Identifying Performance Issues in Cloud Service Systems Based on Relational-Temporal Features

WENWEI GU, The Chinese University of Hong Kong, Hong Kong SAR
JINYANG LIU, The Chinese University of Hong Kong, Hong Kong SAR
ZHUANGBIN CHEN, Sun Yat-sen University, China
JIANPING ZHANG, The Chinese University of Hong Kong, Hong Kong SAR
YUXIN SU, Sun Yat-sen University, China
JIAZHEN GU*, The Chinese University of Hong Kong, Hong Kong SAR
CONG FENG, Huawei Cloud Computing Technology Co., Ltd, China
ZENGYIN YANG, Huawei Cloud Computing Technology Co., Ltd, China
YONGQIANG YANG, Huawei Cloud Computing Technology Co., Ltd, China
MICHAEL R. LYU, The Chinese University of Hong Kong, Hong Kong SAR

Cloud systems, typically comprised of various components (*e.g.*, microservices), are susceptible to performance issues, which may cause service-level agreement violations and financial losses. Identifying performance issues is thus of paramount importance for cloud vendors. In current practice, crucial metrics, *i.e.*, key performance indicators (KPIs), are monitored periodically to provide insight into the operational status of components. Identifying performance issues is often formulated as an anomaly detection problem, which is tackled by analyzing each metric independently. However, this approach overlooks the complex dependencies existing among cloud components. Some graph neural network-based methods take both temporal and relational information into account, however, the correlation violations in the metrics that serve as indicators of underlying performance issues are difficult for them to identify. Furthermore, a large volume of components in a cloud system results in a vast array of noisy metrics. This complexity renders it impractical for engineers to fully comprehend the correlations, making it challenging to identify performance issues accurately. To address these limitations, we propose Identifying Performance Issues based on Relational-Temporal Features (ISOLATE), a learning-based approach that leverages both the relational and temporal features of metrics to identify performance issues. In particular, it adopts a graph neural network with attention to characterizing the relations among metrics and extracts long-term and multi-scale temporal patterns using a GRU and a convolution network, respectively. The learned graph attention weights can be further used to localize the correlation-violated metrics. Moreover, to relieve the impact of noisy data, ISOLATE utilizes a positive unlabeled learning strategy that tags pseudo labels based on a small portion of confirmed negative examples. Extensive evaluation on both public and

*Corresponding author.

Authors' Contact Information: Wenwei Gu, The Chinese University of Hong Kong, Hong Kong SAR, wwgu21@cse.cuhk.edu.hk; Jinyang Liu, The Chinese University of Hong Kong, Hong Kong SAR, jyliu@cse.cuhk.edu.hk; Zhuangbin Chen, Sun Yat-sen University, China, chenzhb36@mail.sysu.edu.cn; Jianping Zhang, The Chinese University of Hong Kong, Hong Kong SAR, jpzhang@cse.cuhk.edu.hk; Yuxin Su, Sun Yat-sen University, China, suyx35@mail.sysu.edu.cn; Jiazhen Gu, The Chinese University of Hong Kong, Hong Kong SAR, jiazhengu@cuhk.edu.hk; Cong Feng, Huawei Cloud Computing Technology Co., Ltd, China, fengcong5@huawei.com; Zengyin Yang, Huawei Cloud Computing Technology Co., Ltd, China, yangzengyin@huawei.com; Yongqiang Yang, Huawei Cloud Computing Technology Co., Ltd, China, yangyongqiang@huawei.com; Michael R. Lyu, The Chinese University of Hong Kong, Hong Kong SAR, lyu@cse.cuhk.edu.hk.

ACM XXXX-XXXX/2024/11-ART
https://doi.org/XXXXXXX.XXXXXXX

industrial datasets shows that ISOLATE outperforms all baseline models with 0.945 F1-score and 0.920 Hit rate@3. The ablation study also proves the effectiveness of the relational-temporal features and the PU-learning strategy. Furthermore, we share the success stories of leveraging ISOLATE to identify performance issues in Huawei Cloud, which demonstrates its superiority in practice.

CCS Concepts: • **Software and its engineering → Software post-development issues**.

Additional Key Words and Phrases: Performance Issue Identification, Multivariate Monitoring Metrics, Anomaly Detection, Cloud Reliability, Cloud Service System

ACM Reference Format:
Wenwei Gu, Jinyang Liu, Zhuangbin Chen, Jianping Zhang, Yuxin Su, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R. Lyu. 2024. Identifying Performance Issues in Cloud Service Systems Based on Relational-Temporal Features. 1, 1 (November 2024), 31 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Cloud computing has surged in popularity in recent years. Large-scale cloud vendors, *e.g.*, Microsoft Azure, Amazon Web Services, and Google Cloud Platform, provide customers with various services over the Internet [56]. Due to the scale and complexity, performance issues are inevitable [30] in cloud systems. Such performance issues may degrade overall availability and increase service response time, thus causing SLA (Service Level Agreement) violations and substantial economic losses [2, 39]. Therefore, identifying performance issues accurately is a critical task during the maintenance of cloud systems [27].

In current practice, cloud vendors typically collect crucial metrics (*i.e.*, Key Performance Indicators, KPI), such as CPU utilization and network latency, and then analyze the collected metrics to identify performance issues. Simple as the process might seem, it is a non-trivial task. In particular, a cloud system is typically vast in scale and consists of tremendous software and hardware components (*e.g.*, microservices, and virtual machines and servers) [20, 54, 80]. Each component may have tens of metrics to be collected in the backend monitoring system, resulting in a large volume of monitoring metrics [52, 71]. Since analyzing tremendous data manually is labor-intensive and error-prone [12], automatic approaches that help on-call engineers identify performance issues are preferred.

In the literature, the performance issue identification problem is generally formulated as an anomaly detection problem based on multivariate metrics [5, 59, 76]. Some existing approaches [62, 65, 87] detect anomalies based on individual metrics, *i.e.*, they only consider the temporal abnormal patterns on individual metrics. However, modern cloud service typically consists of various components (*e.g.*, storage, computing, middleware), whose corresponding monitoring metrics have complicated inter-dependencies [13, 45, 59]. Take disk I/O operations per second and network throughput as an example: Under normal operating conditions, there is typically a correlation between these two metrics; as disk I/O operations increase due to higher data processing and storage demands, network throughput also increases as more data is being transferred to and from the storage systems. However, if a performance issue arises, this correlation can be violated. For instance, if disk I/O remains high while network throughput suddenly drops significantly, it could indicate a network bottleneck, a network gateway failure, or a network misconfiguration. This deviation from the expected correlation between disk I/O and network throughput can thus be a strong indicator of underlying performance issues in the cloud service.

To alleviate the drawback of overlooking correlations, several graph neural network-based approaches that take the spatial-temporal dependency between multiple metrics [9, 16, 21, 44, 88] are proposed. Though these approaches consider the correlation between metrics, these correlations are typically embedded within the latent feature representation in an implicit manner. In other words, these approaches essentially detect temporal anomalies in these latent representations. In contrast, we propose a correlation violation-aware approach that incorporates the correlation violation explicitly. To effectively detect correlation-violated performance problems, it is essential to account for the correlation violation among metrics as performance anomaly criteria.

However, there are four challenges in identifying performance issues through correlation violations. Firstly, it is hard to automatically and effectively model the complicated correlations among a variety of metrics. A cloud application could comprise tens of microservices. Each microservice may still have tens of metrics, leading to a large number of metrics. Moreover, the correlation between metrics is complicated. Even experienced engineers cannot comprehensively clarify the relations among different metrics. Secondly, even with these correlations extracted, the process of accurately identifying the correlation violations from these intricate inter-metric correlations and then determining the happening of performance issues is a non-trivial task. Thirdly, due to the huge volume of metrics, it is still time-consuming for engineers to investigate the underlying issues simply given a binary (*i.e.*, normal and abnormal) result. Automatically pinpointing the anomalous metrics is required. Finally, metric data from large-scale production systems are noisy [11], *i.e.*, mild performance issues without obvious anomalies in the metric data. This can blur the distinction between normal and anomalous instances, leading to more false positives and false negatives. Unfortunately, it is infeasible to annotate a huge volume of noisy data manually, while simply neglecting such noises may lead to unsatisfying results.

To address these challenges, we propose Identifying Performance Issues Based on Relational-Temporal Features (ISOLATE), an automated approach to identify performance issues through both capturing the violation of relational features and detecting temporal anomalous features of metrics. To solve the challenge of intricate correlations, ISOLATE utilizes graph neural networks to capture the complicated relational features among metrics explicitly, namely, it employs the graph attention mechanism to characterize the correlations between metrics. To effectively identify the correlation violation between metrics, relational embedding, which is uncoupled from the temporal information of raw metrics, is utilized and fed to downstream anomaly detection components. The correlation-violation metrics are highlighted to provide insights to software reliability engineers like the root cause of the performance issue. Furthermore, to relieve the impact of noisy data, Positive Unlabeled learning (PU learning) is adopted, which finds positive samples and updates the models iteratively. Since PU learning produces both negative and positive samples, while traditional Variational Auto Encoder-based (VAE-based) methods can only take negative examples, ISOLATE employs a novel LC-VAE (Label-Conditional VAE) to distinguish normal and abnormal patterns from the labeled inputs effectively.

To evaluate the performance of ISOLATE, we conduct extensive experiments on a widely-used public dataset and two industrial datasets collected from Huawei Cloud. The experimental results demonstrate that compared with state-of-the-art baselines, ISOLATE achieves the best performance issue identification accuracy with an average F1 score of 0.945. ISOLATE also provides anomalous metrics localization with a Hit rate@3 of 0.920. Moreover, we also conduct ablation studies to validate the effectiveness of our design. A case study with two real cases in Huawei Cloud further shows the practical usefulness of our proposed ISOLATE.

We summarize the main contributions of this work as follows:

- We propose an end-to-end model that captures the relational violation among monitoring metrics explicitly, offering a more precise way to identify correlation-violated performance anomalies. The violated correlations are utilized to localize the root cause of the performance anomaly. Besides, to alleviate the negative effect of concealed noise in training data, we adopt positive unlabeled learning (PU Learning) on metrics performance anomaly detection and propose the Label-Conditional Variational Autoencoder (LC-VAE) that works seamlessly with PU Learning.

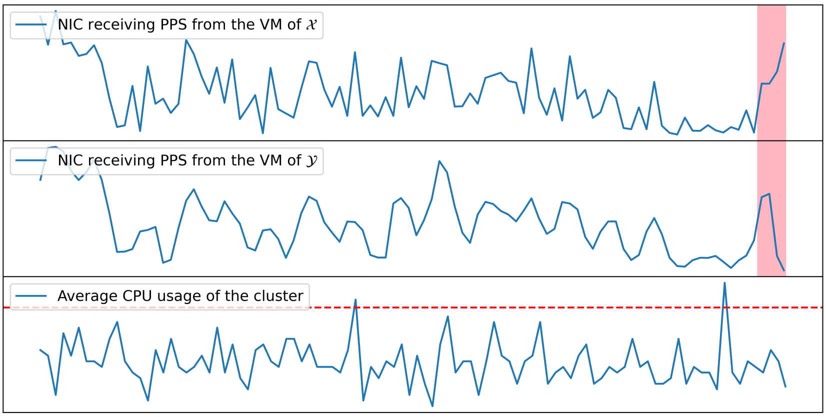


Fig. 1. A real-world example of performance anomaly

- We conduct extensive evaluations of ISOLATE on two industrial datasets collected from large-scale online service systems of Huawei Cloud and a publicly available dataset. The results demonstrate that ISOLATE outperforms eighteen state-of-the-art methods.
- We have successfully deployed ISOLATE into the troubleshooting system of Huawei Cloud. The success stories of our deployment confirm the applicability and effectiveness of our method.

2 Background

In this section, we first introduce the background knowledge about performance issue detection in modern cloud systems. Then, we present an example of an industrial scenario that motivates this work. Finally, we summarize some typical performance issues due to the violation of the correlation of monitoring metrics, which aims to provide a comprehensive understanding of the performance issues and how the correlation of metrics plays a crucial role in detecting and diagnosing performance issues.

2.1 Monitoring Metrics in Cloud Service Systems

In recent years, cloud service systems have gained significant attention due to their ability to provide scalable, on-demand resources and services. Typically, coupled multivariate metrics are collected in run-time to monitor the overall status of the cloud service systems. The collected metrics provide insights into the performance of logical and physical resources within the system, allowing operators to identify and address performance issues before they lead to service disruptions. However, due to the complex inter-dependencies between system components [19], these metrics are often strongly correlated with each other, reflecting the interconnected nature of the system that makes it challenging to isolate and identify specific performance issues. For example, the performance of a virtual machine may be influenced by the workload placed on the underlying physical server, and the performance of a microservice may depend on the performance of other microservices it interacts with. As a result, the metrics collected from different components can exhibit strong correlations with each other, reflecting the complex inter-dependencies within the system.

2.2 Performance Issues due to Correlation Violation

Performance issues have emerged as a primary concern, potentially undermining the effectiveness of cloud service systems. Some typical factors contributing to these performance issues are resource overload and network latency. Resource overload occurs when the demand for resources exceeds

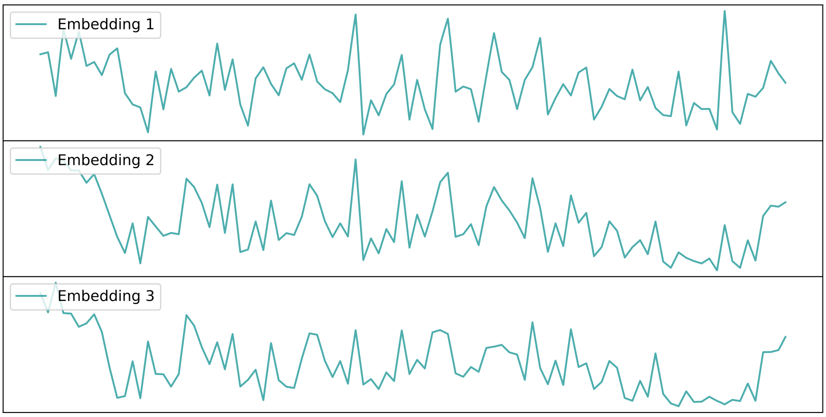


Fig. 2. The latent embedding of graph attention layer

the available capacity, resulting in performance degradation and potential service disruptions [49]. Network latency, exacerbated by the distributed nature of cloud systems, can result in reduced application responsiveness, impacting the overall user experience [4]. While these performance issues can be detected by considering single metrics, there are other performance issues that are caused by the violation of correlations between different system metrics, which do not arise from a single metric exceeding a threshold but from a violation in the expected correlation between metrics. Considering the intrinsic correlations between metrics in cloud service systems, accurately capturing and localizing these correlation violation metrics is of great significance to performance issue identification. By localizing anomalous metrics, we gain valuable insights into the specific metrics that deviate from normal behavior (*i.e.*,violation of correlation to other metrics). Through accurate metric localization, system engineers can swiftly identify the entities that undergo performance anomalies, facilitating prompt troubleshooting and proactive issue resolution.

An industrial case in the online service system of Huawei Cloud is shown in Figure 1. Three metrics of the elastic load balance (ELB) service are shown in the figure for convenience of presentation. Specifically, the first metric represents the network interface card (NIC) receiving packets per second (PPS) from a virtual machine running microservice $\mathcal{X}$, and the second metric represents the NIC receiving PPS from a virtual machine running microservice $\mathcal{Y}$, which is the downstream microservice that processes the outputs of $\mathcal{X}$. The third metric is the average CPU usage of all virtual machines of the ELB service. We can observe that the variation tendencies between the first and second metrics are somehow consistent during the anomaly-free period. The third metric is used to monitor the resource usage of the service and can raise alerts when there are resource overload issues. However, with the existence of load balance [18], a sudden spike in CPU usage will not necessarily lead to a performance issue. Thus, if we trigger alerts based on pre-defined thresholds (as the red dashed line shows), many false alarms will be reported and cause an alert storm that aggravates the burden of engineers [89].

The red areas in Figure 1 denote a confirmed network congestion issue by engineers. This issue is caused by a network device failure affecting communication between virtual machines running microservices $\mathcal{X}$ and $\mathcal{Y}$. As a result, the NIC receiving PPS from the virtual machine running $\mathcal{Y}$ suddenly drops even if the NIC receiving PPS from the virtual machine running $\mathcal{X}$ increases. In this case, the correlation between the first and second metrics is critical to rapidly identifying this issue.

Some of the typical performance issues due to correlation violation are listed in Table 1. Specifically, we take the memory leak as another example to illustrate how the correlation violation reflected

Table 1. A List of Typical Performance Issues Caused by Correlation Violation

Performance Issues	Metrics	Description
Disk Read/Write Delay	CPU I/0 wait time Disk I/O	High CPU I/O wait time with low Disk I/O indicates potential disk failure or disk controller problems
Memory Leak	Memory slab size Garbage collection time	High memory slab size with high garbage collection time indicates potential memory leak
API Server Issue	API requests CPU usage	High API requests processed with low CPU usage indicates a potential problem with the API server or an overly efficient request handling mechanism
Network Congestion	NIC *A* receiving PPS NIC *B* receiving PPS	High NIC *A* receiving PPS with low receiving PPS of NIC *B* or vice versa can indicate potential network congestion
Hardware Interrupt Issue	Interrupt requests CPU usage	High CPU interrupt requests with low CPU usage indicates potential hardware issue or inefficient interrupt handling
Software Interrupt Issue	Soft interrupt requests CPU usage	High CPU soft interrupt request with low CPU usage indicates potential software issue or inefficient interrupt handling
Network Buffering Issue	NAT gateway PPS Memory usage	High NAT gateway PPS with low memory usage indicates potential efficient network buffering mechanism or a network issue
Database Indexing Issue	Database query Memory usage	Slow database query latency with low memory usage indicates potential issues with database indexing or query optimization

on system monitoring metrics can imply performance issues. In cloud service systems, efficient memory management is of great importance and has a direct impact on both the scalability and the operational costs of the cloud environment [48]. A crucial part of the memory management strategy is using slab memory allocation [31], which are pre-allocated extents in persistent memory and containers of free blocks [14] that aid in efficient memory usage. While the garbage collection manages application memory automatically in virtual machines [15], works to obtain the lifetime of the data objects and then allocates and releases memory space accordingly [63]. Memory leaks occur when a program fails to track and release allocated memory back to the system after it's no longer needed [72]. Usually, it is observed that garbage collection time or slab size increases under heavy workloads. An increase in slab size, with a relatively stable garbage collection time, could indicate that the system is effectively managing memory by creating new slabs to handle the increased workload. Similarly, a rise in garbage collection time, with a relatively stable slab size, might suggest that the system reuses existing memory efficiently, resulting in more objects being created and, hence, more garbage to collect. Generally speaking, software systems should efficiently manage memory under heavy workloads, either by creating new memory blocks (indicated by increased slab size) or working harder to clear up unused objects (indicated by increased garbage collection time). Thus, these two cases are not typically considered performance anomalies. However, when both the slab size and garbage collection time consistently increase, even when the workload has reduced, it could suggest a memory leak because memory is being allocated faster than it's being released. It should be noted that a heavy workload can cause these two metrics to increase, hinting at a memory leak when such a leak exists. However, a heavy workload itself will not cause a memory leak, as the root cause of memory leaks lies in programming errors.

It is worth noting that a straightforward way to identify this issue is to build a new metric that combines the first metric and the second metric. However, since a cloud service system typically has a variety of metrics, it is infeasible for engineers to design such combined metrics comprehensively. To alleviate this problem, some GNN-based methods like Graph Attention (GAT) attempt to embed these correlations into latent representations. Nevertheless, we have identified that these methods incorporate correlation information into the latent feature representation implicitly through the combination of all metrics with attention weights. Then the temporal anomalies in these latent representations are detected. The GAT embedding of the example in Figure 1 is shown in Figure 2, where we can observe that the correlation violation between the first and second metrics is even impaired, as the uptrend of the first metric and the downtrend of the second metric are counteracted. Thus, it is crucial to isolate the correlation from the temporal information to detect violation-related performance anomalies.

3 METHODOLOGY

In this section, we present ISOLATE, an approach for identifying performance issues based on multivariate metrics in cloud systems. First, we will give a formal definition of the problem. Then, we introduce the overview of ISOLATE and illustrate the design details of our proposed relational-temporal embedding, Label-Conditioned Variational Auto-encoder (LC-VAE), and a novel positive unlabeled strategy. Eventually, we will introduce how to localize the problematic metrics with the correlations obtained in the previous stage.

3.1 Problem Formulation

Our objective is to identify performance issues from a variety of monitoring metrics by identifying when the performance anomalies happen and pinpointing the metrics that exhibit temporal or relational anomalies. Specifically, a group of metrics can be seen as a multivariate time series $X \in R^{N\times M}$, where N denotes the number of observations collected at an equal interval, *i.e.*, the length of a time series [65] and M is the number of metrics. $x_t = [x_t^1, x_t^2, ..., x_t^M]$ is an M-dimensional vector [26] that reflects the running status of the system at timestamp t. While the N-dimensional vector $x^k = [x_1^k, x_2^k, ..., x_N^k]$ is the k^{th} metric during the whole monitoring period. In addition, a sliding window of historical values $x_{t-c:t}^k = [x_{t-c+1}^k, x_{t-c+2}^k, ..., x_t^k]$ is used for modeling the pattern of a current observation, where c is the length of the sliding window.

To determine whether there is an occurrence of performance issues at observation x_t, the anomaly score $s_t \in [0, 1]$ that represents the degree of being anomalous for each $x_{t-c:t}$ is calculated. Then, the anomaly result can be obtained by comparing the anomaly score against a pre-defined threshold θ. If $s_t > \theta$, the approach will predict the observation x_t as an anomaly. However, it is still unclear to engineers how the anomaly happens. Thus, a kind of anomaly interpretation can be achieved through anomalous metrics localization, *i.e.*, pinpointing a set of metrics $\{x_{k_1}, x_{k_2}, ...x_{k_r}\}$, that is helpful for engineers to find the monitored components that are related to anomaly by the degree of deviation of temporal pattern or break of correlation with other metrics, where r is the number of metrics that are recommended as the anomalous metrics.

Normalization is performed on each individual metric to unify the range of all metrics and improve the robustness of our model. We normalize the metrics with the maximum and minimum values first:

$$x_{pre}^k = \frac{x^k - min(x^k)}{max(x^k) - min(x^k)} \tag{1}$$

Where $max(x^k)$ and $min(x^k)$ represent the maximum and the minimum value of the training set, are computed within the training data and will be used for testing data. For simplicity, we omit the "pre" subscript in the following elaboration.

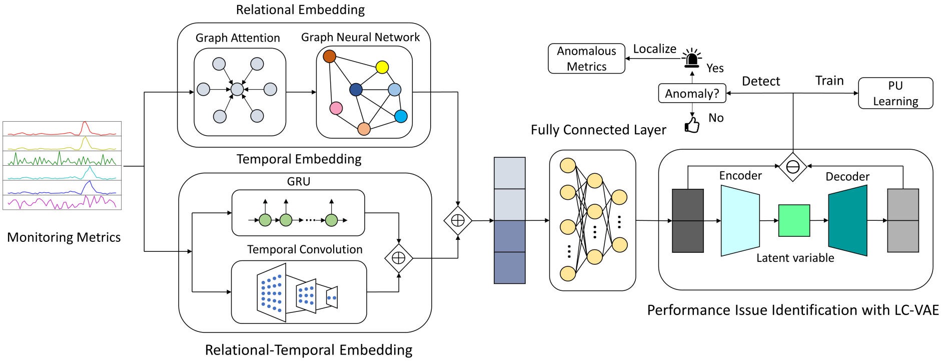


Fig. 3. The Overview of the Proposed Method ISOLATE

3.2 Overview

We propose ISOLATE, an automated method that learns correlations among metrics, detects performance anomalies, and locates anomalous metrics. The overview of ISOLATE is shown in Figure 3, which contains two main parts: the relational-temporal embedding part and the performance issue identification with the LC-VAE part. Specifically, since both abnormal temporal patterns and correlation violations between metrics can indicate performance issues, in the relational-temporal embedding part, ISOLATE captures the relational and temporal patterns from the original metrics (Section 3.3). In particular, due to the lack of information about the correlation among metrics, a complete graph is constructed, then ISOLATE employs graph attention to extract the correlation among metrics. ISOLATE also captures the temporal pattern of each metric through GRU and temporal convolution. In the performance issue identification part, given the inevitable presence of background noise within the data, we propose to use the positive unlabeled learning (PU Learning) strategy during the training phase of ISOLATE (Section 3.4.1), which identifies positive samples in unlabeled training data, avoiding the impact of noisy data. As PU learning leads to pseudo-labeled data, a novel label-conditional-VAE (LC-VAE) is adopted to distinguish anomalies from normal patterns (Section 3.4.2). Unlike existing VAE-based methods that solely learn from normal samples, the LC-VAE can learn features from normal and abnormal samples, achieving better performance. Upon the detection of an anomaly, the correlation learned from relational-temporal embedding can aid in localizing the correlation-violating metrics (Section 3.5).

3.3 Relational-Temporal Embedding

The relational-temporal embedding part takes a group of metrics as inputs. Relational embedding is designed to extract the intrinsic dependencies between metrics and embed the dependencies as a feature vector. Temporal embedding is used to obtain the temporal patterns of metrics as another feature vector because metrics are time series.

3.3.1 Relational Embedding. Specifically, for multivariate metrics with size $M \times N$, we can treat each metric x_i, $(i = 1, 2, ...M)$ as a feature vector. The correlations between nodes can be depicted by an adjacency matrix $A \in R^{M \times M}$. Since we don't have prior knowledge about the correlation between different metrics, we should construct a fully connected graph. We then adopt graph attention networks (GAT) [7] to learn the correlation between metrics. The attention score is calculated as follows:

$$a_{ij} = \frac{\exp(p^T \text{LeakyReLU}(w \cdot (x_i \oplus x_j)))}{\sum_{k=1}^{M} \exp(p^T \text{LeakyReLU}(w \cdot (x_i \oplus x_k)))} \tag{2}$$

The symbol $\oplus$ represents the concatenation operator between two metrics x_i and x_j, $w \in R^{2N \times d}$ is a matrix of learnable parameters to aggregate the two features. After a nonlinear activation function LeakyReLU [75], another learnable vector $p \in R^d$ is applied. To make the training process more robust and reduce the impact of noise, a threshold α is set to make the adjacency matrix a sparse binary matrix. In other words, the edge between two nodes will be removed if the attention score is below the threshold α. Then, a widely used graph convolution layer [35] is adopted. The formula of graph convolution is shown as follows:

$$\hat{h}^{(l+1)} = \sigma(\widetilde{D}_l^{-\frac{1}{2}} \widetilde{A}_l \widetilde{D}_l^{-\frac{1}{2}} h^{(l)} \Theta_l) \tag{3}$$

where σ is the ReLU activation function and $\widetilde{A_l} = A_l + I$ is the adjacency matrix at the layer l, $\widetilde{D} \in R^{M \times M}$ is the degree matrix of $\widetilde{A_l}$, $h^{(l)}$ is the output representation of the hidden layer l and $\Theta_l \in R^{M \times F}$ is a learnable parameter. Due to the limitation of space, we only show one layer in Figure 3.

We further apply graph pooling layers between the graph convolution layers to reduce the number of parameters, which can also avoid overfitting. Specifically, as proposed by [38], self-attention [67] is utilized to focus more on important features and less on unimportant features. Thus, self-attention scores can be obtained by using another graph convolution. After obtaining the attention scores Z, a portion of the nodes and features will be reserved according to the scores. A hyperparameter k refers to the pooling ratio, and the corresponding nodes with the top $\lfloor kM \rfloor$ value of Z will be retained.

Readout layer [8] is useful to aggregate all node features and get a summarized representation, which is used to output the relational embedding. The output of the readout layer is as follows:

$$r_l = \frac{1}{N_l} \sum_{n=1}^{N_l} h_n^{(l)} \oplus \max_{n=1}^{N_l} h_n^{(l)} \tag{4}$$

where N_l denotes the node number of layer l, $h_n^{(l)}$ is the n^{th} node feature of $h^{(l)}$ and $\oplus$ is concatenation operator. The outputs of each layer will go through readout layers and will be added up as the output of the relational embedding module because the features of different graphs with different sparsities can be combined.

3.3.2 Temporal Embedding. Existing metrics anomaly detection works [26, 87] utilize LSTM to acquire sequential information as Long-term temporal dependency inherently exists in monitoring metrics [25]. However, LSTM suffers from the gradient vanishing problem incurred by long-time lags [17]. To overcome the drawbacks of LSTM, we apply a GRU to capture the sequential information t_g, especially the long-term pattern in the metrics. Then, global average pooling layers are applied on the time series dimension of the output of GRU to get the temporal embedding.

Temporal convolution is useful for capturing the multi-scale temporal information of metrics. Unlike the existing methods that embed metrics with only recurrent neural networks, we also deploy causal convolution implemented by shifting the output of the 1D convolution. To further increase the receptive field of the convolutions, we use dilated convolutions. Dilated convolution is equivalent to filling the convolution kernel with zero padding so as to get a larger convolutional filter. Thus, we adopt dilated causal convolution (DC convolution) [51] to extract the temporal embedding of the metrics as it has advantages over the original convolutional operation with a larger receptive field, which is beneficial to our temporal embedding module as it can capture the behaviors of monitoring metrics at multi-scale. A block that consists of DC convolution, batch normalization [28], and activation function (*i.e.* ReLU) is utilized to form the temporal convolution. Eventually, the temporal embedding will be concatenated to the relational embedding and go through a fully connected layer to get the relational-temporal embedding.

3.4 Performance Issues Identification with LC-VAE

With the extracted relational-temporal embedding, we then use a novel Label Conditional VAE to detect performance issues. Unlike traditional autoencoder-based methods [5, 65, 76, 78, 93] that only take normal samples as input to capture the distribution of metrics, our proposed Label Conditional VAE takes the label obtained from positive unlabeled learning as a part of the input to further help the model differentiate anomalies from normal data because there exist some mild performance issues that are ignored by engineers in training data.

3.4.1 Positive Unlabeled Learning. Unsupervised methods assume that the training data is anomaly-free. However, there are inevitably some unlabeled anomalies that are ignored by engineers. This assumption can lead to a decline in the overall accuracy as the presence of these hidden anomalies can skew the model's understanding of normal behavior. In our scenario, we only have a small portion of negative data (representing normal, anomaly-free metrics) with high confidence, but we don't have positive data (representing anomalous samples) because finding positive from a large number of unlabeled samples is like looking for a needle in the ocean.

To tackle this, ISOLATE tries to find out the anomalous samples using the idea of positive unlabeled learning (PU learning) [36]. Specifically, as shown in Figure 4, a small amount of negative samples (around 5%) labeled by engineers is utilized for training the model. These labels are obtained by initially randomly selecting 5% of the entire training dataset. Engineers then meticulously filter out the noise in these samples and label them as true negatives. This manual labeling process is feasible due to the high confidence in the selected data and can be completed within a few minutes. Consequently, we incorporate human expertise into our method.

With the model trained on these labeled negatives, the remaining unlabeled training data can be predicted. Intuitively, the anomalous samples concealed in the training data are hard to reconstruct with this model and, thus, have a high anomaly score. After obtaining the anomaly score, the samples with an anomaly score that exceeds a pre-defined threshold β will be labeled as positive. All the data with pseudo labels are used to update the model. Finally, this updated model will be used to detect performance issues from metrics.

3.4.2 Label-Conditional VAE. Though the posterior of the distribution $p_\theta(z|y,e)$ is critical for training and prediction of the model, it is hard to obtain. Thus, the variational inference is used to fit a neural network as the approximation posterior $q_\phi(z|y,e)$. Suppose the prior of the latent variable Z is Gaussian distribution $\mathcal{N}(\mathbf{0},\mathbf{1})$ and y is the true label of the input sliding windows. Then both posteriors of e and z are chosen to be Gaussian distribution: $p_\theta(e|y,z) = \mathcal{N}(\mu_\theta(z), \sigma_\theta^2(z))$ and $q_\phi(z|y,e) = \mathcal{N}(\mu_\phi(e), \sigma_\phi^2(e))$, where $\mu_\theta(z)$, $\sigma_\theta(z)$ and $\mu_\phi(e)$, $\sigma_\phi(e)$, are the means and standard deviations of input embedding and latent variable. The input embedding e will be concatenated with the one-hot label vector y, and then the latent variable z will be sampled from a posterior $q_\phi(z|y,e)$ at the encoder, which is usually derived by linear layers [34]. Eventually, the latent variable will also be concatenated with y, and the input embedding will be reconstructed from $p_\theta(e|y,z)$ at the decoder. The reconstructed embedding can be denoted as $\hat{e}$.

In general, the parameters of the LC-VAE can be estimated efficiently with the stochastic gradient variational Bayes (SGVB) algorithm [58]. The evidence-lower bound (ELBO) is a surrogate objective function that can help the estimation. Besides, to better capture the latent pattern of normal sliding

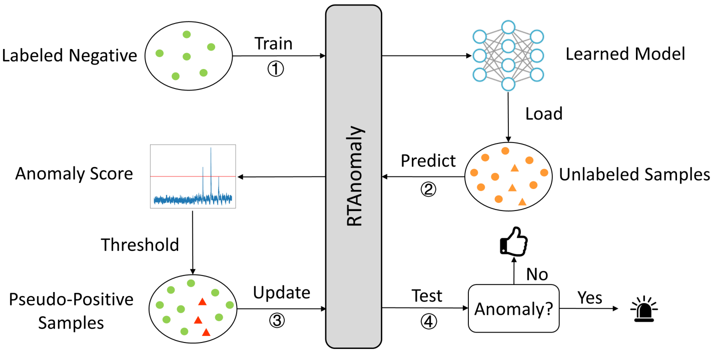


Fig. 4. Positive Unlabeled Learning

windows, we add an additional reconstruction error term. The loss function of the proposed LC-VAE is shown as follows:

$$\begin{aligned}\mathcal{L}_{LCVAE} =& sgn(0.5 - y) * (-KL(q_\phi(z|y,e)\|p_\theta(e|y,z)) \\ &+ \frac{1}{S}\sum_{s=1}^{S}(\log p_\theta(e_s|y) + \lambda \cdot \|e_s - \hat{e_s}\|_2))\end{aligned} \tag{5}$$

Where S is the number of sliding window samples, the first two terms are from evidence-lower bound (ELBO), and the third term is the reconstruction error of the embedding. In this way, we combine the strength of reconstruction and probabilistic estimation together. The coefficient $\lambda > 0$ is to trade off the loss terms. As mentioned above, metrics anomaly detection works by learning the normal patterns of sliding windows of metrics; thus, we should minimize the loss function for a coming normal sliding window. However, when there is an anomalous sample input, we should avoid the model to learn the pattern of anomaly by maximizing the loss function. Thus, we denote the loss function with the Signum function to control the sign. In this way, during the detection phase, the anomalies are easy to differentiate by ISOLATE. The reconstruction error $\|e_s - \hat{e_s}\|_2$ will be used as the anomaly score during the detection phase.

3.5 Correlation Violation Metrics Localization

Once an anomaly has been detected in a service system, further analyses will be enacted to determine the possible causes for such a performance anomaly [40, 61, 92]. This allows application operators to determine which part of the service this performance issue reported is related to. For monitoring metrics, to further understand the mechanism of performance issues, pinpointing a few metrics that are highly correlated with the root cause is crucial [79, 81]. For example, when we observed that the throughput metrics of two devices were highlighted, it seemed to be a performance issue related to the communication between two devices. While our model highlights the CPU utilization of a service, it is likely to occur due to a lack of computing resources in the run-time environment [50].

However, existing methods regarding monitoring metrics have not integrated anomaly detection and metric localization, namely root cause localization together in a unified pipeline [64]. In this case, the knowledge during the anomaly detection phase cannot be shared with the localization. We integrate these two closely related tasks into a unified framework to provide more hints to system operators.

Since there exist correlations between metrics of service in modern online service systems, the learned correlation graph between metrics during the anomaly detection phase is useful for localizing the metrics that reveal the cause of the anomaly [46, 68, 73]. Regarding localizing anomalous metrics,

Table 2. Statistics of Industrial Dataset

Industrial	Dataset A	Dataset B
Services	21	31
Metrics	4~23	3~25
Train Length	366,513	541,043
Test Length	244,356	360,716
Anomaly Ratio	6.71%	5.88%

correlation information can be more indicative than temporal information. It is straightforward to understand that no matter whether the anomaly is a temporal anomaly that happens on some specific metrics or an anomaly due to the contravention of correlation compared with the anomaly-free stage, the correlation between anomalous metrics and others will undergo drastic changes. However, when some metrics collectively spike, they do not violate the correlations and thus are not anomalous metrics. So, we only utilize the correlations in metrics localization. Particularly based on the above assumption, we can calculate the correlation change as follows:

$$\Delta A_i = \sum_{j \neq i} \|A^a_{ij} - A^n_{ij}\|_1 \tag{6}$$

Where ΔA_i is the variation of correlation between normal and abnormal periods for metric i, the A^n_{ij} is computed by averaging a_{ij} on the training period, while the A^a_{ij} is the mean of a_{ij} during the anomaly segment. Eventually, ISOLATE would highlight a few metrics with high ΔA_i and recommend them to engineers to help them get fine-grained information on the performance issue and double-check the devices related to these metrics.

4 EVALUATION

To comprehensively evaluate the effectiveness of our proposed approaches ISOLATE, we use both a public dataset and two real-world monitoring metric datasets from the online services of Huawei Cloud Company. Particularly, we aim to answer the following research questions (RQs):

- RQ1: How effective is ISOLATE compared with performance anomaly detection baselines?
- RQ2: How effective is each component of ISOLATE in performance anomaly identification?
- RQ3: How effective is ISOLATE in localizing the anomalous metrics?
- RQ4: How sensitive is ISOLATE to the parameters?
- RQ5: How efficient is ISOLATE regarding the number of metrics?

4.1 Datasets

We conduct experiments on a publicly available dataset. To confirm the practical significance of ISOLATE, we collect two datasets from large-scale online services of Huawei Cloud. The statistics of our industrial datasets are shown in Table 2

Public Dataset The public dataset for our experiments is SMD (Server Machine Dataset), which is collected from a large Internet company containing a 5-week-long monitoring metrics of 28 machines [65]. The authors divided the SMD into two subsets of equal size: the first half for the training set and the second half for the testing set. Domain experts labeled anomalies in the SMD testing set based on incident reports.

Industrial Dataset To evaluate the effectiveness of ISOLATE in production scenarios, we collect metrics Application CPU Usage, Memory Usage, Interface Throughput, and so on from the online service of the company. We collect metrics with a sampling interval of one minute for more than one

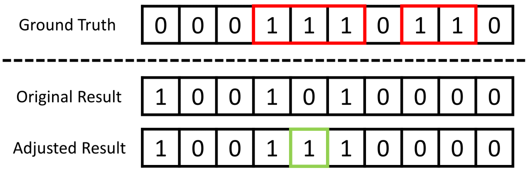


Fig. 5. An Illustration of the Point Adjustment Process in Our Evaluation

week from two regions of the company. The anomalies representing the performance issues of the service are labeled by experienced software engineers with incidents associated with the metrics. The performance issues consist of correlation-violated issues like memory leaks or network congestion listed in Table 1 and non-correlation-violated issues like resource overload. Based on the incident reports, engineers also label the metrics that are correlated to the performance issues. Using these labels, we can also evaluate the accuracy of metrics localization of ISOLATE.

4.2 Experiment Setting

4.2.1 Baselines. The following methods are compared to evaluate the effectiveness of ISOLATE. All the baselines are implemented from the open-sourced codes released by the authors. For both the public dataset SMD and our industrial datasets in Huawei Cloud, we employed a grid search strategy to explore the most suitable parameter configurations. A summary of the baseline methods is shown in Table 3.

Table 3. A Summary of the Baseline Methods

Method Category	Supervision Type	Reference Baseline Methods
Signal Processing-based	Unsupervised	JumpStarter [47]
Machine Learning-based	Unsupervised	LOF [6], IForest [41], OCSVM [60]
LSTM-based	Unsupervised	LSTM [26, 87], LSTM-VAE [53], THOC [62]
Autoencoder-based	Unsupervised	DAGMM [93], OmniAnomaly [65], ACVAE [85] TranAD [66], MTSAD [69], MSCRED [82]
	Semi-supervised	SLA-VAE [24]
Graph Neural Network-based	Unsupervised	MTAD-GAT [88], GDN [16], GTA [9], FuSAGNet [21]

- *OCSVM* [60]. OCSVM is a clustering-based anomaly detection method that learns the boundary for the normal data points and identifies the data outside the border as anomalies. The input in this baseline is the observation of time series at each timestamp and the output is the anomalous timestamps.
- *IForest* [41]. Isolation Forest ensembles a number of isolation trees and recursively partitions the feature space to detect anomalies. The samples with awfully shorter heights are likely to be anomalies. In this baseline, the input is the observation at each timestamp and the output is the anomalous timestamps.
- *LOF* [6]. Local Outlier Factor (LOF) is based on density estimation that calculates the local density deviation of a given sample with respect to its neighbors. The anomalies have a substantially lower density than their neighbors. The input is the observation at each timestamp and the output is the anomalous timestamps.

- *DAGMM* [93]. DAGMM is a model that utilizes an autoencoder to generate a low-dimensional representation and a Gaussian Mixture Model to go through a probabilistic estimation to obtain the anomaly score.
- *LSTM* [26, 87]. LSTM neural network captures the normal behaviors of metrics by forecasting the next values of metrics based on historical observations. Anomalies will be reported if the differences between predicted values and real values exceed a pre-defined threshold
- *LSTM-VAE* [53]. LSTM-VAE detects anomalies by integrating LSTM and VAE. It projects observations at each timestamp into a latent space and then estimates the distribution of it using VAE.
- *OmniAnomaly* [65]. OmniAnomaly is a model that captures the normal patterns by learning robust representations of metrics with stochastic Recurrent Neural Network (RNN) and planar normalizing flow based on the reconstruction error.
- *THOC* [62]. THOC is a model that captures the multi-scale temporal features from dilated recurrent layers by a hierarchical clustering mechanism and detects the anomalies by the multi-layer distances.
- *MTSAD* [69]. MTSAD is a deep unsupervised anomaly detection model that incorporates the strength of active learning, including three feedback strategies, namely denominator penalty, negative penalty, and metric learning.
- *MSCRED* [82]. MSCRED utilizes convolutional LSTM layers to capture the temporal information and embed the inter-metric information through a signature matrix. It detects anomalies with reconstruction errors of the input metric.
- *MTAD-GAT* [88]. MTAD-GAT is a graph attention-based model that captures both feature and temporal correlations. It passes these correlations to a Gated-Recurrent-Unit (GRU) network for reconstruction and forecast.
- *GDN* [16]. GDN learns the graph of relationships between metrics through graph structure learning. It then uses attention-based forecasting and deviation scoring to output anomaly scores.
- *GTA* [9]. GTA is a transformer-based model that employs graph structure learning to learn the relationship among multiple IoT time series, and the Transformer for temporal modeling and the reconstruction error for anomaly detection.
- *TranAD* [66]. TranAD is a transformer-based model that detects anomalies with reconstruction error by incorporating attention mechanisms and adopting adversarial training.
- *SLA-VAE* [24]. SLA-VAE is a semi-supervised VAE-based anomaly detection model for online service systems, which employs active learning to update the model.
- *ACVAE* [85]. ACVAE is a self-adversarial variational autoencoder combined with a contrast learning mechanism that allows the encoder to obtain more training samples.
- *FuSAGNet* [21]. FuSAGNet jointly optimizes reconstruction using a sparse autoencoder and forecasting using a graph neural network. It captures the interdependencies between time series in sensors.
- *JumpStarter* [47]. JumpStarter is a compressed sensing-based method combined with shape-based clustering and an outlier-resistant sampling algorithm. This combination ensures a shorter initialization time.

4.2.2 Evaluation Metrics. The anomaly detection problem is modeled as a binary classification problem, so the widely-used binary classification measurements can be applied to evaluate the performance of models. We employ Precision: $PC = \frac{TP}{TP+FP}$, Recall: $RC = \frac{TP}{TP+FN}$, F1 score: $F1 = 2 \cdot \frac{PC \cdot RC}{PC+RC}$. Specifically, TP is the number of abnormal samples that the model correctly discovered; FP is the number of normal samples that are incorrectly classified as anomalies; FN is the number of anomalous samples that failed to be detected by the model. F1 score is the harmonic mean of

Table 4. Experimental Results of Different Anomaly Detection Methods.

Methods	SMD			Dataset A			Dataset B		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
OCSVM	0.4434	0.7672	0.5619	0.8639	0.5491	0.6446	0.8979	0.5955	0.6547
IForest	0.4231	0.7329	0.5364	0.9375	0.5782	0.6726	0.9443	0.6713	0.7431
LOF	0.5634	0.3986	0.4668	0.9218	0.5744	0.6693	0.9583	0.4302	0.6160
DAGMM	0.5951	0.8782	0.7094	0.7514	0.8108	0.7792	0.8112	0.9073	0.8421
LSTM	0.7855	0.8528	0.8178	0.9177	0.8107	0.8366	0.9166	0.6994	0.7665
LSTM-VAE	0.8698	0.7879	0.8083	0.8753	0.7443	0.7936	0.7969	0.7714	0.7839
OmniAnomaly	0.8368	0.8682	0.8522	0.8769	0.9084	0.8892	0.9437	0.7985	0.8607
THOC	0.7976	0.9095	0.8499	0.9502	0.8022	0.8347	0.9613	0.8400	0.8866
MTSAD	0.8745	0.9395	0.9042	0.8719	0.9171	0.8950	0.9377	0.8765	0.9026
MSCRED	0.7876	0.9374	0.8434	0.8928	0.8451	0.8570	0.9554	0.8522	0.8838
MTAD-GAT	0.8210	0.9215	0.8683	0.8519	0.9019	0.8682	0.9329	0.8769	0.9012
GDN	0.7670	0.9362	0.8312	0.8831	0.9109	0.9018	0.9614	0.8662	0.8994
GTA	0.8768	0.8987	0.8822	0.8671	0.9098	0.8784	0.9431	0.8791	0.8554
TranAD	0.8882	0.9023	0.8953	0.9275	0.8703	0.8867	0.8816	0.8929	0.8874
SLA-VAE	0.8672	0.9371	0.9019	0.9523	0.8605	0.8975	0.9361	0.8859	0.8937
ACVAE	0.8779	0.8384	0.8612	0.8988	0.8405	0.8645	0.9151	0.8432	0.8563
FuSAGNet	0.8319	0.9473	0.8736	0.9011	0.8736	0.8790	0.9038	0.8941	0.8978
JumpStarter	0.8913	0.9174	0.9063	0.8427	0.7959	0.8295	0.8697	0.7869	0.8435
ISOLATE	**0.8998**	**0.9745**	**0.9346**	**0.9871**	**0.9435**	**0.9497**	**0.9759**	**0.9367**	**0.9514**

the precision and recall, which symmetrically represents both precision and recall in one metric. Following [3], we get the anomaly threshold by grid search for all baselines and ISOLATE to evaluate the performance.

In real-world applications, anomalies will last for a while, leading to consecutive anomalies in the monitoring metrics. Therefore, it is acceptable for the model to trigger an alert for any point in a contiguous anomaly segment if the delay is within the acceptable range. Thus, we adopt the evaluation strategy following [57, 65, 66] that marks the whole segment of continuous anomalies as an anomaly. In other words, we consider the model to correctly predict an anomalous segment if at least one timestamp is successfully predicted as an anomaly. An example of the point adjustment strategy is shown in Figure 5. The first anomaly segment is treated as correctly predicted, so the second point of the segment is adjusted to correctly predicted as anomalous.

4.2.3 Implementations. We run all the experiments on a Linux server with Intel Xeon Gold 6140 CPU @ 2.30GHZ and Tesla V100 PCIe GPU. The proposed model is implemented under the PyTorch framework and runs on the GPU. The hidden sizes of the GAT layer, GRU layer, and temporal convolution layers are 256, 128, and 128. The coefficient of loss function λ is 0.5. The dimension of the latent variable in LC-VAE is 10. The threshold of positive learning is 0.9. We train ISOLATE with the Adam optimizer [33] with a learning rate of 0.001, a batch size of 128, and an epoch number of 50. We have released the artifacts and data for future research purposes on https://github.com/WenweiGu/ISOLATE.

4.3 Experimental Results

*4.3.1 **RQ1** The effectiveness of ISOLATE.* To answer this research question, we compare the performance of ISOLATE with other state-of-the-art baselines on a public dataset and two industrial datasets. First, We train ISOLATE on a small portion of negative samples. Then, ISOLATE will assign pseudo labels to the remaining training data according to the anomaly score and threshold

Table 5. Experimental Results of the Ablation Study

Methods	SMD			Dataset A			Dataset B		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ISOLATE *w/o* $\mathcal{RT}$	0.8512	0.9079	0.8723	0.9791	0.8285	0.8738	0.9465	0.8879	0.9035
ISOLATE *w/o* $\mathcal{PU}$	0.8730	0.9407	0.9062	**0.9912**	0.8405	0.8869	0.9396	0.9189	0.9231
ISOLATE	**0.8998**	**0.9745**	**0.9346**	0.9871	**0.9235**	**0.9497**	**0.9759**	**0.9367**	**0.9514**

β. During the test phase, the anomaly score for each timestamp will be computed. The anomaly threshold will be searched following [3] to produce the prediction result.

The results are shown in Table 4, where the best F1 scores are marked with boldface. We can see the average F1 score of ISOLATE outperforms all baseline methods in three datasets. The experimental results are shown in Table 2. In Dataset B, the improvement achieved by ISOLATE is more significant as the metrics correlations between metrics in Dataset B are more complicated and the ratio of performance anomalies caused by correlation violation is higher. Generally speaking, ISOLATE 's good performance can be attributed to two reasons: Firstly, the utilization of relational-temporal embedding, as the anomalies can be caused by the violation of correlation, which can hardly be detected by finding the spikes on a single metric, it can also help facilitate localization of the anomalous metrics. Thus, there is a significant improvement in the recall of ISOLATE compared to other baselines. Secondly, ISOLATE effectively learns potential anomalous samples from the training data, thereby mitigating the risk of overfitting anomalous patterns during the training process. Consequently, ISOLATE achieves remarkable precision, ranking among the best when compared to other baseline methods.

Typically, the baseline methods have higher precision than recall because there are some anomalies that are not very apparent. We can observe that machine learning-based methods, namely OCSVM, IForest, and LOF have relatively low performance compared with other baseline models since these methods learn the metrics pattern at each timestamp independently without considering the temporal dependency. While LSTM-based methods and autoencoder-based methods, including LSTM, DAGMM, and LSTM-VAE, perform notably better than OCSVM, IForest, and LOF and achieve 0.7094~0.8421 F1 score because these models take the historical observation window of the data, that helps to retain valuable historical temporal pattern. Among the baselines, we can find OmniAnomaly, THOC, MTSAD, MSCRED, TranAD, ACVAE, and SLA-VAE can achieve relatively better performance (especially MTSAD can achieve an F1 score like 0.9042) because these methods introduce some mechanisms to extract temporal information of metrics and ensure robust anomaly detection. It should be noted that SLA-VAE is a semi-supervised VAE-based method and it employs active learning to update the model. However, our proposed method outperforms SLA-VAE, suggesting that the improved performance of our model is not solely due to the semi-supervised mechanism, *i.e.*, PU learning, but also the design of combining correlational violation detection with temporal information. Specifically, OmniAnomaly models the metrics through stochastic variables and uses reconstruction probabilities to determine anomalies. As for MSCRED, the temporal information is also captured through convolutional LSTM. Meanwhile, in MTSAD, active learning has been incorporated into a variational autoencoder to ensure capability against noise. SLA-VAE is also a variational autoencoder-based architecture that employs both active learning and semi-supervised learning. Adversarial training is incorporated in the VAE-based model ACVAE and the transformer-based model TranAD, which assures their robustness. In THOC, the complex nonlinear temporal dynamics of the system's normal behavior are captured. Though extracting temporal information well, the limitation of these approaches lies in not taking the correlation features into consideration, which is essential to successfully detecting anomalies from multivariate metrics.

Table 6. Performance on Metrics Localization

Methods	Dataset A		Dataset B	
	Hit@1	Hit@3	Hit@1	Hit@3
DAGMM	0.5714	0.6667	0.5806	0.7419
LSTM-VAE	0.6190	0.7142	0.6451	0.7741
ISOLATE-Cor	0.7419	0.8387	0.7142	0.8571
ISOLATE	**0.8095**	**0.9048**	**0.8387**	**0.9354**

Graph Neural Network (GNN)-based approaches offer some mitigation to this issue through jointly extracting the relational and temporal information of raw metrics, *e.g.*, MTAD-GAT, GDN, GTA, and FuSAGNet. However, without explicitly capturing the correlation violation, these approaches still achieve suboptimal performance compared with ISOLATE.

4.3.2 ***RQ2*** *The effectiveness of components in ISOLATE.* To answer this research question, we conducted an extensive ablation study on ISOLATE. Particularly, we derive two baseline models based on removing the relational-temporal embedding and positive unlabeled learning parts of ISOLATE to investigate the contribution of these two components.

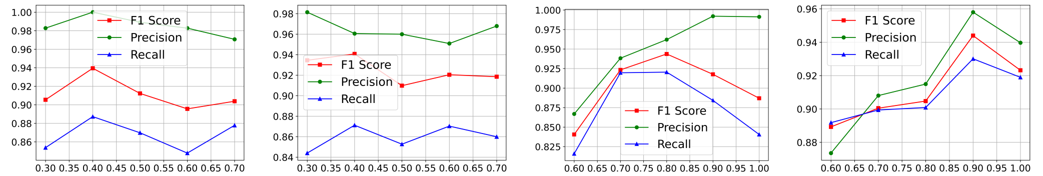
- *ISOLATE w/o $\mathcal{RT}$* This baseline is a variant of ISOLATE that removes relational-temporal embedding that captures both information of metrics. Instead, only an LSTM layer is utilized in this baseline to embed the temporal information of the raw metrics, which will be further fed into the CVAE.
- *ISOLATE w/o $\mathcal{PU}$* This baseline removes the positive unlabeled learning that finds anomalous samples from a large number of normal samples. All the training data are considered normal in this baseline.

Table 5 shows the experimental results of ISOLATE and its variants. Overall, relational-temporal embedding and positive unlabeled learning help to improve the effectiveness of ISOLATE as it performs the best, while the degree of contribution of relational temporal embedding is larger. We attribute this to the good capability of relational temporal embedding in extracting both the temporal information of each metric and correlational information between metrics. When an anomaly happens due to a breach of relationship during the anomaly-free period, it can be easily identified by ISOLATE, while other methods have difficulty identifying it as they are more effective on temporal outliers. We observe that even without PU learning, our model is not worse than all baselines, which further demonstrates the effectiveness of explicitly extracting correlation violation.

The variant without relational-temporal embedding is similar to LSTM-VAE, which employs LSTM to extract the sequential information and VAE to differentiate the anomaly from normal. However, due to the design of positive unlabeled learning and conditional VAE, the variant can identify the noise of training data and label them as positive. Thus, the performances on three datasets of ISOLATE without relational-temporal embedding in terms of F1 score are improved compared to LSTM-VAE, respectively. We believe that in some scenarios with a higher ratio of noise, positive unlabeled learning would play a greater role in improving performance.

4.3.3 ***RQ3*** *The effectiveness of ISOLATE in localizing the anomalous metrics.* To further demonstrate the capability of ISOLATE, experiments on localizing the metrics are conducted. Specifically, we localize the metrics by following four methods:

- *DAGMM.* This baseline uses the anomaly score of each metric output by DAGMM as the degree of the metric being anomalous. The metrics with the highest scores will be highlighted as anomalous metrics.



(a) Sensitivity of α on Dataset A (b) Sensitivity of α on Dataset B (c) Sensitivity of β on Dataset A (d) Sensitivity of β on Dataset B

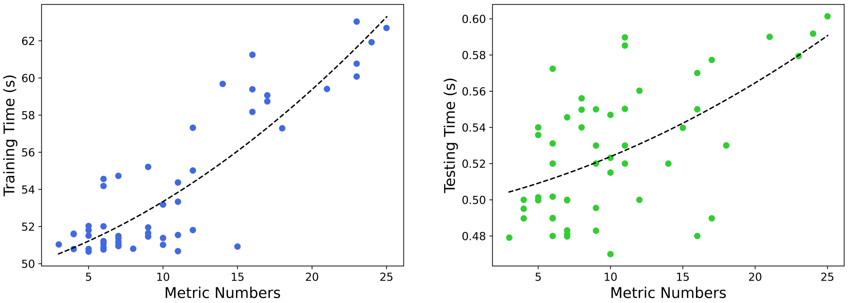
Fig. 6. The Sensitivity Analysis of Threshold α and β

- *LSTM-VAE.* This baseline uses the anomaly score output by LSTM-VAE as the degree of being anomalous for the metrics.
- *ISOLATE-Cor.* This baseline sums up the correlation of a specific metric between other metrics as the degree of being the root cause. Metrics with a high correlation with other metrics would be reported as anomalous metrics.
- *ISOLATE.* Different from ISOLATE-Cor, it uses the discrepancy between the anomaly-free period and anomaly period since the correlation between metrics may undergo drastic changes compared to the normal period when an anomaly happens.

As mentioned in Section 2, a metric that shows very abnormal behaviors compared to the normal period is not necessarily the most anomalous metric because it can have a small influence on other metrics and will self-heal. Using the correlation score itself can also cause inaccurate results because some metrics show consistently high correlations with others, no matter whether it is during an anomaly period or not. Indeed, the correlation difference between normal and abnormal time is a stronger indicator of anomalous metrics because when an anomaly happens, the correlation would be obeyed and cause a huge correlation change.

Table 6 presents the comparison of four methods, and we can observe that metrics localization using ISOLATE outperforms the other three baselines with a Hit@1 of 80.95%, 83.87%, and Hit@3 of 90.48%, 93.54% on two industrial datasets. Thus, our method can provide accurate hints for engineers on which part of the service system they can remedy to ensure the system's reliability. Compared to using the anomaly score of DAGMM and LSTM, using the correlation score to localize metrics (ISOLATE-Cor) is more effective as the anomaly scores of these two methods are computed on a single metric and are not aware of other metrics, while correlation considers the global information of metrics. Usually, the metrics that have a higher correlation with other metrics seem to play a greater influence on the service system and are more likely to be anomalous metrics when an anomaly happens.

4.3.4 ***RQ4*** *The sensitivity of ISOLATE to the parameters.* The threshold α is the parameter that determines the sparsity of the relational learning, while the threshold β, *i.e.*, the parameter that determines the label of training data during positive unlabeled learning, may affect the performance by affecting the label distribution of training samples. We hereon evaluate the sensitivity of ISOLATE to these two hyper-parameters on two industrial datasets. We change the value of α and β while keeping all other parameters unchanged in our experiments to guarantee fairness. Specifically, we choose the value of α in an appropriate range from 0.3 to 0.7 at a step of 0.1. The value of β is selected, ranging from 0.6 to 1 at a step of 0.1. When the value of β is 1, it is equivalent to the variant without positive unlabeled learning since all samples will be seen as normal. When the value of β is lower than 0.5, a large portion of samples will be labeled as anomalous and introduce severe noise to training data, which is not the case in the real scenario.



(a) Training Time versus Metric Numbers (b) Testing Time versus Metric Numbers

Fig. 7. The Efficiency Analysis of ISOLATE

Figure 6 presents the experimental results of RQ4. For the threshold α, the performance is relatively stable under different settings. It can be attributed to the distribution of learned graph attention scores, which are either close to 0 or 1. The polarization of graph attention scores means that the threshold chosen within the aforementioned range tends not to impact the performance in a significant manner, thus stable across different values of α. This makes ISOLATE easy to deploy in practice. For parameter β, a good threshold indeed helps improve the accuracy of our model. In dataset A, the best threshold is between 0.7 and 0.8, while in dataset B, the best threshold is between 0.8 and 0.9. This is because the anomaly ratio of Dataset B is slightly lower than Dataset A, so a lower amount of unlabeled anomalous samples exist in the training part. Compared to without positive unlabeled learning, a properly selected threshold would improve recall because some abnormal behaviors have been labeled as positive, and these abnormal patterns would be reported in the testing set, thus reducing false negatives.

*4.3.5 **RQ5** The efficiency of ISOLATE.* In this section, we evaluate the efficiency of ISOLATE regarding the number of metrics. We perform our method on the industrial dataset A and B and record the training and testing costs. The training time is defined as the total time of feeding the preprocessed data to the model, and the testing time is the total time used to predict whether there are performance anomalies on all the sliding windows in the test set. All the data samples are trained with the same batch size for 20 epochs. The correlation between training/testing time and metrics number is shown in Figure 7, where we can observe that both of them are near quadratic correlations. In our scenario, the metrics collected for node-level and service-level performance anomaly detection are typically smaller than 100. According to the polynomial approximation, the training and testing time for data that contains 100 monitoring metrics is 192.9 seconds and 3.89 seconds, respectively. Typically, model retraining will not be triggered more frequently than once per day due to the computation cost. Thus, the training time of 192.9 seconds is affordable for industrial deployment. Furthermore, since the monitoring metrics are typically collected at an interval of 1 minute, the testing time of 3.89 seconds in the online detection phase is enough for real-time performance issues identification. It should be noted that all the experiments are run on a Linux server with only one Tesla V100 PCIe GPU, however, the efficiency is even further enhanced when utilizing the significant computational power that can be harnessed from thousands of GPUs available in cloud service systems.

4.4 Case Study

To further demonstrate the effectiveness of ISOLATE, we conduct a case study concerning two industrial cases on identifying performance anomalies and localizing the anomaly on metrics that

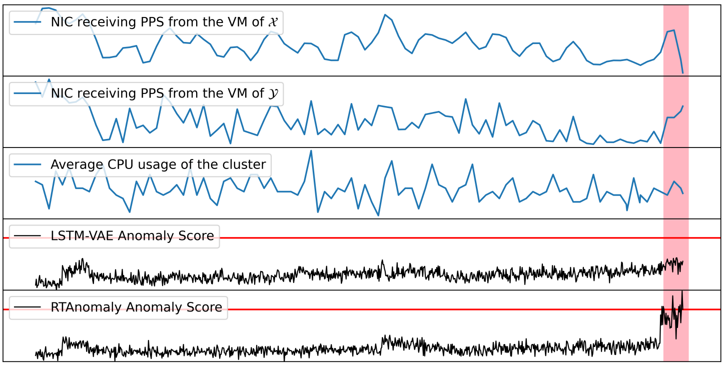


Fig. 8. A case that ISOLATE finds false negative

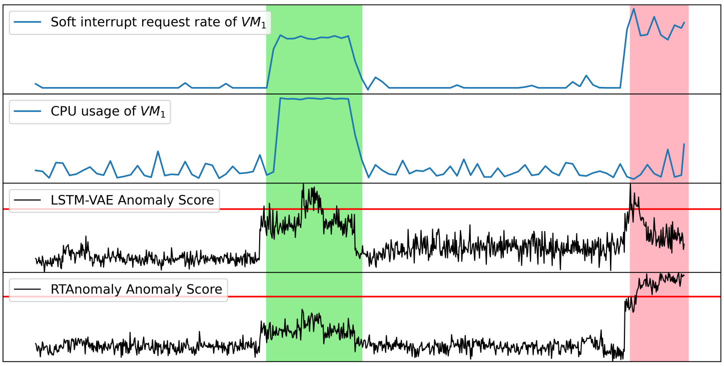


Fig. 9. A Case that ISOLATE avoids false positive

violate their intrinsic correlation. The first case is shown in Figure 8, which is the same example in Section 2.2. The first three rows show the monitoring metrics of the NIC receiving PPS from a virtual machine running microservice $\mathcal{X}$, the NIC receiving PPS from a virtual machine running microservice $\mathcal{Y}$, and the average CPU usage of all virtual machines of the ELB service. The last two rows show the anomaly score of the baseline method LSTM-VAE and our proposed ISOLATE for comparison. This anomaly can hardly be discovered when observing each metric individually, as the correlation violation is critical to identifying this performance issue. However, our ISOLATE can find the false negative that is neglected by baseline models. The first and second metrics can also be highlighted to provide engineers with further insight into the performance issue.

Another example is shown in Figure 9. The first metric is the soft interrupt request rate of a virtual machine VM_1 running on an Elastic Cloud Server (ECS). The second metric is the CPU usage of VM_1. We can observe that the green segment seems like a performance issue, as the soft interrupt request rate of both VM_1 and the CPU usage all encounter spikes. However, this is usually caused by increased user requests and should not be regarded as a performance issue. In contrast, since the correlations between these metrics are not violated, an anomaly alert should not be triggered. The red segment represents a soft interrupt issue because the CPU usage of VM_1 is not synchronized with the increase of the soft interrupt request rate of VM_1. Compared with the baseline method LSTM-VAE, ISOLATE can produce a significantly higher anomaly score in the true anomaly labeled with the red

Table 7. A Summary of the Node Types in the Huawei Cloud

Node Type	Description
Management Nodes	Management nodes are used to deploy FusionSphere OpenStack, an enterprise-level platform to enhance computing, storage, network management, installation and maintenance, security, and reliability while supporting multiple infrastructure virtualization technologies at the resource pool layer. Management nodes use UVP as the host OS. The Computing cloud services, storage cloud services, network cloud services, common components, and management domain components are deployed on VMs.
Network Nodes	The network node uses the UVP as the host OS. The virtual Router, L3NAT, L3 service, and VPN components are deployed on VMs.
Compute Nodes	Compute nodes can be divided into two subtypes. The first is the KVM compute node for general-purpose Elastic Computing Services (ECS) in tenant VMs. The second is the KVM compute node for GPU-accelerated ECSs which provision GPUs for deep learning jobs in tenant VMs. These two types of KVM compute nodes use the UVP as the host OS, and FusionSphere OpenStack (role compute) is also deployed.
Storage Nodes	Distributed storage nodes in Huawei typically support Elastic Volume Service (EVS). After the deployment of Huawei Distributed Block Storage, this node is utilized by the EVS service to provision EVS instances in tenant EVS disks.

area, thus being more effective. In this case, though both ISOLATE and baseline methods like LSTM-VAE can detect the true anomaly, our ISOLATE can avoid sending a false alarm to engineers. This is helpful for the engineers to mend the service system because it prevents unnecessary interventions.

5 DISCUSSION

In this section, we share the success story of our deployment of ISOLATE in the industrial environment of the service system of Huawei Cloud and discuss the threats and limitations of our approach.

5.1 Industrial Experience

In this section, we share our experience in applying ISOLATE to the real-world cloud system of Huawei Cloud, a full-stack cloud system that consists of an infrastructure layer, a platform layer, and an application layer, aiming to demonstrate the practical usefulness of ISOLATE. Huawei Cloud serves hundreds of millions of cloud service tenants, offering them low-latency and high-performance services that span computing, storage, networking, and database solutions. Among them, ELB is a crucial service that is tasked with the automatic distribution of incoming network traffic across a multitude of targets, including Elastic Cloud Server (ECS) instances, containers, and IP addresses across different Availability Zones (AZs). Relational Database Service (RDS) is another reliable and scalable managed DB service that frees up developers from handling time-consuming database administration tasks. In addition, the API Gateway Service (APIG), is an API management service that serves as a single point of entry into cloud systems, sitting between the user and a collection of backend services. It receives requests from an application user, routes the request to the appropriate services, gathers the appropriate data, and combines the results for the user in a single package. The Object Storage Service (OBS) provides stable, secure, efficient, and easy-to-use object storage. It enables storage and retrieval of any amount of data at any time, facilitating cloud storage for applications, big data analysis, and backup and archiving scenarios. The reliability of these services is among the most crucial concerns for Huawei Cloud and its tenants.

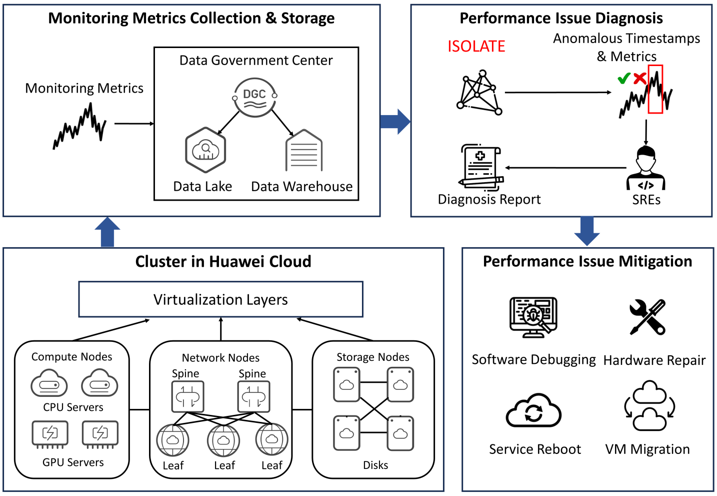


Fig. 10. The Pipeline of Deploying ISOLATE in Huawei Cloud System

ISOLATE has been successfully incorporated into the performance issue detection system of large-scale online service systems in Huawei Cloud since *November 2022*, specifically, the overall pipeline of deployment is shown in Figure 10. The clusters in Huawei Cloud represent a collection of nodes that work together to provide various services, ensuring high availability, reliability, and scalability. Particularly, there are four types of nodes in Huawei Cloud: Management node, network node, compute node, and storage node, where the details of them are elaborated in Table 7. For clusters in cloud service systems, it has been a common practice for monitoring metrics used to profile the runtime status [29, 57, 77]. Thus, software reliability engineers usually collect tens of monitoring metrics (like CPU usage, network traffic, disk I/O, request rates, and memory usage) in nodes of the cluster through monitoring tools like Grafana, Prometheus, etc [1]. Then, these monitoring metrics are stored in the Data Lake of Huawei Cloud, a highly scalable and flexible storage system that consists of the Data Lake Storage, the Data Warehouse, and the Data Lake Governance Center (DGC). Data Lake Storage is the actual storage space where all the data, including the monitoring metrics, are stored while the Data Warehouse is an enterprise system used for reporting and data analysis. On top of these two components, the DGC is responsible for managing the data stored in the data lake that oversees the lifecycle of the data, from ingestion and storage to usage and deletion. With Huawei Cloud's data lake, real-time data analysis is enabled, *i.e.*, as soon as monitoring metrics are collected and stored in the data lake, they can be immediately accessed and analyzed by the performance issue diagnosis system empowered with ISOLATE. The results of ISOLATE provide not only the timing of a performance issue but also the metrics that are most related to this issue. Then, alerts will be triggered immediately and sent to the SREs for further investigation. The SREs first inspect the alert and the associated metrics to understand the mechanism of the problem and find the root cause. Once the root cause has been identified, the SREs prepare a diagnosis report, including a detailed description of the performance issue, the associated metrics, the identified root cause, and potential mitigation strategies. For instance, if the performance issue is a memory leak, engineers may need to perform software debugging involving tracing the program's execution and implementing appropriate code fixes to prevent the leak from happening again. As another example, if the problem is network congestion, the mitigation strategies may include hardware repair like replacement of the NICs.

Due to frequent service updates and changes in user behavior, monitoring metric patterns may also evolve, a phenomenon known as concept drift. This drift can gradually weaken the effectiveness

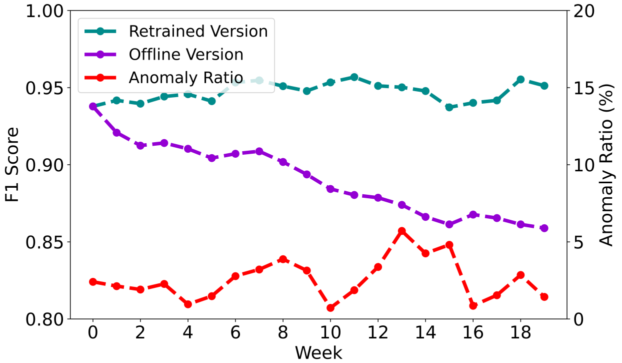


Fig. 11. The Performance of deploying ISOLATE in Huawei Cloud System

of ISOLATE over time. To alleviate this issue, ISOLATE is retrained on a weekly basis with newly collected data. Figure 11 illustrates the performance issue identification accuracy of both the retrained and offline versions over 20 weeks. A noticeable downward trend in the accuracy of the offline version can be observed, attributed to pattern drift. Conversely, the version that is periodically retrained displays relatively stable performance. This adaptability to new patterns in online deployment scenarios underscores the robustness and effectiveness of ISOLATE within dynamically evolving cloud systems. More specifically, take one of the services $\mathcal{R}$ for example, the time proportion of undergoing performance issues during this period is 3.4%, and the online version of ISOLATE achieves a precision of 0.904, recall of 0.951, and F1 of 0.926. Another interesting observation is that though the anomaly ratio analyzed by engineers ranges from 0.7% to 5.6% during the 20-week period, the performance remains relatively stable in the retrained version of ISOLATE, demonstrating that the performance of ISOLATE is not sensitive to the anomaly ratio. It should be noted that the weekly retraining enhances the adaptiveness of our method, with an additional cost of human labeling of the potential performance anomaly data. However, it has been a common practice in cloud systems that on-call SREs manually verify some reported suspicious performance anomaly [10, 43, 70]. Specifically, according to the interview with SREs, they typically spend approximately one hour per week checking these data and they consider it to be affordable.

The ability to timely alert operation engineers after a performance issue happens is also of great significance. The delay time of ISOLATE in Huawei Cloud is the time delay between the detection of performance issues and the happening time of the issues confirmed by customer tickets [42]. The delay time of ISOLATE is shown in Figure 12, where we can observe that most of the performance issues can be captured 5 to 20 minutes after the happening and SREs can take mitigation strategies to avoid continuously compromising the user experience. Consequently, the potential negative impacts on the QoS can be significantly mitigated, enhancing system reliability and ensuring high availability.

5.2 Limitations of ISOLATE

Our approach ISOLATE employs graph attention to extract the correlation between metrics and the PU Learning strategy to identify positive samples in unlabeled training data. Though satisfactory performance in detecting performance anomalies is achieved in our experiments, we have identified several limitations of our approach and the challenges that arise in real-world applications.

Firstly, our approach is not adaptive to the evolving metric patterns and correlation in cloud systems due to rapid software and application upgrades [11]. Thus, our model can result in false positives when new patterns or correlations are not included in the training dataset. Fortunately,

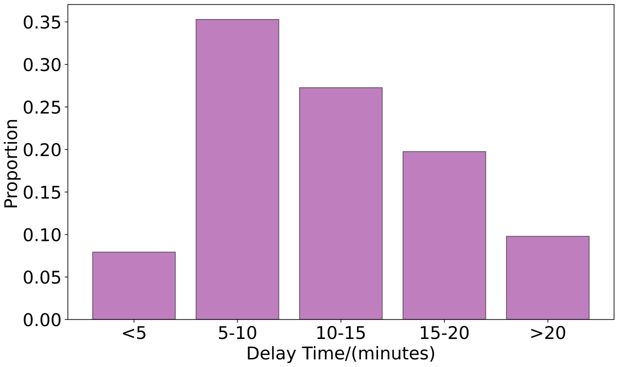


Fig. 12. The Delay Time of ISOLATE in Identifying Performance Issues

we found that regular model retraining, *e.g.*, retraining every week, works well to learn the new behaviors of the cloud system and thus overcome this limitation [22].

Secondly, the complexity of large-scale cloud systems results in a vast number of monitoring metrics, which escalates the computational costs of ISOLATE. Fortunately, our approach is deployed at the node level or service level, where the number of metrics is typically fewer than 100. This reduces computational demands and enhances efficiency. It's important to note that performance issues detected at the node or service level can serve as indicators of system-level performance anomalies, offering valuable insights and early warnings about potential system failures, which is useful for SREs.

Thirdly, our approach lacks interpretability in traditional methods such as decision trees, which can be a limitation to some content. However, it's worth noting that our method is not entirely without interpretability. The correlation violations between metrics can provide engineers with more granular information about anomalous metrics, offering some level of insight into the root cause localization of performance issues, and making it partially interpretable.

5.3 Threats to Validity

Internal threats. The correctness of the implementation of baselines constitutes one of the internal threats to our study's validity. For the baselines, we utilized the open-sourced code released by the authors of the papers or packages on GitHub like [91]. As for our proposed approach, the source code has been reviewed meticulously by the authors, as well as several experienced software engineers, to minimize the risk of errors and increase the overall confidence in our results. For parameter selection, we conducted extensive experiments with different parameters to find the most suitable configurations for both baselines and our proposed method. We chose the parameters based on the best results obtained in these experiments. To make our results reproducible, we have also made our code and partial data available.

External threats. The external threats to the validity of our study mainly lie in the generalizability of our experimental results. We conduct experiments on the large-scale online systems of two regions within a prominent cloud service company. In addition to this, our approach is also evaluated on a publicly available dataset containing monitoring metrics from an Internet company, further expanding the scope of our evaluation. While the diversity of the experimental settings provides some confidence in the generality of our findings, it is essential to acknowledge that results might vary when applied to different cloud service providers, industries, or specific use cases. Nevertheless, we believe that our experimental results, obtained from these multiple sources, can demonstrate the generality and effectiveness of our proposed approach, ISOLATE.

6 RELATED WORK

Detecting performance issues on monitoring metrics for online service systems has been a hot topic. Monitoring metrics used to reflect the run-time status of the whole system are usually denoted as multivariate time series. The main challenge of multivariate metrics anomaly detection is twofold: First, effective modeling of complex relational and temporal dependency [74]. Second, noise data will be introduced inevitably during the manual labeling process. It is hard to get rid of these noises. Related studies can be categorized into machine learning or signal processing-based and deep learning-based approaches.

Machine Learning/Signal Processing Methods. OCSVM [60] is a clustering-based method that learns the boundary for the normal data without anomalous samples and identifies the data outside the border as anomalies. Isolation Forest (iForest) [41] applies multiple isolation trees and ensembles them based on the assumption that anomalies should be rare and isolated from normal observations with very short heights. Local Outlier Factor (LOF) is a density estimation-based anomaly detection approach that calculates the local density. The samples with an extremely lower density compared to their neighbors would be recognized as anomalies. JumpStarter [47] is a signal processing-based method that adopts the compressed sensing technique to reconstruct the input data. It adopts a shape-based clustering strategy to reduce the volume of data and utilizes an outlier-resistant sampling to avoid sampling anomalous values.

Deep Learning Methods. With the surge of deep learning [83, 84], there has been a variety of studies in applying deep learning to conduct anomaly detection on multivariate metrics data. A deep autoencoder with a Gaussian mixture model (DAGMM) [93] is utilized to detect anomalous data points. MSCRED [82] proposes a multi-scale convolutional recurrent encoder-decoder that detects anomalies. The reconstruction error of the input time series is utilized to diagnose anomalies. To detect performance degradation anomalies in software systems, Long Short-Term Memory (LSTM) has been deployed to guarantee high performance in [87]. LSTM-VAE [53] combines the LSTM networks and the VAE to reconstruct the distribution of sliding windows from multivariate metrics regardless of the temporal dependencies in time series [11]. Similarly, LSTM-NDT [26] leverages Long Short-Term Memory (LSTM) networks with non-parametric dynamic thresholds to pursue the reliability of the systems. OmniAnomaly [65] extends the LSTM-VAE with a normalizing flow and utilizes the reconstruction error for detection. However, the capability of this approach is degraded when there is severe noise in the training metric. THOC [62] is a model that fuses the temporal features at multi-scale from intermediate layers by hierarchical clustering and detects the anomalies by the multi-layer distance loss. MTSAD [69] is another model combining active learning with existing VAE-based anomaly detection, which detects the difference between normal and abnormal samples in reconstruction error and latent space. TranAD [66] is a transformer-based anomaly detection model that adopts attention-based sequence encoders to swiftly perform inference with the knowledge of the temporal trends in the data. It further leverages self-conditioning and adversarial training to gain training stability. ACVAE [85] is a variational autoencoder-based method that combines adversarial mechanisms and contrast learning. The former constrains the decoder and gains some discriminatory power, while the latter allows the encoder to obtain more training samples. SLA-VAE [24] is a semi-supervised VAE-based model that employs convolutions to capture inter-metric dependence and utilizes active learning to update the VAE model using a small number of uncertain samples.

Recently, among the deep learning-based anomaly detection model, Graph Neural Networks (GNNs) based methods have shown promising potential to effectively capture temporal and spatial dependencies among metric pairs of multivariate metrics [32]. MTAD-GAT [88] is the first work that utilizes the graph-attention network to model the relationships between sensors in addition to jointly optimized reconstruction and forecasting-based models. Another representative model is the Graph deviation network (GDN) [16], which learns the pairwise relationship through cosine similarity and models the time series as a graph through an adjacent matrix. It predicts future values and computes the forecast error as an anomaly score. GTA [9] is a Transformer-based framework for anomaly detection that automatically learns sensor dependencies. The authors propose an Influence Propagation (IP) graph convolution and multi-branch attention technique that improves the training efficiency. FuSAGNet [21] utilizes a sparse autoencoder to extract latent feature representations. It then predicts future time series from the sparse representations and graph structures learned from graph neural networks. TopoMAD [23] is a topological-aware model that integrates graph neural networks, LSTM and VAE. The topological information is extracted through the pod-based division in a Kubernetes microservice system or pods that share similar behaviors. It should be noted that the topology in Huawei Cloud is highly dynamic due to frequent deployment changes and software updates, which makes it challenging to determine. Thus, TopoMAD does not apply to our scenario.

Furthermore, many efforts have been devoted to multi-source data-based anomaly detection in microservice systems [37]. For example, SCWarn [90] is proposed to identify bad software changes in online service systems via multimodal anomaly detection from metrics and logs. It serializes the metrics and logs separately and extracts the temporal dependency through the LSTM model. AnoFusion [86] is another unsupervised failure detection approach for microservice systems. It employs a Graph Transformer Network (GTN) to capture correlations within the heterogeneous multimodal data. It then seamlessly integrates a Graph Attention Network (GAT) with a Gated Recurrent Unit (GRU) to model the temporal information of dynamically changing multimodal data. Though three types of monitoring data (metrics, logs, and traces) can be utilized to ensure the reliability of microservice systems [55], the real-time collection of these three types of data can be challenging.

Though some GNN-based methods like [16, 88] capture both the temporal dependencies and relational dependencies, performance issues can be indicated by the violation of correlation in our scenario, which is not taken into account explicitly in existing approaches. Besides, the performance of these models will degrade due to the existence of unlabeled noise in training data, where noise is prevalent in our scenario.

7 CONCLUSION

In this work, we propose ISOLATE, a novel framework to mine correlations among metrics, detect performance issues, and localize the correlation-violation metrics. Specifically, ISOLATE leverages a graph neural network with graph attention to capture the complex correlations between a variety of metrics and a label-conditional VAE model to distinguish normal and abnormal patterns. We also propose to utilize the positive unlabeled learning strategy to overcome the impacts of noisy data. Extensive experiments on one public dataset and two industrial datasets show that ISOLATE achieves 0.945 F1-Score on anomaly detection and 0.920 Hit@3 in terms of localizing correlation-violation metrics, outperforming all the baselines. Furthermore, our framework has been successfully incorporated into Huawei Cloud's performance issue monitoring and detection system. Both the codes and data are released to facilitate future research.

ACKNOWLEDGMENTS

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund) and Fundamental Research Funds for the Central Universities, Sun Yat-sen University (No. 76250-31610005).

References

[1] Shubham Agarwal, Sarthak Chakraborty, Shaddy Garg, Sumit Bisht, Chahat Jain, Ashritha Gonuguntla, and Shiv Saini. 2023. Outage-Watch: Early Prediction of Outages using Extreme Event Regularizer. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 682–694.

[2] Yasaman Amannejad, Diwakar Krishnamurthy, and Behrouz Far. 2015. Detecting performance interference in cloud-based web services. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 423–431.

[3] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. 2020. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 3395–3404.

[4] Malvinder Singh Bali and Shivani Khurana. 2013. Effect of latency on network and end user domains in cloud computing. In *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*. IEEE, 777–782.

[5] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. 2019. Anomaly detection using autoencoders in high performance computing systems. In *Proceedings of the AAAI Conference on artificial intelligence*, Vol. 33. 9428–9433.

[6] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 93–104.

[7] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491* (2021).

[8] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. 2018. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287* (2018).

[9] Zekai Chen, Dingshuo Chen, Xiao Zhang, Zixuan Yuan, and Xiuzhen Cheng. 2021. Learning graph structures with transformer for multivariate time-series anomaly detection in IoT. *IEEE Internet of Things Journal* 9, 12 (2021), 9179–9189.

[10] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497.

[11] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xiao Ling, Yongqiang Yang, and Michael R Lyu. 2022. Adaptive performance anomaly detection for online service systems via pattern sketching. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. 61–72.

[12] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xuemin Wen, Xiao Ling, Yongqiang Yang, and Michael R Lyu. 2021. Graph-based Incident Aggregation for Large-Scale Online Service Systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 430–442.

[13] Mohan Baruwal Chhetri, Quoc Bao Vo, and Ryszard Kowalczyk. 2016. CL-SLAM: Cross-layer SLA monitoring framework for cloud service-based applications. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*. 30–36.

[14] Zheng Dang, Shuibing He, Peiyi Hong, Zhenxin Li, Xuechen Zhang, Xian-He Sun, and Gang Chen. 2022. Nvalloc: rethinking heap metadata management in persistent memory allocators. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 115–127.

[15] Ulan Degenbaev, Jochen Eisinger, Kentaro Hara, Marcel Hlopko, Michael Lippautz, and Hannes Payer. 2018. Cross-component garbage collection. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–24.

[16] Ailin Deng and Bryan Hooi. 2021. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4027–4035.

[17] Rui Fu, Zuo Zhang, and Li Li. 2016. Using LSTM and GRU neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE, 324–328.

[18] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. 2017. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications* 88 (2017), 50–71.

[19] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC)*. 126–141.

[20] 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 (ESEC/FSE)*. 1387–1397.

[21] Siho Han and Simon S Woo. 2022. Learning sparse latent graph representations for anomaly detection in multivariate time series. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2977–2986.

[22] Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. 2023. Murphy: Performance Diagnosis of Distributed Cloud Applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 438–451.

[23] Zilong He, Pengfei Chen, Xiaoyun Li, Yongfeng Wang, Guangba Yu, Cailin Chen, Xinrui Li, and Zibin Zheng. 2020. A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems. *IEEE Transactions on Neural Networks and Learning Systems* 34, 4 (2020), 1705–1719.

[24] Tao Huang, Pengfei Chen, and Ruipeng Li. 2022. A semi-supervised vae-based active anomaly detection framework in multivariate time series for online systems. In *Proceedings of the ACM Web Conference 2022*. 1797–1806.

[25] Tao Huang, Pengfei Chen, Jingrun Zhang, Ruipeng Li, and Rui Wang. 2022. A Transferable Time Series Forecasting Service Using Deep Transformer Model for Online Systems. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1–12.

[26] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*. 387–395.

[27] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. 2015. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 1–35.

[28] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning (ICML)*. PMLR, 448–456.

[29] Mohammad S Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. 2021. Anomaly detection in a large-scale cloud platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 150–159.

[30] Miao Jiang, Mohammad A Munawar, Thomas Reidemeister, and Paul AS Ward. 2009. System monitoring with metric-correlation models: problems and solutions. In *Proceedings of the 6th international conference on Autonomic computing*. 13–22.

[31] Hai Jin, Zhiwei Li, Haikun Liu, Xiaofei Liao, and Yu Zhang. 2019. Hotspot-aware hybrid memory management for in-memory key-value stores. *IEEE Transactions on Parallel and Distributed Systems* 31, 4 (2019), 779–792.

[32] Ming Jin, Huan Yee Koh, Qingsong Wen, Daniele Zambon, Cesare Alippi, Geoffrey I Webb, Irwin King, and Shirui Pan. 2023. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection. *arXiv preprint arXiv:2307.03759* (2023).

[33] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[34] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[35] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[36] Ryuichi Kiryo, Gang Niu, Marthinus C Du Plessis, and Masashi Sugiyama. 2017. Positive-unlabeled learning with non-negative risk estimator. *Advances in neural information processing systems (NeurIPS)* 30 (2017).

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

[38] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *International conference on machine learning (ICML)*. PMLR, 3734–3743.

[39] Xiaoyun Li, Guangba Yu, Pengfei Chen, Hongyang Chen, and Zhekang Chen. 2022. Going through the life cycle of faults in clouds: Guidelines on fault handling. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 121–132.

[40] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*. Springer, 3–20.

[41] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining (ICDM)*. IEEE, 413–422.

[42] Jinyang Liu, Wenwei Gu, Zhuangbin Chen, Yichen Li, Yuxin Su, and Michael R Lyu. 2024. MTAD: Tools and Benchmarks for Multivariate Time Series Anomaly Detection. *arXiv preprint arXiv:2401.06175* (2024).

[43] Jinyang Liu, Junjie Huang, Yintong Huo, Zhihan Jiang, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Minzhi Yan, and Michael R Lyu. 2023. Scalable and Adaptive Log-based Anomaly Detection with Expert in the Loop. *arXiv preprint arXiv:2306.05032* (2023).

[44] Jinyang Liu, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Cong Feng, Zengyin Yang, and Michael R Lyu. 2023. Practical Anomaly Detection over Multivariate Monitoring Metrics for Online Services. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 36–45.

[45] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*. 412–426.

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

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

[48] Pieter-Jan Maenhaut, Bruno Volckaert, Veerle Ongenae, and Filip De Turck. 2020. Resource management in a containerized cloud: Status and challenges. *Journal of Network and Systems Management* 28 (2020), 197–246.

[49] Alireza Sadeghi Milani and Nima Jafari Navimipour. 2016. Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. *Journal of Network and Computer Applications* 71 (2016), 86–98.

[50] Hiep Nguyen, Yongmin Tan, and Xiaohui Gu. 2011. Pal: Propagation-aware anomaly localization for cloud hosted distributed applications. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. 1–8.

[51] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[52] Rabi Prasad Padhy. 2013. Big data processing with Hadoop-MapReduce in cloud systems. *International Journal of Cloud Computing and Services Science* 2, 1 (2013), 16.

[53] Daehyung Park, Yuuna Hoshi, and Charles C Kemp. 2018. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters* 3, 3 (2018), 1544–1551.

[54] Manjula Peiris, James H Hill, Jorgen Thelin, Sergey Bykov, Gabriel Kliot, and Christian Konig. 2014. Pad: Performance anomaly detection in multi-server distributed systems. In *IEEE 7th International Conference on Cloud Computing*. IEEE, 769–776.

[55] Rodolfo Picoreti, Alexandre Pereira do Carmo, Felippe Mendonca 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)*. IEEE, 776–784.

[56] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. 2009. Cloud computing: An overview. In *IEEE international conference on cloud computing*. Springer, 626–631.

[57] 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)*. 3009–3017.

[58] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning (ICML)*. PMLR, 1278–1286.

[59] Dominik Scheinert, Alexander Acker, Lauritz Thamsen, Morgan K Geldenhuys, and Odej Kao. 2021. Learning dependencies in distributed cloud applications to identify and localize anomalies. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, 7–12.

[60] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13, 7 (2001), 1443–1471.

[61] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019. ?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference (WWW)*. 3215–3222.

[62] Lifeng Shen, Zhuocong Li, and James Kwok. 2020. Timeseries anomaly detection using temporal hierarchical one-class network. *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), 13016–13026.

[63] Xuanhua Shi, Zhixiang Ke, Yongluan Zhou, Hai Jin, Lu Lu, Xiong Zhang, Ligang He, Zhenyu Hu, and Fei Wang. 2019. Deca: A garbage collection optimizer for in-memory data processing. *ACM Transactions on Computer Systems (TOCS)* 36, 1 (2019), 1–47.

[64] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.

[65] 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)*. 2828–2837.

[66] Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. 2022. TranAD: deep transformer networks for anomaly detection in multivariate time series data. *Proceedings of the VLDB Endowment* 15, 6 (2022), 1201–1214.

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

[68] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. 2021. Groot: An event-graph-based approach for root cause analysis in industrial settings. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 419–429.

[69] Wenlu Wang, Pengfei Chen, Yibin Xu, and Zilong He. 2022. Active-MTSAD: multivariate time series anomaly detection with active learning. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 263–274.

[70] Yaohui Wang, Guozheng Li, Zijian Wang, Yu Kang, Yangfan Zhou, Hongyu Zhang, Feng Gao, Jeffrey Sun, Li Yang, Pochian Lee, et al. 2021. Fast outage analysis of large-scale production clouds with service correlation mining. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 885–896.

[71] Jonathan Stuart Ward and Adam Barker. 2012. Semantic based data collection for large scale cloud systems. In *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date*. 13–22.

[72] Cheng Wen, Haijun Wang, Yuekang Li, Shengchao Qin, Yang Liu, Zhiwu Xu, Hongxu Chen, Xiaofei Xie, Geguang Pu, and Ting Liu. 2020. Memlock: Memory usage guided fuzzing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 765–777.

[73] Jianping Weng, Jessie Hui Wang, Jiahai Yang, and Yang Yang. 2018. Root cause analysis of anomalies of multitier services in public clouds. *IEEE/ACM Transactions on Networking (TON)* 26, 4 (2018), 1646–1659.

[74] Yulun Wu, Zeliang Yu, Ming Wen, Qiang Li, Deqing Zou, and Hai Jin. 2023. Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1046–1058.

[75] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853* (2015).

[76] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 world wide web conference (WWW)*. 187–196.

[77] Shifu Yan, Caihua Shan, Wenyi Yang, Bixiong Xu, Dongsheng Li, Lili Qiu, Jie Tong, and Qi Zhang. 2022. CMMD: Cross-Metric Multi-Dimensional Root Cause Analysis. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4310–4320.

[78] Shili Yan, Bing Tang, Jincheng Luo, Xing Fu, and Xiaoyuan Zhang. 2021. Unsupervised anomaly detection with variational auto-encoder and local outliers factor for kpis. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 476–483.

[79] Guangba Yu, Pengfei Chen, Zilong He, Qiuyu Yan, Yu Luo, Fangyuan Li, and Zibin Zheng. 2024. ChangeRCA: Finding Root Causes from Software Changes in Large Online Systems. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 24–46.

[80] Guangba Yu, Pengfei Chen, Pairui Li, Tianjun Weng, Haibing Zheng, Yuetang Deng, and Zibin Zheng. 2023. Logreducer: Identify and reduce log hotspots in kernel on the fly. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1763–1775.

[81] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

[82] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. 2019. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 1409–1416.

[83] Jianping Zhang, Wenwei Gu, Yizhan Huang, Zhihan Jiang, Weibin Wu, and Michael R Lyu. 2024. Curvature-Invariant Adversarial Attacks for 3D Point Clouds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 7142–7150.

[84] Jianping Zhang, Yizhan Huang, Weibin Wu, and Michael R Lyu. 2023. Transferable adversarial attacks on vision transformers with token gradient regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16415–16424.

[85] Xiaoxia Zhang, Shang Shi, HaiChao Sun, Degang Chen, Guoyin Wang, and Kesheng Wu. 2024. ACVAE: A novel self-adversarial variational auto-encoder combined with contrast learning for time series anomaly detection. *Neural Networks* 171 (2024), 383–395.

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

[87] Guoliang Zhao, Safwat Hassan, Ying Zou, Derek Truong, and Toby Corbin. 2021. Predicting performance anomalies in software systems at run-time. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–33.

[88] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. 2020. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 841–850.

[89] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, et al. 2020. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 162–171.

[90] Nengwen Zhao, Junjie Chen, Zhaoyang Yu, Honglin Wang, Jiesong Li, Bin Qiu, Hongyu Xu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. Identifying bad software changes via multimodal anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 527–539.

[91] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. Pyod: A python toolbox for scalable outlier detection. *arXiv preprint arXiv:1901.01588* (2019).

[92] Renyi Zhong, Yichen Li, Jinxi Kuang, Wenwei Gu, Yintong Huo, and Michael R Lyu. 2024. Automated Defects Detection and Fix in Logging Statement. *arXiv preprint arXiv:2408.03101* (2024).

[93] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations (ICLR)*.