Chao Huang, Yong Xu, Peng Dai, Xiyue Zhang, Hongsheng Yang, Jian Pei, and Liefeng Bo. 2021. Knowledge-Enhanced Hierarchical Graph Transformer Network for Multi-Behavior Recommendation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021.* AAAI Press, 4486–4493. https://ojs.aaai.org/index.php/AAAI/article/view/16576

[41] Shi Ying, Bingming Wang, Lu Wang, Qingshan Li, Yishi Zhao, Jianga Shang, Hao Huang, Guoli Cheng, Zhe Yang, and Jiangyi Geng. 2021. An Improved KNN-Based Efficient Log Anomaly Detection Method with Automatically Labeled Samples. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 3 (2021), 1–22.

[42] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021,* Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 3087–3098. https://doi.org/10.1145/3442381.3449905

[43] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. 2019. Graph Transformer Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada,* Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 11960–11970. https://proceedings.neurips.cc/paper/2019/hash/9d63484abb477c97640154d40595a3bb-Abstract.html

[44] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019,* Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 807–817. https://doi.org/10.1145/3338906.3338931

[45] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2022. Deep Learning on Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.* 34, 1 (2022), 249–270. https://doi.org/10.1109/TKDE.2020.2981333

[46] Nengwen Zhao, Junjie Chen, Zhaoyang Yu, Honglin Wang, Jiesong Li, Bin Qiu, Hongyu Xu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. Identifying bad software changes via multimodal anomaly detection for online service systems. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021,* Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 527–539. https://doi.org/10.1145/3468264.3468543

[47] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2021. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Trans. Software Eng.* 47, 2 (2021), 243–260. https://doi.org/10.1109/TSE.2018.2887384