This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

# A Spatiotemporal Deep Learning Approach for Unsupervised Anomaly Detection in Cloud Systems

Zilong He, Pengfei Chen, Xiaoyun Li, Yongfeng Wang, Guangba Yu, Cailin Chen, Xinrui Li, and Zibin Zheng, *Senior Member, IEEE*

***Abstract*— Anomaly detection is a critical task for maintaining the performance of a cloud system. Using data-driven methods to address this issue is the mainstream in recent years. However, due to the lack of labeled data for training in practice, it is necessary to enable an anomaly detection model trained on contaminated data in an unsupervised way. Besides, with the increasing complexity of cloud systems, effectively organizing data collected from a wide range of components of a system and modeling spatiotemporal dependence among them become a challenge. In this article, we propose TopoMAD, a stochastic seq2seq model which can robustly model spatial and temporal dependence among contaminated data. We include system topological information to organize metrics from different components and apply sliding windows over metrics collected continuously to capture the temporal dependence. We extract spatial features with the help of graph neural networks and temporal features with long short-term memory networks. Moreover, we develop our model based on variational auto-encoder, enabling it to work well robustly even when trained on contaminated data. Our approach is validated on the run-time performance data collected from two representative cloud systems, namely, a big data batch processing system and a microservice-based transaction processing system. The experimental results show that TopoMAD outperforms some state-of-the-art methods on these two data sets.**

***Index Terms*— Cloud computing, neural networks, unsupervised anomaly detection, variational auto-encoder (VAE).**

## I. Introduction

WITH the rapid development of cloud technology, the complexity and scale of cloud systems are continually increasing, which gives rise to frequent system accidents and the downgraded performance. To ensure the performance and reliability of a cloud system, operators need to monitor its state continuously. Metrics, such as CPU usage, number of I/O requests per second, network throughput, and so on, can provide significant insights into the running state of a cloud system, which helps the anomaly detection and troubleshooting.

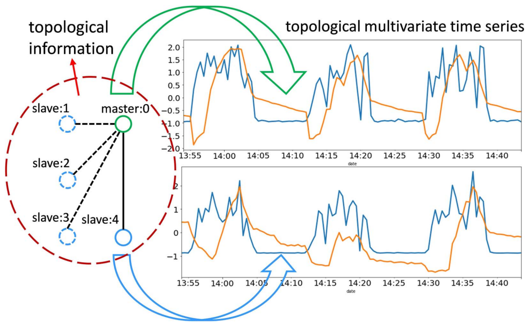

Fig. 1. Example of multivariate time series with topological information.

To reduce the cost of system accidents with the use of metrics we collect, anomaly detection on these metrics is first performed. Since metrics collected are time series data, an efficient and accurate algorithm for anomaly detection on time series is needed. So far, with the trend of developing artificial intelligence for IT operations [1], there have been plenty of works conducted on anomaly detection for time series with the use of AI technology. Some of them perform anomaly detection on univariate time series [2]–[6]. While with the increasing complexity of cloud systems, more and more metrics can be collected and utilized to model a system state [7]. As a result, methods only designed for univariate time series inevitably reveal their weakness. There also exist some methods which take multivariate time series as input [8]–[12], but they only concatenate different time series directly without effective organization. In a complex cloud system, more information about the collected metrics can be obtained and utilized to help organize these metrics. For example, the topology (see Fig. 1) originated from the system can help gain a graph-based representation of the system state, which we also denote as a topological multivariate time series below. Especially, in a microservice system, it becomes more critical to leverage the topological information. However, the problem of how to effectively construct and further perform anomaly detection for such graph-based representations still remains unsettled.

A deep neural network is a powerful tool for modeling dependence in data with complex structure. Therefore, its application for anomaly detection has attracted many

Manuscript received October 4, 2019; revised April 12, 2020 and July 26, 2020; accepted September 22, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1804002, in part by the Research Development Plan of Key Areas in Guangdong Province under Grant 2020B010165002, in part by the Natural Science Foundation of China under Grant U1811462 and Grant 61802448, in part by the Basic and Applied Basic Research of Guangzhou under Grant 202002030328, and in part by the Natural Science Foundation of Guangdong Province under Grant 2019A1515012229. *(Corresponding author: Pengfei Chen.)*

The authors are with the School of Data Science and Computer, Sun Yat-sen University, Guangzhou 510006, China (e-mail: hezlong@mail2.sysu.edu.cn; chenpf7@mail.sysu.edu.cn; lixy223@mail2.sysu.edu.cn; wangyf226@mail2.sysu.edu.cn; yubg5@mail2.sysu.edu.cn; chenclin@mail2.sysu.edu.cn; hongwen000@outlook.com; zhzibin@mail.sysu.edu.cn).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.3027736

2162-237X © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

researchers' attention in recent years. Recent research [3], [12]–[14] discovers that a variational auto-encoder (VAE) [15] shows superiority for the task of anomaly detection. Therefore, we develop an anomaly detector based on the design of a VAE. Beyond its advantages declared in the previous literature, we also demonstrate its effectiveness for training on contaminated data (i.e., data including normal data and abnormal data) without reliance on labels.

In this article, we propose a topology-aware multivariate time series anomaly detector (TopoMAD), which combines graph neural networks [16], [17], long short-term memory (LSTM) [18], [19], and VAE [15] to perform unsupervised anomaly detection for a cloud system. We evaluate our model using metrics collected from an environment running big data batching systems, such as Hadoop and Spark, with fault injections and metrics collected from a microservice-based application, where faults are injected occasionally.

The contributions of TopoMAD are summarized as follows.

1) TopoMAD introduces an unsupervised anomaly detection approach, which considers the topological information originated from a cloud system. We combine this topological information with metrics collected from a cloud system to construct a graph-based representation for anomaly detection.
2) TopoMAD glues graph neural networks and LSTM as the basic structure of VAE to perform anomaly detection in a topological time series. We make use of state-of-the-art graph neural networks, such as graph convolution network (GCN) [16] and graph attention network (GAT) [17] to extract information from the metrics organized in a predefined topology, together with LSTM to extract information from sliding windows over time. The spatiotemporal information extracted by graph neural networks together with LSTM, can help improve the anomaly detection performance in cloud systems.
3) TopoMAD makes use of VAE [15], a stochastic model, to perform anomaly detection for a cloud system in a fully unsupervised way. Instead of making an assumption of data we use, we train our model on all data we collect, no matter it is normal or abnormal. Moreover, we propose an unsupervised threshold selection method and examine this method based on our model.
4) We open-sourced two data sets,[1] including metrics collected from an environment running big data batching systems and metrics collected from a running microservice system. These data sets can be used to reproduce the results of this article or to motivate further research.

This article is organized as follows. The related work is introduced in Section II. In Section III, we present the preliminaries of each main technique in TopoMAD. Section IV outlines and specifies our approach. In Section V, we evaluate our approach using data collected from two application scenarios. Section VI concludes this article.

## II. Related Work

### A. Unsupervised Anomaly Detectors

Enormous methods of unsupervised anomaly detection have been developed in the past years. In this section, we categorize them into traditional approaches and deep learning-based approaches.

Traditional approaches [20] include statistical approaches, such as Gaussian-based model, classification-based approaches, such as one-class support vector machines (OC-SVMs) [21], nearest neighbor-based approaches, such as local outlier factor (LOF) [22], and so on. Usually, a traditional anomaly detector is associated with an assumption about the normal data, and its effectiveness will degrade when the data cannot fit in the corresponding assumption.

Compared with traditional approaches, deep learning-based models are better at modeling complex dependence in data, and thus, gain a lot of attention. For example, deep auto-encoder [23] can be used to perform dimension reduction by utilizing its multiple nonlinear transformations. For anomaly detection, the reconstruction error of an observation is used as its anomaly score. For anomaly detection on time series, Donut [3] leverages VAE [15] to model the reconstruction probability [13] of univariate time series and perform anomaly detection based on this measure. LSTM-AD [8] learns a prediction model using stacked LSTM networks, and the prediction error is used as the measure of an anomaly. LSTM-ED [9] proposes an LSTM-based seq2seq auto-encoder that learns to reconstruct normal multivariate time series and uses the reconstruction error to detect anomalies. Deep structured energy-based models (DSEBMs) [10] connects an EBM with a regularized auto-encoder to model the data distribution, and the energy score or the reconstruction error is used to perform anomaly detection. For sequential data, a recurrent formulation of EBMs can be employed. Though some of these approaches have tackled the problem of modeling complex temporal dependence for time series, we still need an approach to effectively model the spatiotemporal dependence for data which are collected from a specific topology continuously.

### B. Anomaly Detection in Cloud Systems

In Section II-A, we have given a brief introduction about those general unsupervised anomaly detectors. Most of them have been used in cloud systems [3], [24], [25]. In this section, we shall focus on some other aspects of anomaly detection in cloud systems.

*1) Anomaly Types:* According to some work [26]–[31], anomalies in cloud systems can be categorized into two types, including external impairments and internal application faults. External impairments refer to unexpected overloads, such as a memory hog, induced by another co-located application or infrastructure failures, such as disk failure, network disconnection, OS crash, and so on. Internal application faults denote anomalies caused by the misconfiguration or software bugs of the corresponding running application. Different causes can result in different symptoms. Multiple work [26], [29], [30] has been conducted on the analysis of failure characteristics in cloud systems. For example, Zhou *et al.* [29] investigate application faults in microservice systems with different root causes and the debugging practice for them.

*2) Anomaly Indicators:* There are multiple types of indicators which can reveal the state of a cloud system and further help detect and analyze anomalies. Examples of these types include metrics [3], [5], [32], [33], logs [34]–[37], and system

[1]https://github.com/QAZASDEDC/TopoMAD

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

calls [38], [39]. The anomaly detection process varies according to different types of indicators. In this article, we focus on anomaly detection in cloud systems with the use of system metrics. System metrics, such as resource utilization and I/O response time, are widely used as anomaly indicators in cloud systems and have proved their effectiveness in plenty of works [5], [28], [32], and [40]. Also, there exist works which succeed in finding new effective anomaly indicators in some specific scenarios, such as queue lengths of microservices used in [40].

*3) Temporal Model:* Many indicators presented in Section II-B2 can be aggregated as time-series data by applying a sliding window, which helps reveal the system's local state in time. Some work [41] calculates statistical features (such as mean, standard deviation, and gradient) of data in the sliding window as temporal features. There also exists complex but more powerful temporal modeling techniques, such as time series decomposition [32], spectral residual [6], and some deep learning-based methods [3], [8]–[10] mentioned in Section II-A.

*4) Spatial Model:* Copious literature on system performance diagnosis involves constructing a graph-based representation for a system. Some of them [11], [42]–[44] analyze metrics collected from a system in a pairwise way and construct an invariant graph which takes each metric as a node. On the other hand, some work [27], [45], [46] treat each of the system components as a node with multiple attributes and employ edges to represent the connectivity between system components. In this article, we prefer the latter way because analyzing metrics pairwise is time consuming and infeasible for anomaly detection in real time when more and more metrics are collected from cloud systems.

*5) Threshold Selection:* To apply an unsupervised anomaly detector to practice, a corresponding threshold is a necessary and influential hyperparameter. Threshold selection is integral to anomaly detection in cloud systems. Some work [11], [12], [47] has provided some practices for threshold selection. For instance, peaks-over-threshold (POT) [12] utilizes the extreme value theory [2] to perform automatic threshold selection. As for dynamic threshold calculation, nonparametric-dynamic-thresholding (NDT) [47] selects a threshold from the set $\epsilon = \mu(e_s) + z\sigma(e_s)$ with $\mu(e_s)$ and $\sigma(e_s)$ as the mean and standard deviation of anomaly scores over a sliding window and $z$ chosen from a range between two and ten to maximize an equation predefined by us.

### C. Graph Neural Networks for Spatiotemporal Modeling

Learning patterns from spatiotemporal graphs is increasingly important in many applications. Many current approaches [48] apply graph neural networks together with RNN and CNN to simultaneously consider spatial and temporal relations. For instance, diffusion convolutional recurrent neural network (RNN) [49] replaces the matrix multiplications in GRU [50] with a diffusion convolution operation to accomplish the task of traffic forecasting. CNN-GCN [51] overcomes this problem through a complete convolutional structure which interleaves 1-D-CNN with GCN. ST-GCN [52] extends a temporal flow as graph edges and then extracts both temporal and spatial features using a unified GCN to perform human action recognition. Inspired by these models, we develop TopoMAD to include and extract topological information with the help of graph neural networks to perform unsupervised anomaly detection in cloud systems.

## III. Preliminaries

### A. Cloud System Topologies

To illustrate what is the topological information originated from the system, we first define nodes and edges in cloud system topology and expound their characteristics.

*1) Nodes:* We have mentioned in Section II-B that each system component is treated as a node in the cloud system topology. The definition of a system component can vary depending on the different granularity of division. Examples of a course-grained division might be the role-based (e.g., master or slave) division in Hadoop and the service-based division in a microservice system, while examples of a fine-grained division might be the node-based division in Hadoop and the pod-based division in a Kubernetes driven microservice system. In a fine-grained topology, several nodes (e.g., pods) that can be grouped into one node (e.g., services) in a course-grained topology can share the same set of edges with similar behaviors.

*2) Edges:* Multiple relationships of system components can be chosen to be an edge. For example, components that locate in the same physical machine can share an edge, components whose load are balanced can share an edge, components that have the same update setting can share an edge, and so on. Most of these relationships are dynamic. In this article, we define there is an edge between two system components if they interact with each other. According to this definition, we can construct the topology in advance based upon some tracing-based analysis [53], [54] or network traffic correlation-based analysis [27], [55]–[57]. These methods are important complements to our approach.

### B. Problem Statement

We list the notations used in this article with their descriptions in Table I. Anomaly detection for a topological multivariate time series is to determine whether an observation $X_t$ is an anomaly or not. We divide this objective into two steps. First, we calculate an anomaly score $S(X_t|X_{t-W:t-1}, E)$ for $X_t$ considering its recent history $X_{t-W:t-1}$ and its topology $E$. Then, we obtain an anomaly result by comparing $S(X_t|X_{t-W:t-1}, E)$ with a threshold $\tau$, which is selected in an unsupervised manner. If the score is higher than $\tau$, an alert will be triggered.

### C. Basics of Graph Neural Networks

Graph neural networks are modern deep learning techniques designed for graph data. They are helpful when we need to model the spatial dependence in metrics collected from different components with connectivity and extract information from the topology. In this article, we apply two representative graph neural networks, namely, GCN [16] and GAT [17], as the basic layer in an LSTM cell to capture the spatial dependence in the topology. In the following, we will introduce

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
NOTATIONS AND DESCRIPTIONS

| Notations | Descriptions |
|---|---|
| $N$ | The number of nodes in the system. |
| $F$ | The number of metrics collected in each node. |
| $W$ | The size of the sliding window applied over the metrics. |
| $X$ | A multivariate time series recording metrics of nodes. |
| $X_t$ | An $N \times F$ matrix as the observation of $X$ at time $t$. |
| $C$ | The number of edges in the system topology. |
| $E$ | A $2 \times C$ matrix as the edge set array of the system topology. |
| $A$ | The adjacency matrix of the system topology. |
| $\tau$ | The threshold used to determine whether to trigger an alert. |

the layerwise propagation rules of these two techniques in detail.

**GCN** [16] proposes a graph convolution layer which accepts an $N \times F$ matrix $X^{(l)}$ containing features of each node and the adjacency matrix $A$ of the topology as inputs. It propagates forward with the following rule:

$$X^{(l+1)} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X^{(l)} \mathbf{W}^{(l)}. \tag{1}$$

Here, $\tilde{A} = A + I$ denotes the adjacency matrix with inserted self-loops. $\tilde{D}$ is its diagonal degree matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $\mathbf{W}^{(l)} \in \mathcal{R}^{F \times F'}$ is a layer-specific trainable weight matrix. An activation function can also be applied to the result $X^{l+1}$ to introduce nonlinearity.

**GAT** [17] is similar to GCN. It utilizes attention mechanisms [58], [59] to fuse the neighboring nodes and then learn hidden representations for each node in a graph. After inputting $X^{(l)} = \{X_1^{(l)}, X_2^{(l)}, \ldots, X_N^{(l)}\}$, new hidden representations can be computed as follows:

$$X_i^{(l+1)} = \frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{i,j}^k \mathbf{W}^k X_j^{(l)} \tag{2}$$

where the attention coefficients $\alpha_{i,j}$ are computed as

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^\top \left[\mathbf{W}\mathbf{x}_i \parallel \mathbf{W}\mathbf{x}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^\top [\mathbf{W}\mathbf{x}_i \parallel \mathbf{W}\mathbf{x}_k]\right)\right)}. \tag{3}$$

Here, $\mathbf{W} \in \mathcal{R}^{F' \times F}$ and $\vec{\mathbf{a}} \in \mathcal{R}^{2F'}$ are trainable parameters. $K$ denotes the number of independent attention mechanisms executed and $\mathcal{N}_i$ denotes the first-order neighbors of node $i$ (including node $i$). LeakyReLU$(\cdot)$ is kind of an activation function. Nonlinearity can also be introduced here by applying an activation function to the output.

## *D. Basics of Long Short-Term Memory Networks*

Recurrent neural networks (RNNs) are discrete-time state–space models which can recursively process an input sequence and model its temporal dependence. Since a simple RNN has limited ability to learn the long-term dependence in a sequence, LSTM was invented to address this problem. Using gating mechanisms, LSTM can decide whether to forget or update certain information in the transferred cell state $c_{t-1}$ considering the input state $x_t$ and transferred hidden state $h_{t-1}$, and then calculates new cell state $c_t$ and new hidden state $h_t$.

There have been some methods which apply RNN, including LSTM on graph-structured data. For example, Liang *et al.* [60] utilize confidence-driven search to specify the node updating sequence for a graph and then it sequentially update the states of all nodes with LSTM. Peng *et al.* [61] partition a document graph into two directed acyclic graphs and constructs the LSTMs accordingly. In the constructed LSTM, different precedents of a unit are calculated via full parametrization or edge-type embedding and then summed to obtain the output, states, and gates. Jain *et al.* [62] parameterize a spatiotemporal graph with a factor graph [63] and then represent each factor with an RNN. Compared with these methods, we directly replace the linear layers in LSTM with state-of-the-art graph neural layers to perform spatiotemporal learning. A more detailed description is given in Section IV-D.

## *E. Basics of Variational Auto-Encoder*

VAE [15] is a deep probabilistic graphical model. It is a powerful tool for modeling the relationship between observed variables $x$ and their corresponding latent variables $z$ with reduced dimension. VAE assumes that $x$ can be generated through a process which first samples $z$ from some prior distribution $p_\theta(z)$ and then samples $x$ from $p_\theta(x|z)$, which is derived from a neural network with parameter $\theta$. Since $p_\theta(z|x)$ is intractable, VAE introduces a recognition model $q_\phi(z|x)$ to approximate it. Parameters $\phi$ and $\theta$ can be trained through maximizing the variational lower bound $\mathcal{L}(\theta, \phi; x)$ [(4)] on the marginal likelihood with stochastic gradient variational Bayes estimator [15]

$$\mathcal{L}(\theta, \phi; x) = -D_{\text{KL}}\left(q_\phi(z|x) \| p_\theta(z)\right) + \mathbf{E}_{q_\phi(z|x)}(\log(p_\theta(x|z))). \tag{4}$$

A typical choice for $p_\theta(x|z)$ or $q_\phi(z|x)$ is a multivariate Gaussian with a diagonal covariance structure, whose mean and the covariance matrix is derived by neural networks. And in general, a standard normal distribution $\mathcal{N}(0, I)$ is chosen for the prior of $z$.

For the purpose of anomaly detection given a specific input $x$, reconstruction probability $\mathbf{E}_{q_\phi(z|x)}(\log(p_\theta(x|z)))$ [13] is adopted and calculated using Monte Carlo integration [64] as follows:

$$\mathbf{E}_{q_\phi(x|z)}(\log(p_\theta(x|z))) = \frac{1}{L} \sum_{l=1}^{L} \log\left(p_\theta\left(x|z^{(l)}\right)\right) \tag{5}$$

where $L$ denotes the number of samples and $z^{(l)}$, $l = 1, 2, \ldots, L$ are sampled from $q_\phi(z|x)$.

## IV. METHODOLOGY

In this section, we will introduce our motivation and then elaborate on the overall structure and other details of TopoMAD.

## *A. Motivation*

There are two key ideas in TopoMAD, namely, the inclusion of topological information to represent a system state and the unsupervised setting at the stage of model training and threshold selection.

Nowadays, most popular cloud systems can be recognized as distributed systems with multiple interconnected components. The components mentioned here might be virtual

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

machines, microservices, and so on. When we focus on a certain component in these systems, we can find the behavior of a component can be analyzed through the following two factors. One factor is the task the component is responsible for and currently performing. Metrics of the component can provide insights into this factor. Considering a single-selected metric can provide metric-level information while considering multiple metrics of the component can provide component-level information. The other factor is the interacting relationship of the components. Considering this can provide topological information because behaviors of a component can be induced by requests or return values of other components. The integration of topological information and component-level information of each component forms system-level information, which can provide a more comprehensive insight into the whole running system.

Previous research based on deep learning usually neglects the topological information. They either focus on anomaly detection on each particular component or even each particular metric of the whole system individually, or totally ignore the intrinsic communicating behaviors within a system and treat the whole system as a cumbersome component. We argue that paying attention to detecting anomalies at the system level with topological information, rather than only at the component level or metric level is worth considering. First, the decision of whether a component is in an abnormal state needs to consider its connected components sometimes. Second, training and maintaining an individual model for each component will become more and more labor intensive as the cloud system enlarges its complexity. Third, roughly treating the whole system as a cumbersome component loses insight into its inner topology and might, thus, raise the difficulty of modeling its normal behavior.

Including the topological information to represent a system state and utilizing graph neural networks to learn spatial dependence of metrics with the help of this topological information can bring three benefits. First, feature extractors of graph neural networks are shared amongst the same kind of metrics from different components, which helps catch similar pattern amongst the same metric type with unified feature learning. Second, the role of a particular component can be defined by its connections with other components through a graph neural operation, which provides convenience for performing end-to-end learning on patterns and behavior of all components from a system. Third, the topological information can guide models to concentrate attention on interactions of components with a direct connection in reality, which helps prevent overfitting of our model.

TopoMAD follows an unsupervised setting, which means that labels of data keep unavailable throughout the stage of model training and threshold selection. As a result, we cannot deliberately select normal data to be the training data set. In this case, a stochastic model, such as VAE, has superiority over a deterministic model, such as auto-encoder. Auto-encoder can be thought as a model which learns deterministic mappings from input data to their corresponding latent variables; therefore, when training data includes anomalies, auto-encoder can also learn to reconstruct them well, which increases the difficulty of detecting the same kind of anomalies in the testing phase. By contrast, variational auto-encoder learns the distribution of the latent variables instead of the latent variables themselves. The first RHS term $-D_{\mathrm{KL}}(q_\phi(z|x)\|p_\theta(z))$ in the training objective [i.e., (4)] makes input data "compete for" a suitable latent space by driving $q_\phi(z|x)$ close to a standard normal distribution. As a result, anomalies, which occur infrequently, will tend to lose the competition and result in a low reconstruction probability even when they have appeared in the training data.

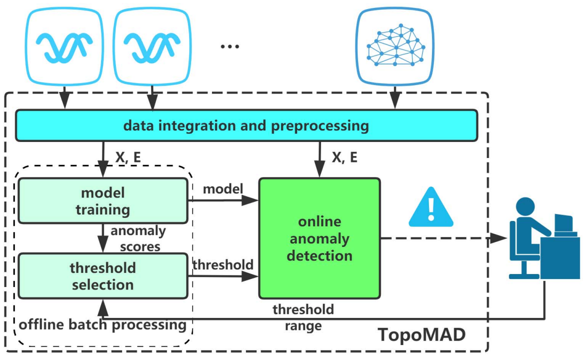

Fig. 2. Overall structure of TopoMAD.

Threshold selection is an essential phase of anomaly detection in practice. However, quite a few methods, such as [3] and [9] neglect this phase and only enumerate all thresholds to gain the best F1 score and use it as the performance metric of an anomaly detector. We argue that the best F1 score is not a good measure of the performance of an unsupervised anomaly detection model, because the performance achieved denoted by the best F1 score is impractical and unachievable without the support of labels. Therefore, we advise using an unsupervised way to select a threshold before evaluating model performance.

## *B. Overall Structure*

Fig. 2 displays the overall structure of TopoMAD. Data collected is processed to gain a tensor $X$ recording metrics collected from each node and an array $E$ which describes the system topology by recording its edges. In the stage of model training, the model is trained with historical data in an offline batch processing way. After the model is trained properly, we select a threshold according to the distribution of anomaly scores of the training data. Then, in the stage of online anomaly detection, the anomaly score of a new observation is calculated using this properly trained model. If the anomaly score of observation is higher than the threshold we select, an alert will be triggered.

## *C. Data Integration and Preprocessing*

Fig. 3 shows an example of an observation from a Hadoop cluster with one master node and four slave nodes. An example of the corresponding input data of Fig. 3 is visualized in Fig. 4.

During data preprocessing, we transform different metrics collected from different nodes through data standardization and then apply sliding windows of length $W$ over these observations. As for the edge set array $E$, we do not consider the construction of a dynamic topology in this article because

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

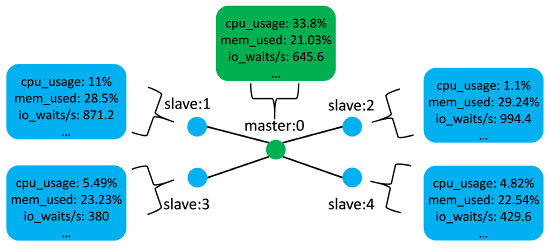

Fig. 3. Graph-based representation of metrics collected from a Hadoop cluster.

$$X=\{\cdots,X_{t-1},\boxed{X_t}\}$$

$$\begin{array}{c|ccc} & cpu\_usage & mem\_used & io\_waits/s \\ \hline master{:}0 & 33.8\% & 21.03\% & 645.6 \\ slave{:}1 & 11\% & 28.5\% & 871.2 \\ slave{:}2 & 1.1\% & 29.4\% & 994.4 \\ slave{:}3 & 5.49\% & 23.23\% & 380 \\ slave{:}4 & 4.82\% & 22.54\% & 429.6 \end{array}$$

$$E \Rightarrow \begin{bmatrix} 0 & 1 & 0 & 2 & 0 & 3 & 0 & 4 \\ 1 & 0 & 2 & 0 & 3 & 0 & 4 & 0 \end{bmatrix}$$

Fig. 4. Example of the input data in Fig. 3.

a real-time topology can change with high frequency owing to load balance or other reasons. It is hard to assign a real-time topology to each time step of the collected metrics.

## *D. Building Block: GraphLSTM*

Before presenting the network architecture of our model proposed in TopoMAD, we first introduce the basic building block to concurrently model the spatial and temporal dependence in the input data. We name this building block GraphLSTM for its nature as a combination of graph neural networks and LSTM. Specifically, we replace fully connected layers in LSTM with graph neural layers, such as GCN [16] and GAT [17] to build a GraphLSTM cell. Fig. 5 shows the inner structure of GraphLSTM.

All the states and gates in GraphLSTM are graph-based representations containing features of each node and a topology denoted by an input edge set array $E$. When a new input $X_t$ from a time series $X$ arrives at time $t$, GraphLSTM updates the cell state $c_t^i$ and calculates the new hidden state $h_t^i$ of a certain node $i$ in a topology considering the input $X_t^{\mathcal{N}_i}$ and past states $c_{t-1}^{\mathcal{N}_i}$, $h_{t-1}^{\mathcal{N}_i}$ of itself and its neighbors. The computations conducted by GraphLSTM can be formulated as (6) in the following, where "$*_{\mathcal{G}}$" denotes the graph neural operator and "$*$" denotes the Hadamard product:

$$\begin{aligned} f_t &= \text{sigmoid}\big(\mathbf{W}_f *_{\mathcal{G}} ([h_{t-1},x_t],E)+\mathbf{b}_f\big)\\ i_t &= \text{sigmoid}\big(\mathbf{W}_i *_{\mathcal{G}} ([h_{t-1},x_t],E)+\mathbf{b}_i\big)\\ g_t &= \tanh\big(\mathbf{W}_g *_{\mathcal{G}} ([h_{t-1},x_t],E)+\mathbf{b}_g\big)\\ c_t &= f_t * c_{t-1} + i_t * g_t\\ o_t &= \text{sigmoid}\big(\mathbf{W}_o *_{\mathcal{G}} ([h_{t-1},x_t],E)+\mathbf{b}_o\big)\\ h_t &= o_t * \tanh(c_t). \end{aligned} \tag{6}$$

## *E. Network Architecture*

In this section, we will introduce the detailed neural network architecture of the model proposed in TopoMAD. Fig. 6 shows a visualization of our model at the stage of inference.

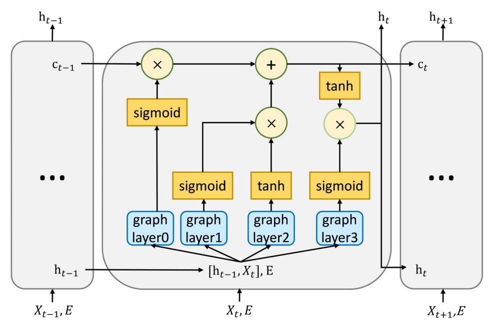

Fig. 5. Inner structure of GraphLSTM.

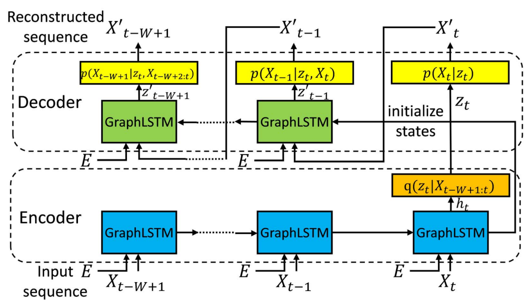

Fig. 6. Overall graphical model of TopoMAD at the stage of inference.

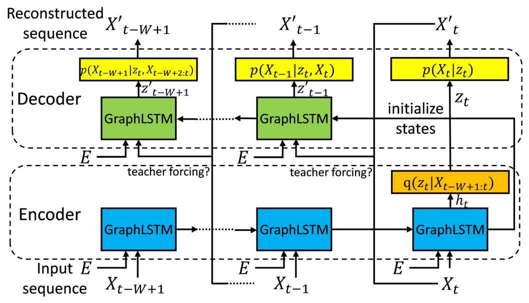

Fig. 7. Overall graphical model of TopoMAD at the stage of training.

The overall network is a stochastic seq2seq auto-encoder [65]. Let $t_0 = t - W + 1$ and $X_{t_0:t} = \{X_{t_0}, X_{t_0+1}, \ldots, X_{t-1}, X_t\}$ denote an input sequence and $E$ denote an input edge set array. The encoder computes the distribution parameters of $q(z_t|X_{t_0:t})$ after iteratively feeding every time step of $X_{t_0:t}$ into it. GraphLSTM is used as part of the encoder to capture the spatiotemporal dependence among nodes and long time. After consuming the entire input sequence, we get a summarized fixed-dimension tensor $h_t$ which is further passed to a couple of multilayer perceptrons to estimate the parameters $\mu(X_{t_0:t})$ and $\sigma(X_{t_0:t})$ of $q(z_t|X_{t_0:t}) = \mathcal{N}(\mu(X_{t_0:t}), \sigma(X_{t_0:t}))$. Multilayer perceptrons here are learned individually for each node. Likewise, similar architecture is adopted as the decoder to reconstruct the sequence in reverse order, and the states of the decoder are initialized using the final states of the encoder. During training, the input for the decoder per time step comes from the original input sequence with probability $\lambda$ or the reconstructed sequence with probability $1 - \lambda$ (see Fig. 7), namely, a scheduled sampling process [66]. In our model, $\lambda$ is initialized to be 1 and will reduce by half at the end

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

of an epoch if the validation loss does not decrease. While during inference, the input for the decoder per time step always comes from its preceding reconstruction (see Fig. 6). Specifically, at time step $t-1$, we use the mean value, $\mu(z_t)$, of $p(X_t|z_t) = \mathcal{N}(\mu(z_t), \sigma(z_t))$ as the input for the decoder.

### F. Offline Model Training

Let $\phi$ denote the encoder network parameters and $\theta$ denote the decoder network parameters. Similar to VAE, our model is trained by optimizing the variational lower bound on the marginal likelihood. Given a topological time series $X_{t_0:t}$, where $t_0 = t - W + 1$ denotes the beginning of the sliding window, the loss function is derived as follows:

$$\begin{aligned}\mathcal{L}(\theta, \phi; X_{t_0:t}) &= -D_{\mathrm{KL}}\big(q_\phi(z_t|X_{t_0:t})\|p_\theta(z_t)\big)\\ &\quad + \mathbf{E}_{q_\phi(z_t|X_{t_0:t})}\big(\log\big(p_\theta(X_{t_0:t}|z_t)\big)\big)\\ &= -D_{\mathrm{KL}}\big(q_\phi(z_t|X_{t_0:t})\|\mathcal{N}(0, I)\big)\\ &\quad + \frac{1}{L}\sum_{l=1}^{L}\sum_{j=t_0}^{t}\log\Big(p_\theta\Big(X_j|z_t^{(l)}, X_{j+1:t}\Big)\Big). \end{aligned} \quad (7)$$

Here, $L$ denotes the sampling number to estimate the expectation. In this article, we set $L = 1$ since it has been reported in [15] that one sample is sufficient as long as the minibatch size is large enough. The pseudocode for training the model GraphLSTM-VAE in TopoMAD is shown in Algorithm 1 in Appendix A of the supplementary material.

### G. Computing Anomaly Scores

As shown in Section III-E, the reconstruction probability $\mathbf{E}_{q_\phi(z_t|X_{t_0:t})}(\log(p_\theta(X_t|z_t)))$ can be immediately calculated at time $t$ and used to indicate whether $X_t$ is anomalous. We follow it and use its additive inverse as the anomaly score of an observation $X_t$ to make a higher anomaly score imply a more anomalous observation. The temporary anomaly score at time $t$ is formulated as follows:

$$\mathrm{temp}S_t = -\mathbf{E}_{q_\phi(z_t|X_{t_0:t})}(\log(p_\theta(X_t|z_t))). \quad (8)$$

It has been reported by [3] that anomalous observations usually occur continuously. In practice, it is acceptable to trigger an alert within a short delay. We allow the anomaly score of an observation $X_t$ to be adjusted according to some of its succeeding observations. As a stochastic sequence-to-sequence model, the model in TopoMAD not only learns to reconstruct $X_t$ but also learns to reconstruct $W-1$ observations preceding to $X_t$, which provides convenience for us to update the anomaly score of an observation $X_t$. The final anomaly score $S_t$ for an observation $X_t$ is formulated as follows:

$$S_t = -\frac{1}{L*D}\sum_{d=0}^{D-1}\sum_{l=1}^{L}\log\Big(p_\theta\Big(X_t|z_{t+d}^{(l)}, X_{t+1:t+d}\Big)\Big) \quad (9)$$

where $L$ denotes the sampling number and $D$ denotes the number of times we calculate or update an anomaly score (i.e., the tolerance of detection delay).

In some cases, a relative low reconstruction probability of a particular component in a system is enough to draw our attention. Therefore, we also calculate an anomaly score from a component perspective as follows:

$$S_t = -\max_{0\le i<N}\frac{1}{L*D}\sum_{d=0}^{D-1}\sum_{l=1}^{L}\log\Big(p_\theta\Big(X_t^i|z_{t+d}^{(l)}, X_{t+1:t+d}\Big)\Big) \quad (10)$$

where $N$ denotes the number of components in the system and $X_t^i$ denotes the metrics of component $i$ at time $t$.

For online anomaly detection, whether to trigger an alert at time $t$ is decided based upon temp$S_t$, which can be immediately calculated at time $t$ and $D-1$ consecutive updated anomaly scores preceding to it. When one of these anomaly scores is higher than the threshold, an anomaly is detected. Algorithm 2 in Appendix A of the supplementary material shows the pseudocode of our online anomaly detection algorithm.

### H. Threshold Selection

An anomaly score can reveal how anomalous an observation is, but in practice, a threshold is still needed to trigger an alert and instruct operators to take actions.

We propose a threshold selection method based on an assumption that anomaly scores of normal data locate in an area with a high density, while anomaly scores of abnormal data locate in an area with a low density. The distance of these two areas is relatively long. Specifically, we define the distance of two anomaly scores sets $S_{<\tau}$ and $S_{>\tau}$ separated by a threshold $\tau$ as follows:

$$d(S_{<\tau}, S_{>\tau}) = \frac{\min(S_{>\tau}) - \max(S_{<\tau})}{\min(S_{>\tau}) + \max(S_{<\tau}) - 2*\min(S_{<\tau})} \quad (11)$$

where $\max(S)$ denotes the maximal element in $S$, and $\min(S)$ denotes the minimal element.

Based upon this assumption, we select a threshold which maximizes the distance [11)] between the two sets cut from the training data set by this threshold from a range provided by an operator. The pseudocode of the threshold selection algorithm is demonstrated in Algorithm 3 in Appendix A of the supplementary material.

If operators want the selected threshold to adapt to the dynamic changes in the system, they can use the automatic threshold selection method on newly collected data and update the selected threshold routinely to maintain its effectiveness.

## V. Experiment Validation

In this section, we conduct experiments to validate our model and answer the following questions.

1) Can TopoMAD outperform other approaches in anomaly detection for topological multivariate time series collected from a cloud system?
2) How does each component of TopoMAD affect the performance?
3) How effective is the topology?
4) How can we interpret the results of TopoMAD?
5) How is the robustness of TopoMAD?
6) How is the efficiency of TopoMAD?
7) How effective is our unsupervised threshold selection method in TopoMAD? Can it recommend a relatively better threshold?

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE II
FEATURES OF OUR TWO DATA SETS

| Features | MBD | MMS |
|---|---|---|
| Total Observations | 8640 | 4370 |
| Total Components | 5 | 50 |
| Metrics per Component | 26 | 7 |
| Anomaly Ratio | 6.55% | 7.03% |

### A. Data Sets

To demonstrate the effectiveness of our model, we conduct experiments using two data sets, of which one is collected from an environment running a big data batch processing system (MBD), and the other is from a microservice-based transaction processing system (MMS). The features of these two data sets are shown in Table II.

We obtain the data set MBD from a cluster containing five nodes with one master and four slaves. During the experiments, we continuously generate multiple random workloads with random parameters using HiBench [67], a big data benchmark published by Intel. At the same time, random faults, including external impairments and application faults, are injected irregularly with random parameters. The injected external impairments include system resource hog (high CPU/memory/disk-IO load), network failure (delay or packets loss), and the application faults are simulated through injecting some delays or exceptions into Hadoop distributed file system (HDFS) (causing symptoms similar to HDFS-448 [68] and HDFS-8160 [69]). We monitor and collect 26 metrics per node, including CPU idle, CPU I/O wait, CPU softirq, CPU system, CPU user, disk I/O wait per second, disk I/O in progress per second, disk used percentage, disk read speed, disk write speed, kernel entropy, load1, load5, load15, memory active, memory available percentage, memory cached, memory dirty, memory free, memory used percentage, network bytes received rate, network bytes sent rate, network TCP time wait, number of processes blocked, number of running processes, and the total number of processes. These metrics are selected for two reasons. First, they are representative system metrics used in a series of previous works [12], [27], [32], [43], [46], and we try to cover many types, ranging from CPU, memory, disk, and network to processes, as we can to gain a thorough insight into the running state of the system. In addition, these metrics are easy to acquire using tools, such as Perf, Telegraf [70], Amazon CloudWatch [71], and so on. These metrics are collected for three days and constitute the data set MBD with anomalies labeled on the basis of the injection log. We can refer to Fig. 3 for the topology input of this data set. There is no doubt that other performance metrics can be fed into our algorithm.

With respect to the data set MMS, we adopt Hipster-Shop [72], a web-based e-commerce microservice benchmark where users can browse hipster goods, add them to the cart and purchase them. This benchmark is deployed in a Kubernetes [73] cluster with 12 VMs. A load generator is included in Hipster-Shop to mimic visits to the website. Moreover, we inject anomalies with random parameters, such as CPU/disk-IO hog, network delay, and container hang to some pods during the experiments. Metrics, including CPU usage, memory usage, network receive rate, network transmit rate, pod latency, pod workload, and pod success rate, are collected from each pod and recorded in the data set MMS. Then anomalies are labeled according to the fault injection log. The pod-level topology, which can be used to construct a graph-based representation for metrics in this data set, is demonstrated in Fig. 8.

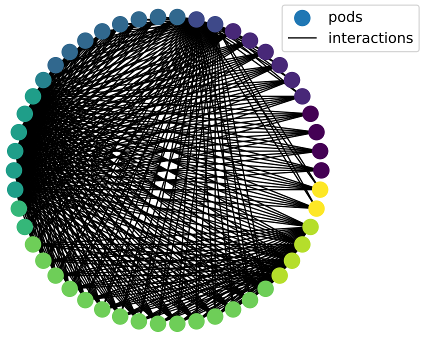

Fig. 8. Pod-level topology of the Hipster-Shop where pods belonging to the same service are plotted with the same color.

### B. Performance Metrics

All the anomaly detectors we evaluate in this article can return an anomaly score for each observation, and therefore, we consider using the average precision (AP) of the abnormal class as the performance metric to compare them. AP is considered more suitable than Area Under the receiver operating characteristic Curve (AUC) when dealing with a highly skewed data set [74], which means the normal data dominates the whole data set. It can be calculated as follows:

$$P = \frac{TP}{TP+FP}, \quad R = \frac{TP}{TP+FN}$$
$$AP = \sum_i (R_i - R_{i-1})P_i, \quad mAP = \frac{\sum AP_C}{N(\text{Classes})} \tag{12}$$

where $TP$, $FP$, and $FN$ refer to true positive, false positive, and false negative, respectively, and $P_i$ and $R_i$ denote the precision and recall at the $i$th threshold. The mean of $AP$ from all classes is known as mAP.

F1 score, the harmonic mean of precision and recall, is also widely used as a performance metric for anomaly detection when the observations have been classified into normal or abnormal with a threshold. It can be calculated as follows:

$$F1 = 2 \cdot \frac{P \cdot R}{P+R}. \tag{13}$$

We use the F1 score to compare the effectiveness of different threshold selection methods.

### C. Experiment Setup

We compare our model in TopoMAD with seven baseline anomaly detectors, including a Gaussian-based anomaly detector, OC-SVM [21], LOF [22], a simple auto-encoder [23], LSTM-AD [8], LSTM-ED [9], and RNN-EBM [10], which have been introduced in Section II-A. Since the anomaly detection for metrics with the help of topological information is kind of an unprecedented attempt, for other anomaly detectors, we conduct experiments in the following two ways. The first way is to concatenate all metrics from different components into one huge vector and use this vector as the input of the evaluated anomaly detector. The second way is to perform

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

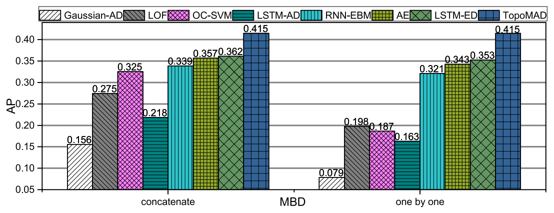

Fig. 9. AP of TopoMAD and other baseline anomaly detectors on MBD.

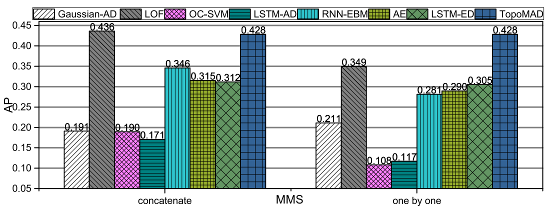

Fig. 10. AP of TopoMAD and other baseline anomaly detectors on MMS.

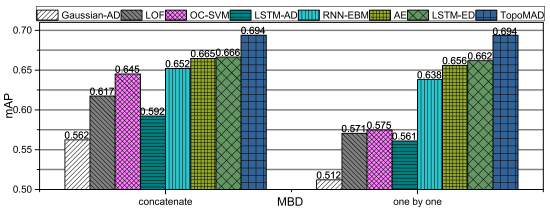

Fig. 11. mAP of TopoMAD and other baseline anomaly detectors on MBD.

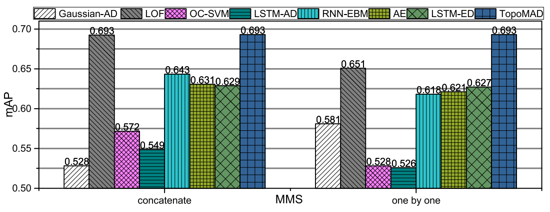

Fig. 12. mAP of TopoMAD and other baseline anomaly detectors on MMS.

anomaly detection on each component one by one and use the average anomaly score of all components as the anomaly score of the whole cloud system.

We employ Pytorch and Pytorch Geometric [75] to implement the model in TopoMAD and its variants. Then, we train them and other baseline anomaly detectors on a server with four NVIDIA Tesla P100-SXM2 GPUs.

Hyperparameters are chosen based on Bayesian optimization [76], [77]. Since a lightweight model is usually preferred when performing anomaly detection in cloud systems, we will not consider using a wide and deep network architecture in our experiments of deep learning-based methods. For both data sets, we set the time window size as 10, which spans 5 min in MBD and 10 min in MMS. We set the hidden dim per component as 3, the number of recurrent layers per GraphLSTM cell as 2, the learning rate $10^{-4}$, the batch size is 32, and the sampling number for VAE as 20 at the inference stage. If the choice of graph neural layer in GraphLSTM is GAT, the number of heads is set to 8, and the dropout rate is set to 0.4. The tolerance of delay is set to 5 min. We calculate anomaly scores from a component perspective [(10)] for both data sets. During experiments, the input for traditional anomaly detectors is each individual observation, while the input for deep leaning-based anomaly detectors in each sliding window applied over the observations. In particular, for a simple auto-encoder which is not customized for time series data, we flatten the time series as its input, just like [3] does.

Detecting anomalies in cloud systems should work online. Moreover, the performance data are collected continuously. Therefore, we take the former two days of data for training with the rest one day for testing. Experiments on each data set using each anomaly detector are repeated five times, and the average results are recorded for comparison.

## D. Overall Performance

In this section, we will answer the question ***"Can TopoMAD outperform other approaches in anomaly detection for topological multivariate time series collected from a cloud system?"*** Figs. 9–12 show comparisons (including AP and mAP) between our models in TopoMAD with other baseline anomaly detectors trained in two ways mentioned earlier. It can be seen that the model in TopoMAD outperforms or matches all baseline anomaly detectors. In the following, we will discuss some baseline anomaly detectors individually in detail.

In our experiments, the performance of traditional anomaly detectors varies a lot from each other. This is mainly because the performance of a traditional anomaly detector highly depends on the corresponding assumption of data distribution. When the corresponding assumption of a traditional anomaly detector is violated, its performance will seriously downgrade. For instance, LOF behaves well on the data set MMS while it shows a poor performance on the data set MBD. This is because we routinely scale the load generator when collecting the data of MMS, and the workload is relatively stable under the same level of workload. As a result, normal observations of MMS tend to locate in a dense neighborhood area. As for the data set MBD, there are multiple workloads with random parameters running in the cluster, leading to more complex workload features. Therefore, when used in data set MBD, LOF does not behave and in the data set MMS.

The overall results of the evaluated algorithms seem not very high in our experiments. Two reasons account for this phenomenon. First, we inject faults with different kinds and severities to the system. Some faults result in a very subtle impact on our system. In Fig. 13, we show the performance of TopoMAD using MMS with more severe faults (e.g., higher network delays). When TopoMAD is evaluated in data with severer faults, the result will be better. Second, there exist some fluctuations which are not caused by the injected faults in the collected metrics. These fluctuations do not last for a long time and might be induced by dynamic changes in the running environment. They are common in reality. However, these fluctuations are usually detected as anomalies with an unsupervised anomaly detector because they indeed occur less frequently and deviate far from normal data. In practice, if operators treat this kind of fluctuations as false alerts, they can set some rules to filter the alerts (e.g., only a corresponding alert lasts long enough will it be sent to the operators) or train a supervised anomaly detector to discriminate these fluctuations from real anomalies. In the following, we shall focus on the comparison among different anomaly detectors.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

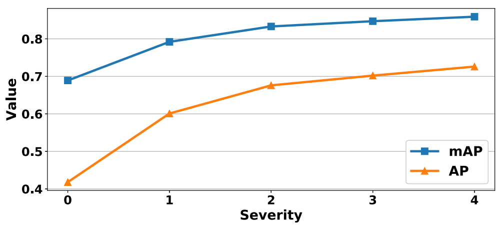

Fig. 13. Performance of TopoMAD on MMS under different levels of severities. A higher level of severity results in anomalies with larger deviations.

Deep learning-based models are good at modeling complex dependencies embedded in data, but this advantage can turn into a disadvantage when these deep anomaly detectors are trained on contaminated training data. With this problem out of consideration when performing unsupervised anomaly detection, attempts to improve model fitting capacity are very likely to backfire. For MBD, LSTM-ED performs the best among all baseline anomaly detectors. For MMS, if we utilize the concatenate policy, RNN-EBM achieves the best result among all baseline deep anomaly detectors, and if we apply the one-by-one policy, a simple auto-encoder is the best in all baseline deep anomaly detectors.

LSTM-AD achieves the worst result in all deep learning-based models evaluated. As a prediction-based model, LSTM-AD heavily relies on the fact that the time series should be predictable; therefore, it is not surprising that LSTM-AD performs the worst when our data sets do not satisfy this setting. By contrast, the assumption that observations are reconstructable is weaker and more reliable. Other deep learning-based models we evaluate, including auto-encoder, LSTM-ED, RNN-EBM, and TopoMAD are based upon this more reliable assumption, and thus, perform better than LSTM-AD in our two data sets.

LSTM-ED is a reconstruction-based model, and our model in TopoMAD can be seen as its extension with consideration of topological information and contaminated training data. In addition to the lack of these considerations, the suboptimal performance of LSTM-ED is also caused by the discrepancy between input distribution at training and testing phases because the decoder in LSTM-ED uses ground truth observations as input at the training stage while reconstructed observations at the inference stage. To show the comparison between LSTM-ED and TopoMAD more clearly, we provide a case study in Appendix B of the supplementary material.

As shown in Section V-C, we evaluate the baseline anomaly detectors using two training policies, namely, a concatenate policy and a one-by-one policy. For MBD, nearly all baseline models achieve better performance using the concatenate policy (note that deep anomaly detectors with the concatenate policy have higher space and time complexity than TopoMAD and those with the one-by-one policy and we will discuss this more specifically in Section V-I). Nevertheless, no matter which policy they utilize, TopoMAD remains considerable superiority over them. In Section IV-A, we claim three benefits of TopoMAD, namely, the unified feature learning, the convenience for end-to-end learning and the ability to prevent overfitting. These three benefits are based upon the comparison between our model and other methods trained using the two policies declared earlier.

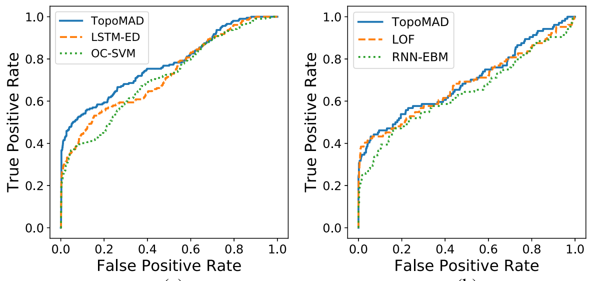

Fig. 14. ROC curves on both data sets, MBD and MMS. (a) MBD. (b) MMS.

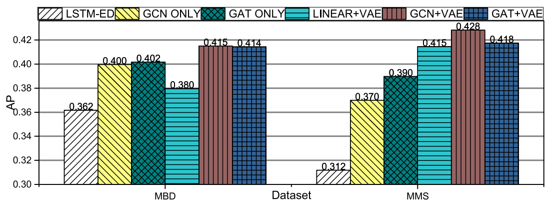

Fig. 15. AP of model variants on two data sets.

Finally, the ROC curves of anomaly detectors with top three performances on each data set are displayed in Fig. 14 to show the performance tradeoff of different models more distinctly. From the ROC curves on MBD, we can see that the false positive rate of TopoMAD is lower compared with the other two baseline models under a relatively low threshold. As for MMS, the performance of TopoMAD is similar to LOF and steadily better than RNN-EBM under different thresholds.

## E. Effects of Major Components

In this section, we will answer the question ***"How does each component of TopoMAD affect the performance?"*** The results are displayed in Fig. 15. We include the mAP of LSTM-ED here because our model is kind of its extension. We select the best performance metric gained by LSTM-ED among two policies mentioned above to display here. "GCN ONLY" and "GAT ONLY" denote model variants of TopoMAD gained by removing its variational component which degrades our model into a deterministic model. "LINEAR + VAE" denotes a model variant gained by replacing the GraphLSTM in TopoMAD with a simple LSTM, which loses sight of the topological information. "GCN + VAE" and "GAT + VAE" denote model variants with two different choices of graph neural layers when implementing GraphLSTM in TopoMAD.

*1) Effect of the Inclusion of Graph Neural Networks:* The objective of including graph neural networks in TopoMAD is to extract features from neighbors of each component. We can see in Fig. 15 that "GCN ONLY" and "GAT ONLY" are superior to LSTM-ED, and "GCN + VAE" and "GAT + VAE" outperform or match "LINEAR + VAE" in both data sets. That is, no matter in a deterministic model or in a stochastic model, utilizing graph neural networks to explicitly model the spatial dependence among components in a cloud system is beneficial.

*2) Effect of VAE:* It can be seen in Fig. 15 that "LINEAR + VAE," "GCN + VAE," and "GAT + VAE" all outperform their corresponding deterministic variants. The superiority of

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE III
DIFFERENT TOPOLOGY INPUTS' AP ON TWO DATA SETS

| Topology Input | MBD | MMS |
|---|---|---|
| random | 0.389-0.408 | 0.406-0.420 |
| origin | 0.414 | 0.418 |

VAE as an anomaly detector over a simple auto-encoder has been demonstrated in [13], which can be concluded as the benefits of representing latent variables and reconstructions as stochastic variables and using probability measures as anomaly scores. In addition, we have also illustrated VAE's superiority when model training is performed on all data (also denoted as contaminated data) in Section IV-A. Therefore, we recommend VAE instead of a deterministic auto-encoder when performing unsupervised anomaly detection.

## F. Impact of the Topology Input

In this section, we will answer the question ***"How effective is the topology?"*** We validate the rationality of our definition of topology input by comparing our method with the performance gained by inputting a random edge set array.

The results in Table III demonstrate that the topology input also plays an important role in the superior performance of TopoMAD. However, a misconfiguration of the topology input does not harm so much. This is because we maintain individual parameters for different nodes outside the GraphLSTM (as described in Section IV-E, multilayer perceptrons outside the GraphLSTM are learned individually for each node), allowing TopoMAD to focus on the learning of each node one by one after processing the topological information. With this design, the output of our model will not be highly dependent on the topology input and yet can gain benefit from it. Thus, our model can also work even when it is confused by a misconfiguration of the topology input. The second reason is that actually components in a system have all kinds of connections with each other and our idea is only to construct a simplification of all these connections. From this point of view, we cannot say an edge we define between the two components is wrong. What we can say is that it does not represent a direct relation. Yet a better way to construct an effective graph-based representation for a cloud system deserves further research.

## G. Interpretability of TopoMAD

In this section, we will answer the question ***"How can we interpret the results of TopoMAD?"*** We will take advantage of the visualization of hidden representations calculated by TopoMAD to explain why it tends to give an anomalous observation a higher anomaly score. In Fig. 16, we show the 3-D hidden representations of each node calculated by TopoMAD with the anomalous observations in red and the normal observations in blue.

From Fig. 16, we can find that there is an overlap between the hidden representations of anomalous observations and normal observations. The hidden representations are used as the input of the decoder to reconstruct the observations. Because normal data occur more frequently, TopoMAD can capture the normal pattern in the training data and reconstruct normal data very well. Meanwhile, because most anomalous

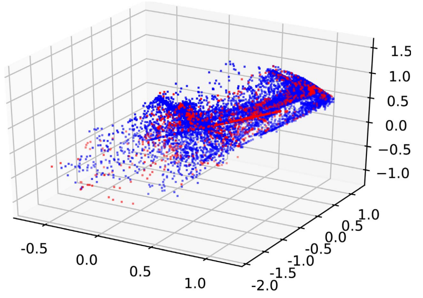

Fig. 16. Three-dimensional hidden representations of each node calculated by TopoMAD for MBD.

TABLE IV
AP OF TOPOMAD UNDER DIFFERENT ANOMALY RATIOS

| Anomaly Ratio | MBD | MMS |
|---|---|---|
| 10% | 0.422 | 0.434 |
| 15% | 0.389 | 0.432 |
| 20% | 0.366 | 0.403 |

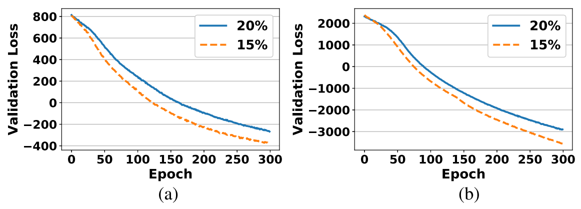

Fig. 17. Convergence of TopoMAD on both data sets. (a) MBD. (b) MMS.

data share similar hidden representations with normal data, the decoder cannot learn to reconstruct them well, resulting in their higher anomaly scores. Su *et al.* [12] also draw a similar conclusion about how VAE-based anomaly detectors work.

## H. Robustness of TopoMAD

In this section, we will answer the question ***"How is the robustness of TopoMAD?"***

First, we evaluate our model under different degrees of data contamination to figure out the relationship between the performance of TopoMAD and the data anomaly ratio. The training data sets with different anomaly ratios are generated through sampling normal data in the original data set until it reaches a corresponding anomaly ratio. Table IV displays the performance of TopoMAD under the anomaly ratios of 10%, 15%, and 20%.

From Table IV, we can see that the performance of TopoMAD will not degrade too much with the increase of the anomaly ratio. This is because the hidden representation calculated by VAE for an observation is a stochastic variable. Even when the anomaly ratio is relatively high, it is still far lower than the normal data ratio. As a result, it is difficult for the decoder to learn to reconstruct the anomaly well when in each epoch, and the calculated latent is sampled from a normal distribution and can be quite different from the latent calculated for this observation in previous epoches.

In addition, we present the convergence of TopoMAD under different anomaly ratios in Fig. 17. It can be seen that TopoMAD can still converge stably at high anomaly proportions.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE V
TIME COMPLEXITY OF DEEP LEARNING RECONSTRUCTION-BASED ANOMALY DETECTION METHODS

| Methods | Time Complexity | |
|---|---|---|
| | concat | one by one |
| FC | $O(\|V\|^2WFF')$ | $O(\|V\|WFF')$ |
| RNN | $O(\|V\|^2WFF')$ | $O(\|V\|WFF')$ |
| TopoMAD | $O(\|V\|WFF' + \|E\|WFF')$ | |

TABLE VI
SPACE COMPLEXITY OF DEEP LEARNING RECONSTRUCTION-BASED ANOMALY DETECTION METHODS

| Methods | Space Complexity | |
|---|---|---|
| | concat | one by one |
| FC | $O(\|V\|^2WFF')$ | $O(\|V\|WFF')$ |
| RNN | $O(\|V\|^2FF')$ | $O(\|V\|FF')$ |
| TopoMAD | $O(\|V\|FF' + \|E\|)$ | |

Last but not least, we discuss the robustness of TopoMAD with regard to initializations. Under the condition of multiple random initializations, the performance of TopoMAD varies. The worst AP TopoMAD achieves on MBD is 0.381, and that on MMS is 0.399. The worst performance of TopoMAD can still remain superiority over all baseline models in MBD and baseline models except for LOF in MMS.

## I. Efficiency of TopoMAD

In this section, we will answer the question ***"How is the efficiency of TopoMAD?"***

Usually, there are a huge amount of data we can collect from a large-scale cloud system, and the computational cost of an anomaly detector will inevitably increase with the enlargement of the cloud system. In the following, we will first give a theoretical analysis about the time complexity and the space complexity of TopoMAD and other deep learning reconstruction-based anomaly detection methods. In Tables V and VI, we present the complexity of a single layer in fully connected feed-forward neural networks-based models (FC), RNNs-based models, and TopoMAD when dealing with data with the window size $W$, the node number $|V|$, the edge number $|E|$, the number of features per node $F$, and the hidden dim per node $F'$. Note that $|E| \ll |V|^2$ usually holds true in a cloud system topology. From Tables V and VI, we can see that the time complexity and the space complexity of TopoMAD are only higher than those of the RNNs-based models trained in a one-by-one way. Besides, there exists some work [78] which can help improve efficiency when training large graph neural networks.

In addition to a theoretical analysis, we also conduct experiments to evaluate the execution time of TopoMAD under topologies of varying complexities. We generate synthetic data sets using the data set MMS by doubling the number of pods from each service. The metrics of newly generated pods are copied from the original data set. The window size and the hidden dim are fixed to 1 during the experiments. Table VII shows the execution time of TopoMAD for testing the entire testing data set. The growth rate of the execution time is acceptable for anomaly detection in cloud systems.

TABLE VII
EXECUTION TIME OF TOPOMAD WITH REGARD TO TOPOLOGIES OF DIFFERENT SCALES

| Nodes | 100 | 200 | 400 | 800 | 1600 | 3200 |
|---|---|---|---|---|---|---|
| Time(s) | 1.19 | 2.67 | 5.40 | 11.26 | 18.35 | 40.97 |
| Std(s) | 0.01 | 0.39 | 0.42 | 1.79 | 1.54 | 8.15 |

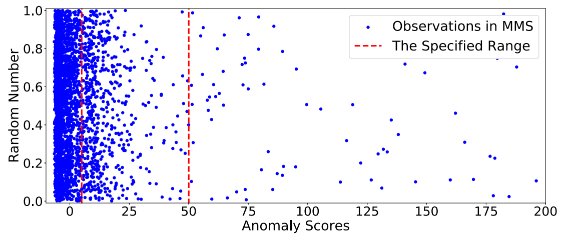

Fig. 18. Distribution of anomaly scores on MMS.

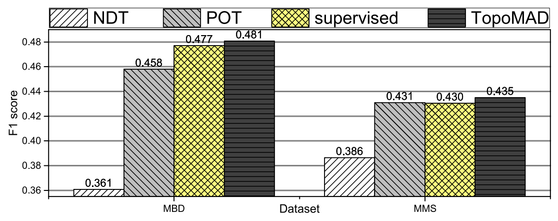

Fig. 19. F1 scores of different threshold selection methods on both data sets.

## J. Effect of Our Threshold Selection Method

In this section, we will answer the question ***"How effective is our unsupervised threshold selection method in TopoMAD? Can it recommend a relatively better threshold?"*** We compare our unsupervised threshold selection method with two unsupervised threshold selection methods POT [12] and NDT [47] and a supervised one which enumerates thresholds to achieve the best F1 score in the training data set with the help of labels. The objective of our method is to recommend a relatively good threshold right after the training stage, and therefore, we assume that the testing data set remains unseen until a threshold is selected. For the supervised method here, a threshold which can achieve the best F1 score in the training data set is selected and applied to the testing data set. For POT, we evaluate hyperparameters used in [12] and display the best result achieved. For NDT, we select the historical window size from 15 min, 30 min, 1 h, and 2 h according to the performance they achieve.

Operators can narrow the search range of thresholds by observing the distribution of anomaly scores on the training data set. To give an example, we show anomaly scores of the training data of MMS in Fig. 18, where the $x$-axis denotes anomaly scores of observations, and the $y$-axis denotes random numbers which help flatten out the displayed anomaly scores so that we can recognize their density. The corresponding threshold can be selected from a roughly narrowed range where the density of anomaly scores begins to decrease (such as 5$\sim$50 here).

The F1 scores obtained are displayed in Fig 19. We can see that the performance of our method is even slightly higher than the supervised one (the best threshold on the training data set does not always guarantee to be the best on the testing data set). This demonstrates that an unsupervised threshold

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

selection method can also gain a relatively good threshold. Considering the lack of operational data with labels, in reality, we recommend selecting a threshold in an unsupervised way.

### *K. Discussions*

*1) Assumptions of Unsupervised Anomaly Detectors:* Due to the lack of labeled data for training, the effectiveness of an unsupervised anomaly detector relies heavily on its assumption. As introduced in Section II-A, assumptions of traditional anomaly detectors include stochastic model-based assumptions, density-based assumptions, classification-based assumptions, and so on. As for deep learning-based anomaly detectors examined in this article, putting aside their superiority in modeling dependence in data with complex structure, their assumptions are actually quite simple. What they assume is that normal data are easier to generate than abnormal data. This generation process can be further subdivided into reconstruction based or prediction based. The quality of an assumption is determined by the data rather than the assumption itself; therefore, it is no wonder a simple traditional model can achieve same or even superior performance compared with some state-of-the-art deep learning-based models, as visualized in Figs. 9 and 10. Integration of the capacity of modeling dependence in data with complex structure in deep learning-based models and a diversity of assumptions in traditional models is worth of a try, which is denoted as a deep hybrid model in [79].

*2) Anomaly Localization:* Once an anomaly is detected, system operators usually need to locate the real root causes of this anomaly. To locate an anomaly, operators can take the anomaly scores of all components generated by TopoMAD as the input of some anomaly localization algorithms, such as CauseInfer [27] and microscope [80]. For example, when an anomaly in a microservice system is detected by TopoMAD, operators can locate the anomaly using the cause inference algorithm proposed in microscope [80], which traverses across the service topology to find the root cause.

## VI. Conclusion

Since anomaly detection is essential to operate a cloud system, this article proposes a topology-aware multivariate time series anomaly detector, namely, TopoMAD for unsupervised anomaly detection. To achieve that, we propose a novel architecture of a neural network by integrating the technologies of graph neural networks, LSTM and VAE. Our approach can robustly and effectively model the complex spatiotemporal dependence in contaminated data. The time complexity of TopoMAD is quantitatively analyzed, which shows that TopoMAD is efficient to analyze a large-scale cloud system. Moreover, a threshold selection algorithm is also proposed to help optimize TopoMAD. The experimental results conducted on two real-world data sets demonstrate our model's superior performance over other baseline anomaly detectors.

In future works, we will focus on the following.

1) Research in online learning techniques for TopoMAD.
2) The choice of metrics which are better at exposing the failure of a cloud system.
3) Research in combining some other state-of-the-art deep anomaly detectors to gain a better performance.

## References

[1] Y. Dang, Q. Lin, and P. Huang, "AIOps: Real-world challenges and research innovations," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2019, pp. 4–5.

[2] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1067–1075.

[3] H. Xu *et al.*, "Unsupervised anomaly detection via variational autoencoder for seasonal KPIs in Web applications," in *Proc. World Wide Web Conf. World Wide Web WWW*, 2018, pp. 187–196.

[4] Z. Li, W. Chen, and D. Pei, "Robust and unsupervised KPI anomaly detection based on conditional variational autoencoder," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2018, pp. 1–9.

[5] W. Chen *et al.*, "Unsupervised anomaly detection for intricate KPIs via adversarial training of VAE," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1891–1899.

[6] H. Ren *et al.*, "Time-series anomaly detection service at microsoft," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 3009–3017.

[7] Z. Zheng, J. Zhu, and M. R. Lyu, "Service-generated big data and big data-as-a-service: An overview," in *Proc. IEEE Int. Congr. Big Data*, Jun. 2013, pp. 403–410.

[8] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, 2015, pp. 89–94.

[9] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," in *Proc. Anomaly Detection Workshop 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1–5.

[10] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 1100–1109.

[11] C. Zhang *et al.*, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 1409–1416.

[12] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2828–2837.

[13] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lect. IE*, vol. 2, no. 1, pp. 1–18, Dec. 2015.

[14] L. Li, J. Yan, H. Wang, and Y. Jin, "Anomaly detection of time series with smoothness-inducing sequential variational auto-encoder," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 13, 2020, doi: 10.1109/TNNLS.2020.2980749.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–14.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–13.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000.

[20] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[21] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 582–588.

[22] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, vol. 29, 2000, pp. 93–104.

[23] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier detection using replicator neural networks," in *Proc. Int. Conf. Data Warehousing Knowl. Discovery*. Berlin, Germany: Springer, 2002, pp. 170–180.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[24] S. Fu, J. Liu, and H. Pannu, "A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines," in *Proc. Int. Conf. Adv. Data Mining Appl.* Berlin, Germany: Springer, 2012, pp. 726–738.

[25] T. Huang *et al.*, "An LOF-based adaptive anomaly detection scheme for cloud computing," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2013, pp. 206–211.

[26] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. 1st ACM Symp. Cloud Comput. SoCC*, 2010, pp. 193–204.

[27] P. Chen, Y. Qi, P. Zheng, and D. Hou, "CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 1887–1895.

[28] Q. Lin *et al.*, "Predicting node failure in cloud service systems," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. ESEC/FSE*, 2018, pp. 480–490.

[29] X. Zhou *et al.*, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Softw. Eng.*, early access, Dec. 18, 2018, doi: 10.1109/TSE.2018.2887384.

[30] X. Zhou *et al.*, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 683–694.

[31] H. Liu, S. Lu, M. Musuvathi, and S. Nath, "What bugs cause production cloud incidents," in *Proc. Workshop Hot Topics Operating Syst.*, 2019, pp. 155–162.

[32] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A novel technique for long-term anomaly detection in the cloud," in *Proc. 6th USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, 2014, p. 15.

[33] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online detection of utility cloud anomalies using metric distributions," in *Proc. IEEE Netw. Oper. Manage. Symp. NOMS*, Apr. 2010, pp. 96–103.

[34] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[35] W. Meng *et al.*, "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 4739–4745, doi: 10.24963/ijcai.2019/658.

[36] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. ESEC/FSE*, 2019, pp. 807–817.

[37] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 207–218.

[38] N. Khadke, M. P. Kasick, S. P. Kavulya, J. Tan, and P. Narasimhan, "Transparent system call based performance debugging for cloud computing," Presented at the Workshop Manag. Syst. Automatically Dynamically, 2012.

[39] W. Sha, Y. Zhu, M. Chen, and T. Huang, "Statistical learning for anomaly detection in cloud server systems: A multi-order Markov chain framework," *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 401–413, Jun. 2018.

[40] Y. Gan *et al.*, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 19–33.

[41] M. Ma, W. Lin, D. Pan, and P. Wang, "MS-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2019, pp. 60–67.

[42] A. B. Sharma, H. Chen, M. Ding, K. Yoshihira, and G. Jiang, "Fault detection and localization in distributed systems using invariant relationships," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2013, pp. 1–8.

[43] P. Chen, Y. Qi, X. Li, and L. Su, "An ensemble MIC-based approach for performance diagnosis in big data platform," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2013, pp. 78–85.

[44] W. Cheng, K. Zhang, H. Chen, G. Jiang, Z. Chen, and W. Wang, "Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations," in *Proc. 22Nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 805–814.

[45] Z. Zheng, T. Chao Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Trans. Services Comput.*, vol. 5, no. 4, pp. 540–550, 4th Quart., 2012.

[46] M. Zasadziński, M. Solé, A. Brandon, V. Muntés-Mulero, and D. Carrera, "'Next stop' noops': Enabling cross-system diagnostics through graph-based composition of logs and metrics," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*. Sep. 2018, pp. 212–222.

[47] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 387–395.

[48] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," 2019, *arXiv:1901.00596*. [Online]. Available: http://arxiv.org/abs/1901.00596

[49] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–14.

[50] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: http://arxiv.org/abs/1412.3555

[51] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 3634–3640.

[52] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 7444–7452.

[53] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *Proc. OSDI*, vol. 4, 2004, p. 18.

[54] R. Fonseca, G. Porter, R. H. Katz, and S. Shenker, "X-trace: A pervasive network tracing framework," in *Proc. 4th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2007, pp. 271–284.

[55] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating network application dependency discovery: Experiences, limitations, and new solutions," in *Proc. OSDI*, vol. 8, 2008, pp. 117–130.

[56] P. Barham *et al.*, "Constellation: Automated discovery of service and host dependencies in networked systems," Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2008-67, 2008, pp. 1–14.

[57] J. Hwang, G. Liu, S. Zeng, F. Y. Wu, and T. Wood, "Topology discovery and service classification for distributed-aware clouds," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2014, pp. 385–390.

[58] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.

[59] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[60] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with graph lstm," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2016, pp. 125–143.

[61] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-T. Yih, "Cross-sentence N-ary relation extraction with graph LSTMs," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 101–115, Dec. 2017.

[62] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5308–5317.

[63] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[64] C. Robert and G. Casella, *Monte Carlo Statistical Methods*. Berlin, Germany: Springer, 2013.

[65] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[66] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1171–1179.

[67] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Proc. IEEE 26th Int. Conf. Data Eng. Workshops (ICDEW )*, Mar. 2010, pp. 41–51.

[68] (2020). *HDFS-448*. Accessed: 2020. [Online]. Available: https://issues.apache.org/jira/browse/HDFS-448

[69] (2020). *HDFS-8160*. Accessed: 2020. [Online]. Available: https://issues.apache.org/jira/browse/HDFS-8160

[70] (2020). *Telegraf*. Accessed: 2020. [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/

[71] (2020). *Cloudwatch*. Accessed: 2020. [Online]. Available: https://aws.amazon.com/cn/cloudwatch/

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[72] (2019). *Hipster Shop: Cloud-Native Microservices Demo Application*. Accessed: 2019. [Online]. Available: https://github.com/GoogleCloudPlatform/microservices-demo

[73] (2019). *Kubernetes*. Accessed: 2019. [Online]. Available: https://kubernetes.io/

[74] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proc. 23rd Int. Conf. Mach. Learn. ICML*, 2006, pp. 233–240, doi: 10.1145/1143844.1143874.

[75] M. Fey and J. Eric Lenssen, "Fast graph representation learning with PyTorch geometric," 2019, *arXiv:1903.02428*. [Online]. Available: http://arxiv.org/abs/1903.02428

[76] P. I. Frazier, "A tutorial on Bayesian optimization," 2018, *arXiv:1807.02811*. [Online]. Available: http://arxiv.org/abs/1807.02811

[77] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.

[78] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 257–266.

[79] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019, *arXiv:1901.03407*. [Online]. Available: http://arxiv.org/abs/1901.03407

[80] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *Proc. Int. Conf. Service-Oriented Comput.* Berlin, Germany: Springer, 2018, pp. 3–20.

**Zilong He** received the B.E. degree from Sun Yat-sen University, Guangzhou, China, in 2019.

He is currently a Post-Graduate Researcher with the School of Data and Computer Science, Sun Yat-sen University. His current research areas include anomaly detection algorithms and AI-driven operations.

**Pengfei Chen** received the Ph.D. degree from the Department of Computer Science, Xi'an Jiaotong University, Xi'an, China, in 2016.

He visited the IBM Thomas J. Watson Research Center, Ossining, NY, USA, in 2017. He is currently an Associate Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, where he is also a Ph.D. Advisor. He has strong skill in cloud computing. He current research interests include distributed systems, artificial intelligence for IT operations, cloud computing, microservice, and blockchain.

**Xiaoyun Li** is currently pursuing the Ph.D. degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

Her current research areas include log analysis and AI-driven operations.

**Yongfeng Wang** received the B.E. degree from the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

His current research areas include persistent memory, storage system, and cloud computing.

**Guangba Yu** is currently pursuing the Ph.D. degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

His current research areas include distributed system, cloud computing, and AI-driven operations.

**Cailin Chen** is currently a Post-Graduate Researcher with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. Her current research area includes AI-driven operations.

**Xinrui Li** received the B.E. degree from the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

**Zibin Zheng** (Senior Member, IEEE) received the Ph.D. degree from The Chinese University of Hong Kong, Hong Kong, in 2011.

He is currently a Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include blockchain, services computing, software engineering, and financial big data.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on April 25,2021 at 00:35:04 UTC from IEEE Xplore. Restrictions apply.