Efficient and Robust KPI Outlier Detection for Large-Scale Datacenters

Yongqian Sun, *Member, IEEE*, Daguo Cheng, Tiankai Yang, Yuhe Ji, Shenglin Zhang, *Member, IEEE*, Man Zhu, Xiao Xiong, Qiliang Fan, Minghan Liang, Dan Pei, *Senior Member, IEEE*, Tianchi Ma, and Yu Chen

***Abstract*—To ensure the performance of large-scale datacenters, operators need to monitor up to tens of millions of various-type KPIs, e.g., CPU utilization, memory utilization. For each KPI, it is crucial but challenging to detect outliers that deviate from its historical patterns or the patterns of other KPIs in the same period. In this work, we propose *OutSpot*, an unsupervised outlier detection framework that integrates hierarchical agglomerative clustering (HAC) with conditional variational autoencoder (CVAE), which significantly improves computational efficiency and comprehensively learns the above two patterns. Additionally, two simple yet effective techniques, soft threshold and median filter, are applied to precisely determine outlier KPIs. Using two real-world datasets collected from the datacenters owned by a top-tier global short video service provider and a top-tier domestic operator,respectively. It demonstrates that *OutSpot* achieves the best F1 score of 0.95 and 0.91, AUC of 0.99 and 0.99 on the two datasets, significantly outperforming seven baseline outlier detection methods.**

***Index Terms*—Outlier detection, KPI, deep generative model, AIOps.**

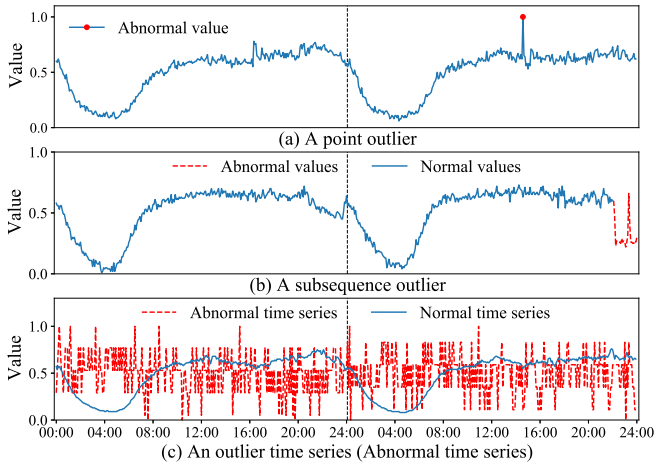


Fig. 1. Outlier type in time series (KPIs). (two-day-long data).

I. Introduction

TODAY's large-scale datacenters house tens of thousands to millions of servers, hundreds of thousands of switches and routers, and millions of cables and fibers [1], and the performance of these devices is vitally important to the services provided by datacenters. The unexpected behaviors of these devices, which may be caused by hardware failures (e.g., uncorrectable hardware errors, machine aging, power down), software bugs (e.g., thread crash, memory leaking), or cyber-attacks, will degrade the quality of service (QoS) and may even lead to a drop in revenue [2], [3]. Therefore, operators carefully monitor the status of these devices through various types of key performance indicators (KPIs), e.g., CPU utilization, memory utilization, TCP retransmission percentage, disk I/O rate [4], [5], [6], [7], [8], [9], [10]. Usually, the monitoring data of a KPI, which is collected with a fixed time interval (e.g., 5 minutes), forms a univariate time series (from now on, we use "KPI" and "KPI time series" interchangeably). Considering the significant number of devices in a large-scale datacenter, operators have to continuously monitor tens of millions of KPIs.

An outlier of a KPI usually denotes an unexpected behavior of a device, and it can be classified into three types: a point outlier, a subsequence outlier, or an outlier time series [11], as shown in Fig. 1. A point outlier is a data point that behaves abnormally compared to the historical data points in the KPI or its adjacent points. A subsequence outlier denotes consecutive data points (i.e., a time segment) in a KPI whose collective behavior is abnormal compared to this KPI's historical normal pattern. However, each data point of it individually is not necessarily a point outlier. Moreover, an outlier time series denotes that the entire time series is an outlier compared to other KPIs in the same period. Due to the auto-recovery and load balancing mechanisms of datacenters, usually, a point outlier does not imply a device

Manuscript received 8 July 2022; revised 6 April 2023; accepted 23 April 2023. Date of publication 1 May 2023; date of current version 6 September 2023. This work was supported in part by the National Natural Science Foundation of China under Grants 62072264, 62272249, and in part by the Natural Science Foundation of Tianjin under Grant 21JCQNJC00180. Recommended for acceptance by C. Li. *(Yongqian Sun and Daguo Cheng contributed equally to this work.) (Corresponding author: Shenglin Zhang.)*

Yongqian Sun, Yuhe Ji, Man Zhu, Xiao Xiong, Qiliang Fan, and Minghan Liang are with the College of Software, Nankai University, Tianjin 300071, China (e-mail: sunyongqian@nankai.edu.cn; jiyuhemail@foxmail.com; zhuman2019@mail.nankai.edu.cn; xiongxiao@mail.nankai.edu.cn; fanqiliang@mail.nankai.edu.cn; liangminghan@mail.nankai.edu.cn).

Shenglin Zhang is with the College of Software, Haihe Laboratory of Information Technology Application Innovation (HL-IT), Nankai University, Tianjin 300071, China (e-mail: zhangsl@nankai.edu.cn).

Daguo Cheng is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100190, China (e-mail: cdg22@mails.tsinghua.edu.cn).

Dan Pei is with the Department of Computer Science, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100190, China (e-mail: peidan@tsinghua.edu.cn).

Tiankai Yang is with the Viterbi School of Engineering, University of Southern California, Los Angeles, CA 90007 USA (e-mail: tiankaiy@usc.edu).

Tianchi Ma and Yu Chen are with the Kuaishou Technology, Beijing 100085, China (e-mail: matianchi@kuaishou.com; chenyu11@kuaishou.com).

Digital Object Identifier 10.1109/TC.2023.3272288

0018-9340 © 2023 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: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

failure, and operators often ignore it. However, a subsequence outlier or an outlier time series denotes that a device suffers from abnormal behaviors or becomes poorly managed for an extended period. Operators should take measures to mitigate it.

Over the years, a large number of point outlier/anomaly detection methods have been proposed in the literature [8], [9], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25]. These methods cannot be easily improved to detect subsequence outlier or outlier time series because: 1) they focus on the abnormal behavior of each data point instead of a time segment, and 2) they do not compare a KPI with other KPIs in the same period. Although a few works have been proposed for subsequence outlier detection through *learning the historical pattern for each KPI* [26], [27], or outlier time series detection by *learning the pattern of all KPIs in the same period* [28], none of these methods can detect the above two outliers simultaneously. Considering the massive number of KPIs, these subsequence outlier detection methods are computationally intensive because they have to train a separate model for each KPI. Additionally, outlier time series detection methods can miss subsequence outliers, which behave normally compared to other KPIs in the same period but abnormally compared with the KPI's historical pattern. Therefore, we aim to design a framework to *efficiently and accurately* detect both subsequence outlier and outlier time series of KPIs.

Due to the large number of KPIs in large-scale datacenters, we cannot deploy supervised methods requiring a considerable amount of labeling effort to detect KPI outliers (hereinafter, a KPI outlier refers to a subsequence outlier or an outlier time series). Consequently, we apply unsupervised methods, which automatically learn KPI's normal patterns and need no labels, for KPI outlier detection. Recently, deep generative models have shown its superior performance for KPI outlier detection in an unsupervised manner [4], [5], [12], [14], [15], [19], [20], [21], [23]. We thus also apply deep generative models to comprehensively learn KPI's normal patterns and accurately detect KPI outliers. However, applying deep generative models for KPI outlier detection faces the following four challenges.

1) A Considerable Number of KPIs With Various Types. In large-scale datacenters, operators should detect outliers for tens of millions of KPIs. However, a deep generative model usually consumes high computational resources in the training stage [6], [7]. Therefore, training a separate outlier detection model for each KPI is almost infeasible. However, the various types of KPIs can have very different patterns, and one outlier detection model ignoring these differences is not expressive enough by nature. Therefore, training one model for all KPIs would bound the accuracy.

2) Detect Both Subsequence Outliers and Outlier Time Series. To detect both subsequence outlier and outlier time series, a deep generative model has to learn both the historical pattern of each KPI and the pattern of all KPIs in the same period. It introduces a significant challenge to the deep generative model because it usually learns one type of pattern at a time [29].

3) Determine Outliers. Existing point outlier/anomaly detection works usually apply the reconstruction probability (density estimate) of deep generative models to determine whether a data point is an outlier/anomaly [4], [5], [12], [14], [15], [19], [20], [21], [23]. The lower the reconstruction probability of a data point, the more likely it is to be an outlier. However, the reconstruction probability-based methods can frequently assign a higher reconstruction probability to outliers and thus suffer from low accuracy in outlier determination for high-dimensional data [15]. Therefore, they are inappropriate for subsequence outliers and outlier time series, both of which are high-dimensional data containing multiple data points.

4) Lack of Labels. Although there is no need to obtain labels to train a model for unsupervised methods, we still have to label a collection of KPIs to verify or improve outlier detection methods' performance. To determine an outlier KPI, operators have to check the historical pattern of each KPI and the pattern of all KPIs in the same period. It is labor-intensive and time-consuming if no labeling tool is applied.

To tackle the above challenges, we propose an efficient and robust unsupervised outlier detection framework, *OutSpot*. It can detect both subsequence outlier and outlier time series for large-scale datacenters. Specifically, it applies the hierarchical agglomerative clustering (HAC) method to cluster KPIs based on their patterns. For learning both the historical pattern of each KPI and the pattern of all KPIs in the same period, it then encodes the clustering information into the generative model using the conditional variational autoencoder (CVAE) method. Finally, it compares the reconstructed and original KPI shapes to determine whether a KPI is an outlier.

The main contributions can be summarized as follows:

1) To address the first and second challenges, we propose integrating HAC with CVAE. Specifically, we first apply HAC to cluster KPIs according to their patterns and then embed the cluster information of each KPI into the CVAE method. Since the number of clusters is much smaller than that of KPIs (three in our scenario), and the KPIs of each cluster resemble in pattern, *OutSpot* can detect outliers for the large-scale KPIs with diverse patterns. Additionally, the cluster-information-encoded CVAE method has learned both the historical pattern of each KPI and the pattern of all KPIs in the same period, facilitating the detection of both subsequence outliers and outlier time series.

2) To tackle the third challenge, we determine whether a KPI is an outlier by comparing the original and reconstructed KPI shapes instead of calculating the reconstruction probability. Additionally, we propose to combine soft threshold (ST) and median filter (MF) for improving *OutSpot*'s accuracy.

3) To demonstrate *OutSpot*'s performance, we conduct extensive experiments on the dataset $\mathcal{A}$ from a top global short video service provider and $\mathcal{B}$ from one of the three major domestic communication operators of China. *OutSpot* achieves the best F1 score of 0.95 and 0.91, AUC of 0.99 and 0.99 on the two datasets, significantly outperforming seven baseline outlier detection methods.

4) For addressing the fourth challenge, we release a labeling tool for KPI outlier detection.[1] Additionally, to get readers better understand our work, we also make our labeled dataset $\mathcal{A}$[2]

[1]https://github.com/OutlierDetection-OutSpot/Label-tool
[2]https://github.com/OutlierDetection-OutSpot/Dataset

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

TABLE I
DETAILED INFORMATION ABOUT THE 18 KPIS

KPI categories (count of KPIs)	KPIs
CPU related (7)	*cpu_idle, cpu_sintr, cpu_system, cpu_wio, cpu_user, cpu_ctxt,cpu_nice*
Memory related (1)	*memory_utilization*
VM related (2)	*vm_pgfault, vm_pgmajfault*
TCP related (4)	*tcp_retrans_percentage, tcp_insegs, tcp_outsegs, tcpext_listendrops*
Network related (4)	*rx_bytes_eth0, rx_pkts_eth0, tx_bytes_eth0, tx_pkts_eth0*

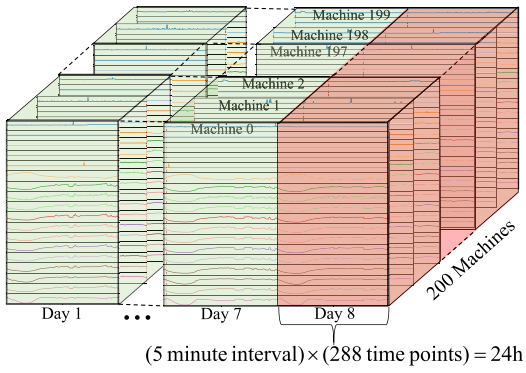


Fig. 2. Examples of KPIs in our scenario. The red area refers to the KPIs to be detected and the green area refers to the historical KPIs. "..." indicates that the similar information is hidden due to space limit.

and code[3] publicly available. Dataset $\mathcal{B}$ is not public due to commercial issues.

II. PRELIMINARIES

A. Problem Statement

Servers are the most prevalent devices in datacenters, and the KPIs of other types of devices, including routers, switches, firewalls, etc., resemble those of servers. The outlier detection methods proposed for server KPI can be easily generalized to other types of devices. Therefore, in this paper, we mainly focus on the KPIs of servers in large-scale datacenters, e.g., CPU idle, memory utilization, as listed in Table I. The KPI values are usually collected with a fixed time interval, like one or five minutes. For example, as shown in Fig. 2, in the studied datacenters, operators monitor 18 KPIs for each server, and the monitoring data of these KPIs are collected every five minutes. Thus a one-day-long KPI has 288 data points.

In this paper, we determine that a one-day-long KPI is an outlier KPI because it contains one or more outliers (e.g., subsequence outliers or outlier time series). To find out the outlier KPIs more frequently, we use a sliding window technique to divide the KPI data with window size = one day and step size = 1.5 hours. First of all, setting the window size = one day because the KPI of a machine is usually periodical, and its period is one day. This is because it changes as the user workload of the machine changes, which conforms to the pattern of user behavior typically having a one-day-long period. As shown in Figs. 1 and 7, we can see that the real-world KPI data is periodical with a one-day-long period. In addition, the KPIs of different machines can have different periodic characteristics (i.e., patterns described in the paper). For example, the KPIs of machine 1 and machine 2 are all periodic but have different periodic characteristics (patterns), so *OutSpot* will group them into different clusters and process them separately. Second, setting the step size = 1.5 hours because:

1) Both subsequence outliers and outlier time series are outliers that last a long time, and we need a relatively long time to determine a subsequence outlier or an outlier time series. In our scenario, for example, the operators think that an outlier KPI should have continuous abnormal values of 1.5 hours (18 points for the interval of 5 minutes) at least.
2) Outlier detection differs from anomaly detection [6], [9], [12], [13] in that it does not need to detect outlier KPIs as quickly as possible. In our work, KPI outlier detection aims to find the devices/services suffering from abnormal behaviors or becoming poorly managed for an extended period. Note that collecting KPI data more frequently does not help shorten the latency of detecting outliers. For example, the KPI data collection intervals of the two datasets are 5 minutes and 15 minutes, respectively. However, both datasets have window size = one day and step size = 1.5 hours. This is mainly because we need a relatively long time (1.5 hours) to determine a subsequence outlier or an outlier time series.

B. Basics of HAC and CVAE

HAC first treats each sample as a singleton cluster. It then successively agglomerates closer pair of clusters and returns the clustering result according to the number of clusters to be found or the linkage distance threshold above which clusters will not be merged. Compared with other clustering algorithms (e.g., DBSCAN), HAC is not sensitive to the choice of the distance metric because it combines two nodes based on the rank of distances instead of their absolute distance values. Therefore, we adopt HAC for clustering in *OutSpot*.

Deep generative models, especially variational autoencoder (VAE) based models, have shown their superior performance in point outlier/anomaly detection [4], [5], [12], [14], [15], [19], [20], [21], [23]. A generative model usually learns the low-dimensional representations of complex data through an encoder and generates new samples following it by a decoder. Compared to VAE, CVAE generates new samples of specified categories by condition variable [30], [31]. As shown in Fig. 3, CVAE also consists of an encoder and a decoder.

1) *Encoder*. Map the input $\mathbf{x}$ with category information $\mathbf{y}$ to the latent variable distribution $\mathbf{z}$ (i.e., $\mathbf{z} \sim p_\theta(\mathbf{z}|\mathbf{x}, \mathbf{y})$). $p_\theta(\mathbf{z})$ is the prior distribution of $\mathbf{z}$.
2) *Decoder*. Generate an $\mathbf{x}$ with known category information, i.e., $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z}, \mathbf{y})$.

[3] https://github.com/OutlierDetection-OutSpot/Code

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

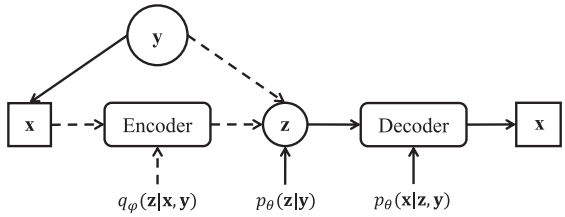


Fig. 3. The architecture of CVAE. The dash lines denote encoding. The solid lines denote decoding.

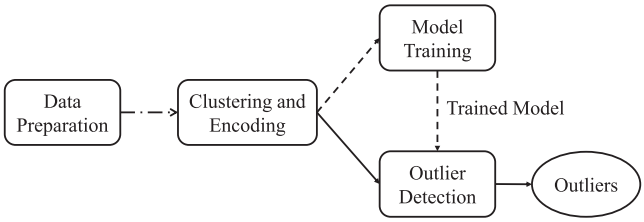


Fig. 4. The overall framework of *OutSpot*. The dash lines denote the training procedure, the solid lines denote the detection procedure, and the dot-and-dash line denotes the modules shared by the two procedures.

Note that the true posterior $p_\theta(\mathbf{z}|\mathbf{x},\mathbf{y})$ is intractable [30]. We alternatively get its approximate distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$ through variational inference. Now we can trace the integral of the marginal log-likelihood

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{y}) &= \int_{\mathbf{z}} q_\phi(\mathbf{x}|\mathbf{z},\mathbf{y}) \log p_\theta(\mathbf{z}|\mathbf{y}) d\mathbf{z} \\ &= \int_{\mathbf{z}} q_\phi(\mathbf{x}|\mathbf{z},\mathbf{y}) \log \frac{p_\theta(\mathbf{x},\mathbf{z}|\mathbf{y}) q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) p_\theta(\mathbf{z}|\mathbf{x},\mathbf{y})} d\mathbf{z} \\ &= \int_{\mathbf{z}} q_\phi(\mathbf{x}|\mathbf{z},\mathbf{y}) \log \frac{p_\theta(\mathbf{x},\mathbf{z}|\mathbf{y})}{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} d\mathbf{z} \\ &\quad + KL[q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \parallel p_\theta(\mathbf{z}|\mathbf{x},\mathbf{y})]. \end{aligned} \tag{1}$$

The first term in (1) is the evidence lower bound (i.e., ELBO), and KL refers to Kullback-Leibler divergence which measures the distance between two probability distributions. Because KL loss is always greater than zero, CVAE is trained through maximizing ELBO

$$\begin{aligned}\mathcal{L}(x,y) &= \log p_\theta(\mathbf{x}|\mathbf{y}) - KL[q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \parallel p_\theta(\mathbf{z}|\mathbf{x},\mathbf{y})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \log \frac{p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y}) p_\theta(\mathbf{z}|\mathbf{y})}{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}. \end{aligned} \tag{2}$$

III. Design

A. Overall Framework

We propose *OutSpot*, integrating HAC with CVAE, to accurately and efficiently detect both subsequence outlier and outlier time series simultaneously for large-scale datacenters. The framework of *OutSpot* is shown in Fig. 4. It consists of two procedures: model training (the dash lines) and detection (the solid lines). The two modules, i.e., Data Preparation, Clustering and Encoding, are shared by the above two procedures (linked by the dot-and-dash line). Data Preparation standardizes all the KPIs at the same magnitude through z-score normalization [5]. In the Clustering and Encoding module, the KPIs are clustered based on their patterns through HAC, and then the cluster information is encoded as the condition variable $\mathbf{y}$ of CVAE. Model Training captures the normal patterns of each cluster using CVAE, and Outlier Detectin determines whether a KPI is an outlier.

The core idea of *OutSpot* is combining HAC with CVAE, and then using the CVAE model to detect two types of outliers: subsequence outlier and outlier time series. For some KPI's one-day-long monitoring data $\mathbf{x}$, we first cluster KPIs into fewer clusters according to the patterns of $\mathbf{x}$ and other KPIs in current period (i.e., the red area of Fig. 2), and then obtain $\mathbf{x}$'s cluster information. After that, we encode this information into CVAE as conditional variables and train the model using all KPIs' historical data (i.e., the green area of Fig. 2) of this cluster. Next we explain how can we use one model to detect the two types of outliers. Since the outlier KPIs usually account for a small proportion, keeping the number of clusters τ_c relatively small can ensure that normal data dominate each cluster, and the outlier KPIs can be clustered into a "normal cluster" instead of separate clusters. Intuitively, to judge whether a KPI $\mathbf{x}$ is a subsequence outlier, we should compare the $\mathbf{x}$'s current pattern with its historical pattern and calculate the difference, and compare $\mathbf{x}$'s current pattern with other KPIs' to verify whether it is an outlier time series. For cluster $\mathbb{C}$, *OutSpot* trains a model using the historical KPIs to learn the historical pattern. Note that this model contains not only the historical patterns of this cluster but also the current patterns of other KPIs (most KPIs are normal, so their historical patterns are similar to current's.). Detecting outlier KPIs through historical patterns is straightforward. However, if the current pattern is consistent with the historical pattern of a KPI that have outliers for multiple days, it is not enough to rely on the historical pattern alone. Actually, when the model also captures the normal patterns of other KPIs in the same period, these outlier KPIs can be easily detected because they are significantly different from the normal patterns of other KPIs in the same period. Therefore, when we obtain the reconstructed $\mathbf{x}'$ by inputting $\mathbf{x}$ into this model, $\mathbf{x}'$ represents the historical pattern of x and the current patterns of other KPIs. Thus, *OutSpot* determines whether $\mathbf{x}$ is a subsequence outlier or an outlier time series just by comparing the difference of $\mathbf{x}$ and $\mathbf{x}'$.

B. Clustering and Encoding

Clustering. As mentioned in Section II-B, we adopt HAC for clustering in this work. Since HAC is insensitive to distance metrics, we choose one of the most popular metrics, euclidean Distance [32]. With a cluster number threshold, τ_c, KPIs are divided into at least τ_c clusters according to Ward linkage [33] ($\tau_c = 3$ in our scenario, and more details can be seen in Section IV-E). A cluster represents the pattern of a type of normal KPI, and it may contain outlier KPIs (more details can be seen in Section V-B).

Encoding. We encode the cluster information as a set of one-hot vectors [21]. One-hot encoding converts the positive

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

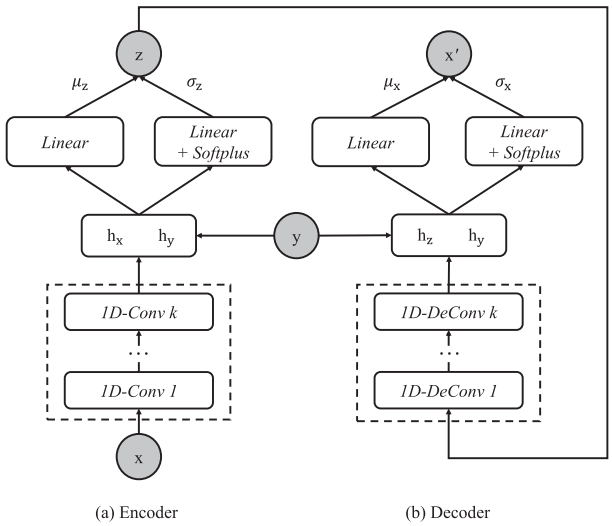


Fig. 5. The network architecture of *OutSpot* consisted of Encoder and Decoder. $\mathbf{x}$ is the input data, $\mathbf{x}'$ is reconstructed output; k is the number of layers of 1D-convolution and 1D-deconvolution, respectively; $\mathbf{z}$ and $\mathbf{y}$ are latent and condition variables, respectively; $\mathrm{h}_{\mathbf{x}}$ is the output of k 1D-convolutions; $\mathrm{h}_{\mathbf{z}}$ is the output of k 1D-deconvolutions; $\mathrm{h}_{\mathbf{y}}$ is the one-hot vectors of condition variables. We concatenate $\mathrm{h}_{\mathbf{x}}$ and $\mathrm{h}_{\mathbf{y}}$, $\mathrm{h}_{\mathbf{z}}$ and $\mathrm{h}_{\mathbf{y}}$ as new tensors in the network, respectively.

integer which represents the cluster information to a binary vector with τ_c dimensions. For example, if $\tau_c = 3$, we will get 3 one-hot vectors: (1, 0, 0), (0, 1, 0), (0, 0, 1). Each of the one-hot vectors corresponds to a cluster. The one-hot vectors of cluster information are treated as the condition variable $\mathbf{y}$ in CVAE, i.e., the condition variable $\mathbf{y}$ represents the category of the input KPI. Please note that the Encoding here is a preprocessing step for the encoder of CVAE.

C. Network Architecture

OutSpot is a reconstruction-based model that consists of an encoder (i.e., $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$) and a decoder(i.e., $p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})$). As shown in Fig. 5(a), in the encoder, *OutSpot* first extracts the shape features of the input KPIs through k 1D-convolution (i.e., one-dimensional convolutional neural network) layers. Then a fully-connected layer maps the shape features (e.g., seasonality, trend and stationarity, etc [34].) of input KPIs $\mathbf{x}$ (i.e., $\mathrm{h}_{\mathbf{x}}$) combined with the cluster information $\mathbf{y}$ (i.e., $\mathrm{h}_{\mathbf{y}}$) to the low-dimensional latent space as a stochastic latent variable $\mathbf{z}$.

For the decoder as shown in Fig. 5(b), *OutSpot* uses 1D-deconvolution layers followed by fully-connected layers. The decoder takes the latent variable $\mathbf{z}$ (i.e., $\mathrm{h}_{\mathbf{z}}$) as input and outputs the reconstructed data, $\mathbf{x}'$, with the cluster information $\mathbf{y}$ (i.e., $\mathrm{h}_{\mathbf{y}}$). Detailedly, the latent variable $\mathbf{z}$ represents the variational distribution in latent space and $\mathbf{x}'$ represents the KPIs' distribution. The probabilities of $\mathbf{z}$ and $\mathbf{x}'$ are assumed to be Gaussian distributions. Therefore, $\mathbf{z}$ and $\mathbf{x}'$ are generated by their mean μ and standard deviation σ instead (i.e., $\mu_{\mathbf{z}}$, $\sigma_{\mathbf{z}}$ and $\mu_{\mathbf{x}}$, $\sigma_{\mathbf{x}}$). Since σ is always greater than zero, we apply *Softplus* as the activation function: $Softplus(\mathbf{x}) = \log(1 + exp(\mathbf{x}))$.

D. Model Training

The model of CVAE in *OutSpot* is trained by maximizing the ELBO (i.e., the loss function) on the marginal log-likelihood

$$\begin{aligned}\mathcal{L}(\mathbf{x},\mathbf{y}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \log \frac{p_\theta(\mathbf{x},\mathbf{z}|\mathbf{y})}{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \log \frac{p_\theta(\mathbf{z}|\mathbf{y})p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})}{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \\ &\quad + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \log p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y}). \end{aligned} \tag{3}$$

The encoder of CVAE defines the approximate posterior distribution as a multivariate Gaussian distribution: $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \sim \mathcal{N}(\mu_{\mathbf{z}}, \sigma_{\mathbf{z}}^2\mathbf{I})$, while the decoder defines the conditional distribution of the KPIs to be detected: $p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y}) \sim \mathcal{N}(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}^2\mathbf{I})$. Besides, as [35] suggests, we can make the latent variable $\mathbf{z}$ independent of condition variable $\mathbf{y}$, i.e., $p_\theta(\mathbf{z}|\mathbf{y}) = p_\theta(\mathbf{z})$.

With further analysis to (3), the first term is the regularization (i.e., Kullback-Leibler loss) on $\mathbf{z}$, which minimizes the difference between the approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$ and the prior distribution $p_\theta(\mathbf{z})$ (i.e., regularize the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x},\mathbf{y})$ by the prior distribution $p_\theta(\mathbf{z})$). The second term is the reconstruction probability which maximizes the likelihood of $\mathbf{x}$.

We then optimize the single sample Monte Carlo estimate of ELBO (3)

$$\mathcal{L}(\mathbf{x},\mathbf{y}) = \log \frac{p_\theta(\mathbf{z}^{(l)})}{q_\phi(\mathbf{z}^{(l)}|\mathbf{x},\mathbf{y})} + \log p_\theta(\mathbf{x}|\mathbf{z}^{(l)},\mathbf{y}), \tag{4}$$

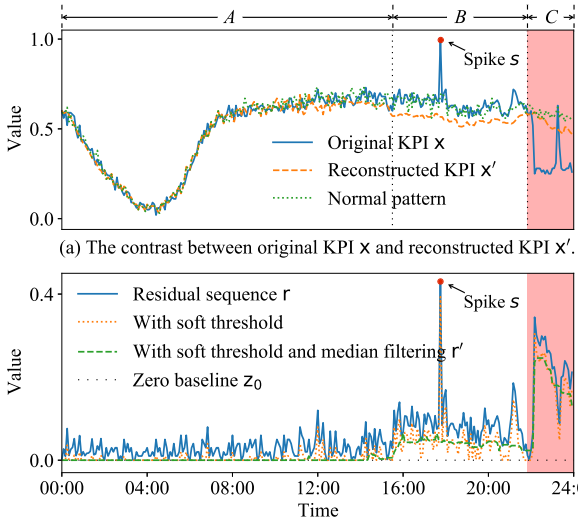
where $\mathbf{z}^{(l)}$ is sampled from $q_\phi(\mathbf{z}^{(l)}|\mathbf{x},\mathbf{y})$.

For each cluster, *OutSpot* can learn the major patterns, which are essentially normal patterns, because most KPIs of each cluster are normal (in our scenario, 82.4% of KPIs are normal). Here, *OutSpot* learns the approximate pattern of historical workload so that the detection results will not be affected when there are small differences between historical and current workload patterns. In addition, we observe that the periodicity of KPIs on a machine does not frequently change since the workload pattern of the machine is pretty stable. Moreover, even if the workload pattern changes, we can retrain the model to adapt to the new pattern.

E. Outlier Detection

As mentioned above, *OutSpot* learns the normal patterns due to the normal KPIs are the majority. Therefore, a normal KPI can be easily reconstructed with little difference to the original one, because the KPI entered during reconstruction is normal and the model represents the normal patterns. However, an outlier KPI, containing lots of abnormal values which have been get rid of during reconstruction, is supposed to be significantly different from its reconstructed data. Consequently, we determine whether KPI $\mathbf{x}$ is an outlier by measuring the difference between $\mathbf{x}$ and its reconstructed data $\mathbf{x}'$. The greater the distance, the more likely $\mathbf{x}$ is an outlier.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.



(a) The contrast between original KPI x and reconstructed KPI x′.

(b) The effects of soft threshold technique and median filtering.

Fig. 6. An example of outliers. The red point s in (a) and (b) is the point outlier. The red area in (a) and (b) is the subsequence outlier. Zero baseline $\mathbf{z}_0$ is a line with a value of zero which means the original and reconstructed time series ($\mathbf{x}$ and $\mathbf{x}'$) are identical.

Unfortunately, the reconstructed data does not always work well as expected, especially for an outlier KPI experiencing long-period abnormal behaviors. For example, as shown in Fig. 6(a), the original KPI suffers from a long time decline (i.e., the area C), while the reconstructed data has a slightly decreasing trend (i.e., the area B) in front of the decline area which is unexpected. This will affect the measurement of their difference. The main reason is that the generative model reconstructs the entire one-day-long KPI data rather than part of the KPI data. The reconstruction process can be impacted by every part of the one-day-long KPI data, including the decline in area C. Therefore, the decline of area C may also cause the decline of its adjacent area B. In addition, it is normal for the reconstruction data to decline slightly in area C, because this input KPI is the outlier in area C (i.e., decline), which will partially affect the reconstruction KPI, resulting in some deviations from the normal patterns. However, it does not affect outlier detection because the reconstructed KPI $\mathbf{x}'$ is still very different from the original KPI $\mathbf{x}$. Moreover, we do not intend to detect the point outlier, so how to ignore the spike (or dip) points (i.e., the spike point s) when measuring the difference is a vital problem. In addition, noises are not unusual even in the normal part of an outlier KPI (i.e., the area A in Fig. 6(b)), which can affect the distance measuring.

To measure the distance, *OutSpot* first gets the residual sequence by calculating the absolute values of the original KPI and the reconstructed one, i.e., $\mathbf{r} = |\mathbf{x} - \mathbf{x}'|$, and then *OutSpot* employs Manhattan distance [32] of the residual sequence $\mathbf{r}$ and the zero baseline $\mathbf{z}_0$: $d(\mathbf{r}, \mathbf{z}_0) = \sum_{i=1}^{n} |r_i - z_{0i}|$, where $\mathbf{r}$ is $(r_1, r_2, \ldots, r_n)$ and $\mathbf{z}_0$ is $(z_{01}, z_{02}, \ldots, z_{0n})$.

Now the problems of unexpected decreasing, the spike point and the noises are reflected in the residual sequence $\mathbf{r}$. To solve these problems, we first apply a soft threshold technique to $\mathbf{r}$ to reduce the impact of unexpected decreasing and noises, and then apply median filtering to $\mathbf{r}$ to remove the spikes (or dips).

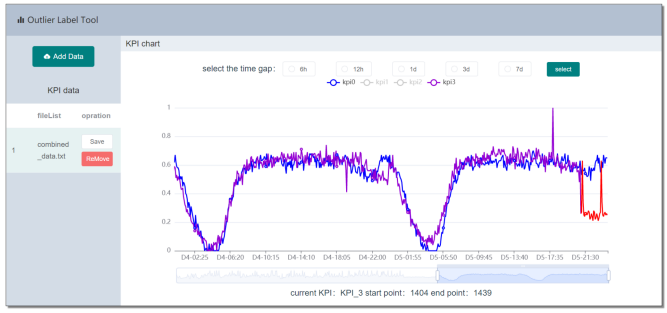


Fig. 7. The interface of the labeling tool.

The soft threshold technique makes smooth transitions between the original and deleted values. It sets values below the threshold τ_s to zero and subtracts the τ_s from original values above τ_s (or adds τ_s to original values below $-\tau_s$)

$$v_s = \begin{cases} 0 & \text{for } |v| \le \tau_s \\ v - \tau_s & \text{for } v > \tau_s \\ v + \tau_s & \text{for } v < -\tau_s \end{cases}, \tag{5}$$

where v and v_s represents the value of a sequence before and after applying soft threshold ($\mathbf{r}$ in *OutSpot*), respectively.

Median filtering can effectively remove impulse noise including point outliers. Its process is accomplished by sliding a window over the KPI, and replacing the original value at the center of the window with the median value in the window [36].

Now we get a new residual sequence $\mathbf{r}'$. Then we compute the Manhattan distance $d(\mathbf{r}', \mathbf{z}_0)$ between the new residual sequence $\mathbf{r}'$ and zero baseline $\mathbf{z}_0$. Finally, with a distance threshold τ_d, we determine the KPI $\mathbf{x}$ is an outlier if $d(\mathbf{r}', \mathbf{z}_0) > \tau_d$. τ_d is an empirical value, which can vary according to different scenarios. In our scenario, for example, the operators think that an outlier KPI should have continuous abnormal values of 1.5 hours (18 points for the interval of 5 minutes) at least, and the abnormal range should be greater than 20%. Then we calculate this threshold in this way: $\tau_d = (20\% - \tau_s) \times 18$. We set the value of the threshold for all the machines of a datacenter instead of for each machine. It is because the value represents the normal patterns of all the machines in the datacenter.

F. Labeling Tool

Although *OutSpot* is an unsupervised method, we still need labeled data to verify or improve its performance. Therefore, we develop a labeling tool with a friendly graphical user interface (GUI), with which operators can label KPI outliers visually. This labeling tool is implemented in Python with Vue.js and FLASK, with about 1000 lines of code. The interface of this labeling tool is shown in Fig. 7, and its workflow is as follows:

1) Users (i.e., operators) upload the original data to be labeled.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

TABLE II
STATISTICS FOR THE DATASET (AN OUTLIER KPI IS A ONE-DAY-LONG KPI CONTAINING ONE OR MORE OUTLIERS)

Dataset	Total KPIs	Total outlier KPIs	Subsequence outlier KPIs	Outlier time series KPIs	Monitering interval	Monitering duration	Data points
$\mathcal{A}$	3600	633	300	333	5 min	eight days	8.3 million
$\mathcal{B}$	1988	98	55	43	15 min	eight days	1.53 million

2) Users use the labeling tool to visualize the data to be labeled and its historical data. They can now visualize, drag, zoom in, or zoom out any segment.
3) Users select the beginning and end of a segment for outlier labeling, and the labeled segment will be highlighted in the labeling tool. At the same time, the labeling results will also be recorded.
4) Users can download the labeled result after the data labeling process.

For the example shown in Fig. 7, the data of Day 4 and Day 5 of the two KPIs (kpi0 and kpi3) are visualized, respectively. Comparing it with its historical data shows that kpi3 experiences an outlier (a subsequence outlier) around 21:30 on Day 5, so operators label this segment an outlier. In addition, the Day 1 to Day 4 displayed without zooming in is also an outlier, and the outlier type is the outlier time series. On the one hand, the labeling tool helps us observe and label data more conveniently. In this paper, three experienced operators utilize this labeling tool to label the datasets, ensuring accurate labeling results. On the other hand, the labeled data is used to evaluate the performance of *OutSpot* from many aspects, such as the overall accuracy, computational efficiency, and the effects of the main components (more details can be seen in Section IV).

IV. EVALUATION

To evaluate the effectiveness and efficiency of *OutSpot*, we perform extensive experiments to answer the following four research questions.

- *RQ1 (Overall accuracy):* How accurate is *OutSpot* in outlier detection compared to baseline methods?
- *RQ2 (Computational efficiency):* Is *OutSpot* computationally efficient enough for large-scale datacenters?
- *RQ3 (Effect of main components):* In this work, we propose to combine HAC with CVAE, and present soft threshold and median filter for outlier determination. How prominently do they impact the effectiveness of *OutSpot*?
- *RQ4 (Effect of hyper-parameters):* How do the hyper-parameters of *OutSpot* impact its effectiveness?

A. Experimental Setup

1) Datasets: To verify the effectiveness of *OutSpot*, we collect data from two different datacenters. The statistics of the two datasets are shown in Table II, which are $\mathcal{A}$ and $\mathcal{B}$, respectively.

Dataset $\mathcal{A}$ comes from a top global short video service provider that provides services for hundreds of millions of DAU. It consists of the eight-day-long monitoring data of 18 KPIs collected from 200 different servers. Dataset $\mathcal{B}$ is collected from one of the three major domestic communication operators of China that provide network services to billions of users. It is composed of the eight-day-long monitoring data from 54 work orders (from the wireless base station), each work order includes 37 KPIs.

For the above two datasets, we take the same processing, i.e., divide them into two parts: the data from the first 7 days constitute the training set, and the data from the last day constitute the test set, which has been labeled by experienced operators using the labeling tools developed by us (see Section III-F). Operators use this tool to compare the KPI on the day of detection with its historical data and other KPI data in the same period. Then, outliers are labeled by three experienced operators: Two operators label the outliers independently; When their labels diverge, the third operator is involved and makes the final decision. For the two datasets in this paper, it takes the three operators about four weeks to complete the labeling work. At last, we count the number of outlier KPIs.

2) Evaluation Metrics: We use two metrics, the best $F1\,Score$ ($F1_{best}$) and Area Under receiver operating characteristic curve (AUC), to evaluate how accurate each method is for outlier detection. To obtain more reliable results, we repeat all the experiments ten times to calculate the average $F1_{best}$ and AUC.

Specifically, KPI outlier detection is essentially a two-class classification problem, i.e., classifying KPIs into outlier KPIs and normal KPIs. $F1\,Score$, taking both precision and recall into account, is usually applied to evaluate the effectiveness of a two-class classification method. Therefore, we choose $F1\,Score$ as our evaluation metric. It is calculated as follows: $F1\,score = \frac{2\times Precision\times Recall}{Precision+Recall}$, where $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$. After enumerating all possible thresholds, we can get the best $F1\,Score$, i.e., $F1_{best}$, for each method. $F1_{best}$ represents the best performance of a method on a dataset.

An ROC curve is plotted based on the true positive rate ($TPR = \frac{TP}{TP+FN}$) and false positive rate ($FPR = \frac{FP}{TN+FP}$) by enumerating all the thresholds. AUC is the area under the ROC curve, and it denotes a method's overall performance on a dataset.

3) Compared Methods: We compare *OutSpot* with some representative outlier detection methods to demonstrate its performance. Since *OutSpot* is designed to detect both subsequence outlier and outlier time series, we compare it with EDBT-15 [26], a representative subsequence outlier detection method, and ICDMW-15 [28], a typical outlier time series detection method. At the same time, we also compare it with DOMI [5], a method for detecting outlier machine instances. Additionally, we also compare it with four state-of-the-art point outlier/anomaly detection methods, including CTF [6], Donut [12], AE-RNN [13], and SR [9].

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

TABLE III
THE $F1_{best}$ AND ITS CORRESPONDING PRECISION, RECALL, AND THE AUC OF EACH METHOD

Dataset	$\mathcal{A}$				$\mathcal{B}$			
Method	Precision	Recall	$F1_{best}$	AUC	Precision	Recall	$F1_{best}$	AUC
OutSpot	**0.9512**	**0.9473**	**0.9492**	**0.9868**	**0.8846**	**0.9388**	**0.9109**	**0.9933**
Donut [12]	0.2087	0.9384	0.3415	0.6095	0.2051	0.7487	0.3221	0.8623
DOMI [5]	0.9019	0.6825	0.7770	0.9384	0.0663	0.9400	0.1238	0.5265
CTF [6]	0.3919	0.8357	0.5335	0.8388	0.2280	0.4400	0.3003	0.8482
AE-RNN [13]	0.2009	0.9526	0.3319	0.5816	0.0858	0.9110	0.1569	0.3957
SR [9]	0.2691	0.4455	0.3355	0.5987	0.1976	0.3300	0.2472	0.7444
EDBT-15 [26]	0.2161	0.6998	0.3302	0.5938	0.0700	0.9900	0.1308	0.6288
ICDMW-15 [28]	0.2075	0.6477	0.3143	0.5344	0.1263	0.4800	0.2000	0.6141

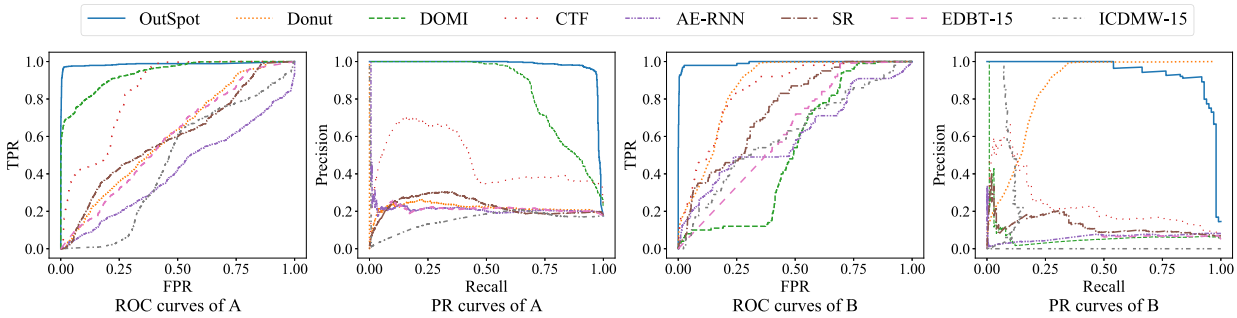


Fig. 8. The ROC curves and PR curves of each method.

4) Hyper-Parameters: For $\mathcal{A}$, we set the hyper-parameters of *OutSpot* as follows. The cluster number threshold $\tau_c = 3$. In CVAE, both the convolutional and deconvolutional architectures have two layers, the kernel sizes and strides of which are $\{6 \times 1, 3 \times 1\}$ and $\{2 \times 1, 1 \times 1\}$, respectively. Moreover, we apply L^2 regularization with a coefficient of 10^{-4} for all layers, and the dimension size of z-space is four. We use Adam optimizer [37], which is reduced by a factor of 0.5 every five epochs, with an initial learning rate of 10^{-3}. Additionally, we run ten epochs for training with early stopping, and the batch size is 18. We set the soft threshold $\tau_s = 0.05$ and the window length of median filter $\omega = 11$. Besides, for $\mathcal{B}$, we set $\tau_c = 5$, batch size $= 37$ and $\omega = 5$. Other parameters are the same as $\mathcal{A}$. The choices of hyper-parameters are detailedly discussed in Section IV-E.

For the setting of some important parameters in the baseline methods, we also made some attempts during the experiment and finally set them to the best for $F1_{best}$ and AUC. For Dount, we set the size of the window to 120, the latent dimension to 8, batch size $= 256$ and epochs $= 250$. For DOMI, we set the dimension of z-space variables to 10 and the number of components of c to 4, epoch $= 10$. For CTF, we set the dimension of z-space variables to 3, the length of input data sequence to 60, batch size $= 50$ and epochs $= 20$. For AE-RNN, we set the number of hidden LSTM units to 8 and the number of autoencoders to 40. For SR, we set the size of the sliding window to 40 and the estimated points number to 5. For EDBT-15, we set the length of the sliding window to 5, PAA size to 3 and alphabet size $= 3$. For ICDMW, we set the number of extracted features to 10.

B. Overall Accuracy (RQ1)

For each method, Table III lists the $F1_{best}$, AUC and its corresponding precision and recall on the test sets of $\mathcal{A}$ and $\mathcal{B}$. We can see that *OutSpot* outperforms the seven baseline methods in terms of both $F1_{best}$ and AUC. More specifically, in $\mathcal{A}$, its $F1_{best}$ and AUC are 0.95 and 0.99, which are 0.17 and 0.05 higher than the best baseline method, respectively. In $\mathcal{B}$, its $F1_{best}$ and AUC are 0.91 and 0.99, which are 0.59 and 0.13 higher than the best baseline method, respectively. Additionally, their precision-recall (PR) curves and receiver operating characteristic (ROC) curves are shown in Fig. 8, respectively. *OutSpot* achieves better performance than the seven baseline methods in both ROC curves and PR curves, demonstrating that it obtains high TPR, low FPR, high precision, and high recall simultaneously.

Donut [12], CTF [6], AE-RNN [13], and SR [9] are both point outlier/anomaly detection methods. They treat each point deviating from normal behavior compared to its historical or adjacent data points as outliers. However, these individual outlier data points usually do not imply a device failure because of auto-recovery and load balancing mechanisms. Thus we do not label them as outliers in our scenario. Therefore, they all generate a large number of false alarms and suffer from low precision.

DOMI [5] is designed for detecting outlier machine instances. Because all KPIs of a machine instance at the same time can represent the state of the machine, it is more concerned with the overall KPI of a machine rather than each KPI. When we apply it to determine whether each KPI is outlier, the effect is not good, especially the $F1_{best}$ in $\mathcal{B}$ is only 0.12. EDBT-15 [26] and

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
THE $F1_{best}$ AND CORRESPONDING PRECISION, RECALL UNDER DIFFERENT OUTLIER TYPES. (T1 REPRESENTS SUBSEQUENCE OUTLIERS, T2 REPRESENTS OUTLIER TIME SERIES, T1+T2 MEANS THE COMBINATION OF EDBT-15 (FOR T1) AND ICDMW-15 (FOR T2) RESULTS)

Method	***OutSpot***				EDBT-15 [26]				ICDMW-15 [28]				EDBT-15 + ICDMW-15	
Dataset	$\mathcal{A}$		$\mathcal{B}$		$\mathcal{A}$		$\mathcal{B}$		$\mathcal{A}$		$\mathcal{B}$		$\mathcal{A}$	$\mathcal{B}$
Outlier Types	T1	T2	T1	T2	T1	T2	T1	T2	T1	T2	T1	T2	T1 + T2	T1 + T2
Precision	0.88	0.89	0.98	0.97	0.10	-	0.04	-	-	0.86	-	0.70	0.20	0.07
Recall	0.97	0.93	0.83	0.70	0.94	-	0.95	-	-	0.13	-	0.03	0.99	0.97
$F1_{best}$	0.92	0.91	0.90	0.81	0.19	-	0.08	-	-	0.22	-	0.06	0.33	0.13

ICDMW-15 [28] are designed for detecting subsequence outliers and outlier time series, respectively. For a KPI, EDBT-15 detects outliers only according to its historical patterns, while ICDMW-15 conducts outlier detection merely based on the patterns of other KPIs in the same period, and neither of them learns the two types of patterns simultaneously. Therefore, both methods miss many outliers, generate many false positives, and thus have low precision.

To verify the ability of *OutSpot* to detect two types of outliers (i.e., subsequence outliers and outlier time series), we have listed the precisions, recalls, and $F1_{best}$ of *OutSpot* in detecting either type of outliers in Table IV on the two datasets, respectively. Additionally, for the two datasets, we have also listed in Table IV the precisions, recalls, and $F1_{best}$ of EDBT-15 [26] in detecting subsequence outliers and of ICDMW-15 [28] in detecting outlier time series, respectively. Moreover, we combine EDBT-15 and ICDMW-15 by adopting EDBT-15 for detecting subsequence outliers and ICDMW-15 for detecting outlier time series and determining outliers whenever either model detects any outliers.

As seen from Table IV, *OutSpot* outperforms EDBT-15, ICDMW-15, and their combination in detecting both outlier types. Since there are many noisy data points in our scenario, and EDBT-15 is sensitive to them, it generates many false positives and suffers from low precision. In addition, ICDMW-15 is based on principal component analysis (PCA), and the features (principal components) extracted by PCA can only sketch out the KPI patterns. Therefore, it cannot accurately quantify the difference between the various KPI patterns in our scenario. In some cases, the normal KPIs cannot be distinguished from the outlier KPIs, resulting in high false negatives (i.e., low recall). Finally, the combination of the two methods cannot mitigate the intrinsic shortcomings of these two methods and still suffers from a low $F1_{best}$.

Furthermore, it can be seen from Tables III and IV that all methods (including *OutSpot*) perform worse in $\mathcal{B}$ than $\mathcal{A}$. The more important reason is that the $\mathcal{B}$ has a large monitoring interval (15 min), resulting in a small amount of data. At the same time, the $\mathcal{B}$ comes from the wireless base station, so compared with the $\mathcal{A}$ from the Web service, its KPI data is more complicated.

C. Computational Efficiency (RQ2)

As aforementioned, to detect outliers for the massive number of KPIs in large-scale datacenters, we propose to combine HAC with CVAE in *OutSpot*. Because *OutSpot* trains only one model for all KPIs, the computational efficiency is significantly improved. However, the previously-proposed deep generative model-based methods, including Donut and AE-RNN, have to train a separate model for each KPI, and thus consume much more computational resources and suffer from low computational efficiency.

TABLE V
THE TRAINING TIME OF *OUTSPOT*, DONUT, AND DAGMM ON THE TRAINING SET AND ON ONE MILLION KPIS IN DATASET $\mathcal{A}$ AND $\mathcal{B}$

Method	Training set		One million KPIs	
	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}$	$\mathcal{B}$
OutSpot	**63.3s**	**23.4s**	**4.89h**	**3.24h**
Donut [12]	11.7h	3.57h	4.53 month	2.48 month
DOMI [13]	88.8s	45.1s	6.85h	6.27h
CTF [13]	2486s	447s	192h	62.15h
AE-RNN [13]	18.1h	7.69h	6.91 month	5.35 month

More specifically, we implement *OutSpot*, Donut, DOMI, CTF and AE-RNN with Python, and run them on a server with 2*16-Core Intel(R) Xeon(R) Gold 5218 CPU @2.30 GHz and 192 G RAM. As shown in Table V, for the training set containing 3600 ($\mathcal{A}$) and 1998 ($\mathcal{B}$) KPIs, it takes *OutSpot* 63.3 s and 23.36 s to train the model, which is the least among the above methods. To more intuitively compare the five methods' efficiency, we calculate the training time of the five methods when we apply them to conduct outlier detection for one million KPIs. *OutSpot* costs 4.89 h and 3.24 h to finish training for two training sets, which is quite acceptable in practice. However, both Donut, DOMI, CTF and AE-RNN take a long time for training, making them inappropriate in our scenario, especially Donut and AE-RNN. In addition, we can see that the efficiency of DOMI is close to that of *OutSpot*, which is mainly due to the fact that it is mainly aimed at machine instances, not every KPI. Please note that none of SR, EDBT-15, or ICDMW-15 needs model training. However, as we can see from Table III, all the three methods suffer from low precision and low recall, and none of them can be used for outlier detection in practice.

D. Effect of Main Components (RQ3)

As aforementioned, the main technical contributions of this work are: 1) we propose to integrate HAC with CVAE to detect both subsequence outlier and outlier time series for a considerable number of KPIs with various types, and 2) we design two simple yet effective techniques, ST and MF, to accurately determine outlier KPIs. Therefore, we evaluate the effect of

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
THE $F1_{best}$ AND AUC OF DIFFERENT METHODS

Dataset	$\mathcal{A}$				$\mathcal{B}$			
Method	Precision	Recall	$F1_{best}$	AUC	Precision	Recall	$F1_{best}$	AUC
OutSpot	**0.9512**	**0.9473**	**0.9492**	**0.9868**	**0.8846**	**0.9388**	**0.9109**	**0.9933**
"*OutSpot* w/o C"	0.9027	0.8357	0.8679	0.9821	0.8375	0.6837	0.7528	0.9685
"*OutSpot* w/o ST"	0.6925	0.9463	0.7997	0.9654	0.7590	0.6429	0.6961	0.9628
"*OutSpot* w/o MF"	0.8205	0.8736	0.8462	0.9786	0.6764	0.7041	0.6900	0.9581
"*OutSpot* w RP"	0.7331	0.6161	0.6695	0.9116	0.3465	0.7143	0.4667	0.9327

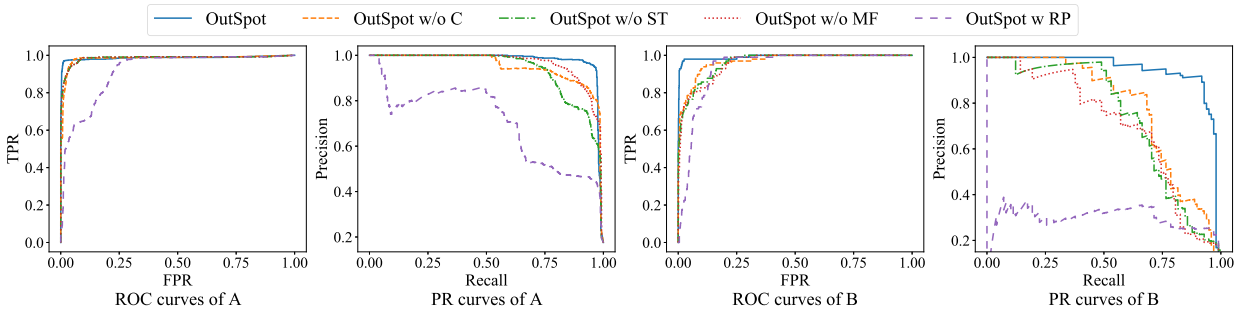


Fig. 9. The ROC curves and PR curves of different methods.

these main components on *OutSpot*'s overall performance by removing one or two of them from *OutSpot* as follows.

- "OutSpot*w/o C*": We replace the combination of HAC and CVAE with VAE in *OutSpot*.
- "OutSpot*w/o ST*": We remove ST from *OutSpot*.
- "OutSpot*w/o MF*": We remove MF from *OutSpot*.
- "OutSpot*w RP*": We replace the combination of ST and MF with "reconstruction probability" (RP).

Table VI lists, for each above method, the AUC as well as $F1_{best}$ and its corresponding precision and recall. Moreover, Fig. 9 shows the ROC curves and PR curves of different methods, respectively. *OutSpot* outperforms all the other methods in terms of $F1_{best}$ and AUC.

Without the clustering information embedded in the deep generative model, VAE merely learns the historical pattern of all types of KPIs, and cannot comprehensively capture the patterns of historical KPIs and those of KPIs in the same period, likely leading to more false alarms and miss more true outliers. Therefore, "*OutSpot* w/o C" achieves lower precision and lower recall than *OutSpot*.

Since ST and MF respectively alleviate the impact of reasonable variance and point outliers on the performance of *OutSpot*, removing them leads to that both "*OutSpot* w/o ST" and "*OutSpot* w/o MF" suffer from more false alarms and lower precision. Additionally, the reconstruction probability-based methods can frequently assign a higher reconstruction probability to outliers and thus suffer from low accuracy in outlier determination for high-dimensional data [15]. Therefore, they are inappropriate for subsequence outliers and outlier time series, both of which are high-dimensional data containing multiple data points. As shown in Table VI and Fig. 9, our experiments have verified this point, because "*OutSpot* w PR" degrade both the precision and recall of *OutSpot*.

E. Effect of Hyper-Parameters (RQ4)

Several essential hyper-parameters may impact the performance of *OutSpot*, including τ_c (cluster number threshold), dimension size of z-space, number of epochs, ω (window size of median filter), and τ_s (soft threshold). To measure these hyper-parameters' effect on *OutSpot*, we calculate *OutSpot*'s $F1_{best}$ and AUC as the values of them vary, as shown in Figs. 10 and 11. More specifically, a larger dimension size of z usually leads to a stronger representation ability, but it can lead to higher training overhead. Similarly, the number of epochs indicates the number of complete passes through the training set. A smaller number of epochs may result in insufficient model training, but a larger number of epochs will degrade the computational efficiency.

For $\mathcal{A}$ (Fig. 10), we can see that τ_c impacts little on the effectiveness (in terms of $F1_{best}$ and AUC) of *OutSpot*, and *OutSpot* achieves relatively high $F1_{best}$ and AUC when $\tau_c = 3$ or $\tau_c = 6$. Therefore, we set $\tau_c = 3$ in our scenario. As the dimension size of z-space increases, the effectiveness of *OutSpot* improves and becomes stable when its value reaches four. Thus the dimension size of z-space is four in our scenario. Similarly, the number of epochs is eight, $\omega = 11$, and $\tau_s = 0.05$ in our scenario.

For $\mathcal{B}$ (Fig. 11), relative to $\mathcal{A}$, we can see that τ_c, the dimension size of z and the number of epochs have a greater impact on the effectiveness of *OutSpot*, but the optimal parameters appear in a similar range to $\mathcal{A}$. As the parameter size increases, the optimal parameters are quickly found, i.e., $\tau_c = 5$, the dimension size of z is four and the number of epochs is eight. In addition, we can see that the parameters ω and τ_s have little effect on the effectiveness of *OutSpot*, and the best results are achieved when taking 5 and 0.05 in our scenario, respectively.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

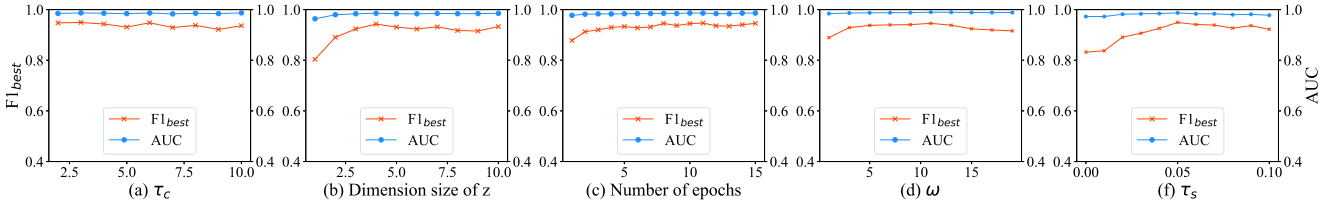


Fig. 10. The $F1_{best}$ and AUC of *OutSpot* as the values of hyper-parameters vary in dataset $\mathcal{A}$.

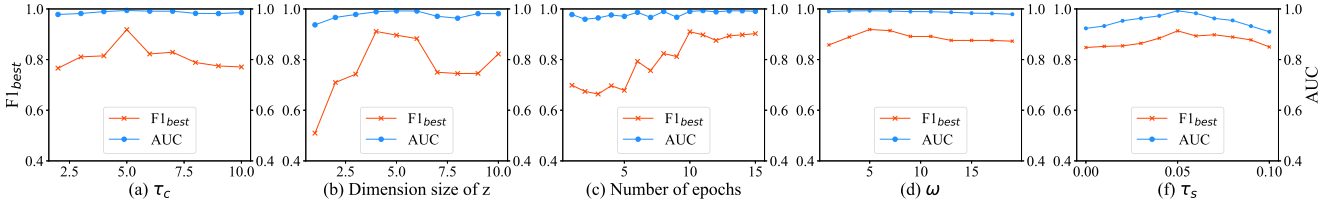


Fig. 11. The $F1_{best}$ and AUC of *OutSpot* as the values of hyper-parameters vary in dataset $\mathcal{B}$.

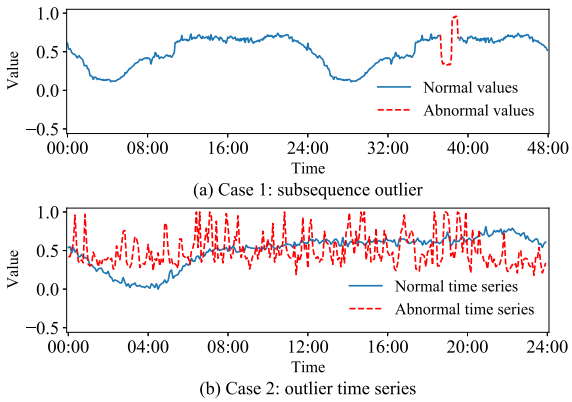


Fig. 12. Two real-world KPI outlier cases detected by *OutSpot*.

V. Discussion

A. Case Study

OutSpot is being deployed on the datacenters of a top-tier global short video service provider that provides services for hundreds of millions of DAU. During the process of deployment, we find the following two interesting cases.

The monitoring data of *rx_pkts_eth0*, representing the number of received packets, suffered from a 2.5-hour-long subsequence outlier, as shown in the red dash line of Fig. 12(a). *OutSpot* believed that the pattern of this outlier segment deviated from normal patterns, according to the historical patterns of all the KPIs that share the same cluster with this KPI. The outlier score generated by *OutSpot* is 8.80, significantly higher than the threshold (2.30 in our scenario). Operators confirmed this KPI outlier, and found that it was caused by a top-of-rack (ToR) switch failure. After operators mitigated the switch failure, the KPI returned to normal status.

Additionally, *OutSpot* detected an outlier time series on *cpu_user*, which denotes the CPU utilization at user level, as shown in Fig. 12(b). *OutSpot* determined this KPI as an outlier because its pattern deviated significantly from other KPIs in the same period. After careful investigations, operators found that this server had been poorly managed for one week. The Spark software on this server failed one week ago, and the Spark system randomly assigned jobs to this server. Since this server had been experiencing outlier for a week, the outlier/anomaly detection methods according to only a specific KPI's historical patterns, including Donut [12], AE-RNN [13], and SR [9], EDBT-15 [26], etc., can hardly find this type of outliers (outlier time series).

B. The Limitations of OutSpot

OutSpot has two main limitations as follows:

1) After clustering KPIs through HAC, a cluster usually represents a normal pattern. The main reasons are:

a) A subsequence outlier KPI still has many similarities with the normal KPIs. Therefore, the subsequence outlier KPIs often do not form a specific cluster but are assigned to different clusters, most of which are normal KPIs.

b) The patterns of outlier time series KPIs are so different that they tend to be clustered into a close cluster, most of which are normal KPIs. In extreme cases, when the outlier KPIs occupy a large portion of all KPIs, one or more clusters may contain only outlier ones. But in this case, operators can easily find them. For example, operators can determine whether outlier patterns dominate a cluster by manually checking its central KPI through a labeling tool like the one introduced in Section III-F. Operators can determine that outlier patterns dominate the cluster if its central KPI is an outlier KPI. Then we use the remaining clusters to train the model. Furthermore, after investigating extensive real-world KPIs, we find outlier KPIs rarely dominate a cluster. For example, in our scenario, the outlier KPIs occupy 17.58% and 4.9% of all KPIs in $\mathcal{A}$ and $\mathcal{B}$, respectively. No outlier clusters appear when we increase τ_c from 2 to 10.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

2) A subsequence outlier of ε may be missed by *OutSpot* in the following scenario. ε has different patterns with its historical patterns, but its pattern resembles one of the historical patterns of another KPI sharing the same cluster with ε. In this case, *OutSpot* believes that ε does not suffer from an outlier, because *OutSpot* is trained according to the historical patterns of the KPIs having the same cluster-ID with ε. Nevertheless, after careful investigation on the 3,600 KPIs in the experiment, we do not find any such ε. We will design a method to address the challenge imposed by such ε in the future.

VI. Related Work

Recently, a large number of time-series outlier detection methods have been proposed in the literature. The majority of these works focus on point outlier detection [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [16], [18], [19], [20], [21], [22], [23], [24], [25], [34]. Additionally, a few works have been conducted on subsequence outlier detection [26], [27], [38], [39], outlier time series detection [16], [17], [28], [40] and machine instance outlier detection [5]. However, none of these works can detect both subsequence outlier and outlier time series simultaneously.

Among the point outlier detection methods, Donut [12] applied VAE, a typical generative model, for KPI point outlier/anomaly detection for the first time. It learned the normal patterns of historical data using VAE and determined whether a KPI data point was anomalous through reconstruction probability. CTF [6] combined clustering and transfer learning to improve the scalability of existing anomaly detection algorithms, which makes it efficient to detect a large number of KPIs simultaneously. AE-RNN [13] integrated multiple autoencoders (AE) through RNN. It applied the median reconstruction error of multiple autoencoders to determine whether a KPI was an outlier. SR [9] applied the spectral residual model to obtain the significant part of the time series, i.e., the outlier part. However, none of the above methods could learn the pattern of all KPIs in the same period and accurately detect outlier time series. Additionally, they were designed mainly for detecting point outliers, which are ignored in our scenario.

The subsequence outlier detection methods usually conducted outlier detection through learning KPIs' historical patterns [26], [27], [38], [39]. However, they did not learn the pattern of all KPIs in the same period. For example, EDBT-15 [26], a typical subsequence outlier detection method, discretized a time series into symbolic form, and performed numerical reduction and grammatical induction to obtain variable-length strings. Moreover, the outlier time series detection methods detected outliers based on the patterns of all the time series in the same period [16], [17], [28], [40]. For instance, ICDMW-15 [28], a representative outlier time series detection method, integrated principal components analysis (PCA) with α-convex hulls. Nevertheless, none of the above methods can detect both subsequence outlier and outlier time series simultaneously, and thus they are inappropriate to our scenario. Besides, DOMI [5] is a typical machine instance outlier detection method, which learns normal machine instance patterns through GMVAE to find outlier machines. However, its main target is the entire machine instance rather than each KPI, so it is inappropriate to our scenario either.

VII. Conclusion

In large-scale datacenters, outlier detection for a large number of various-type KPIs is vitally important. In this work, we propose *OutSpot*, an efficient and robust outlier detection framework, which can detect subsequence outlier and outlier time series simultaneously. *OutSpot* combines HAC and CVAE to learn the historical pattern of each KPI and the patterns of all KPIs in the same period. We applied ST and MF to solve the challenges introduced by point outliers and reasonable variance during the detection process. Moreover, we also develop a labeling tool to help operators label KPI outliers. Extensive experiments using two real-world datasets (including 3600 and 1988 KPIs, respectively) demonstrate that *OutSpot* achieves that $F1_{best} = 0.95\ and\ 0.91$, $AUC = 0.99\ and\ 0.99$, significantly outperforming the seven baseline methods. The core idea of *OutSpot* can be applied for more scenarios beyond large-scale datacenters, e.g., IoT devices and mobile devices. In the future, we will verify *OutSpot*'s performance in more scenarios.

References

[1] C. Guo et al., "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 139–152.

[2] S. Moss, "Visa details cause of widespread outage, blames data center switch failure," 2018. [Online]. Available: https://www.datacenterdynamics.com/en/news/visa-details-cause-of-widespread-outage-blames-data-center-switch-failure/

[3] Incident review: Core router outages, 2020. [Online]. Available: https://www.freistil.it/incident-review-core-router-outage/

[4] L. Dai et al., "SDFVAE: Static and dynamic factorized VAE for anomaly detection of multivariate CDN KPIs," in *Proc. Web Conf.*, 2021, pp. 3076–3086.

[5] Y. Su et al., "Detecting outlier machine instances through Gaussian mixture variational autoencoder with one dimensional CNN," *IEEE Trans. Comput.*, vol. 71, no. 4, pp. 892–905, Apr. 2022.

[6] M. Sun et al., "CTF: Anomaly detection in high-dimensional time series with coarse-to-fine model transfer," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[7] M. Ma et al., "Jump-starting multivariate time series anomaly detection for online service systems," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 413–426.

[8] X. Zhang et al., "Cross-dataset time series anomaly detection for cloud systems," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 1063–1076.

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

[10] Z. He et al., "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 1705–1719, Apr. 2023.

[11] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," 2020, *arXiv:2002.04236*.

[12] H. Xu et al., "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications," in *Proc. World Wide Web Conf.*, 2018, pp. 187–196.

[13] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Outlier detection for time series with recurrent autoencoder ensembles," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 2725–2732.

[14] B. Zong et al., "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proc. Int. Conf. Learn. Representations*, 2018.

[15] Z. Wang, B. Dai, D. Wipf, and J. Zhu, "Further analysis of outlier detection with deep generative models," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, Art. no. 753.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

[16] L. Beggel, B. X. Kausler, M. Schiegg, M. Pfeiffer, and B. Bischl, "Time series anomaly detection based on shapelet learning," *Comput. Statist.*, vol. 34, no. 3, pp. 945–976, 2019.
[17] S.-E. Benkabou, K. Benabdeslem, and B. Canitia, "Unsupervised outlier detection for time series by entropy and dynamic time warping," *Knowl. Inf. Syst.*, vol. 54, no. 2, pp. 463–486, 2018.
[18] D. Liu et al., "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proc. ACM Internet Meas. Conf.*, 2015, pp. 211–224.
[19] W. Chen et al., "Unsupervised anomaly detection for intricate KPIs via adversarial training of VAE," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1891–1899.
[20] 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. Discov. Data Mining*, 2019, pp. 2828–2837.
[21] 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.*, 2018, pp. 1–9.
[22] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1939–1947.
[23] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 665–674.
[24] J. Bu et al., "Rapid deployment of anomaly detection models for large number of emerging KPI streams," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf.*, 2018, pp. 1–8.
[25] G. Pang, C. Shen, and A. van den Hengel, "Deep anomaly detection with deviation networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 353–362.
[26] P. Senin et al., "Time series anomaly discovery with grammar-based compression," in *Proc. 18th Int. Conf. Extending Database Technol.*, 2015, pp. 481–492.
[27] D. Carrera, B. Rossi, P. Fragneto, and G. Boracchi, "Online anomaly detection for long-term ECG monitoring using wearable devices," *Pattern Recognit.*, vol. 88, pp. 482–492, 2019.
[28] R. J. Hyndman, E. Wang, and N. Laptev, "Large-scale unusual time series detection," in *Proc. IEEE Int. Conf. Data Mining Workshop*, 2015, pp. 1616–1619.
[29] A. Pol, V. Berger, G. Cerminara, C. Germain, and M. Pierini, "Trigger rate anomaly detection with conditional variational autoencoders at the CMS experiment," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019.
[30] K. Sohn, X. Yan, and H. Lee, "Learning structured output representation using deep conditional generative models," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, Cambridge, MA, USA, MIT Press, 2015, pp. 3483–3491.
[31] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, Cambridge, MA, USA, MIT Press, 2014, pp. 3581–3589.
[32] Deeksha and S. Sahu, "Finding similarity in articles using various clustering techniques," in *Proc. IEEE 6th Int. Conf. Rel. Infocom Technol. Optim. (Trends Future Directions)*, 2017, pp. 344–347.
[33] A. Großwendt, H. Röglin, and M. Schmidt, "Analysis of ward's method," in *Proc. 30th Annu. ACM-SIAM Symp. Discrete Algorithms*, SIAM, 2019, pp. 2939–2957.
[34] Y. Su et al., "CoFlux: Robustly correlating KPIs by fluctuations for service troubleshooting," in *Proc. IEEE/ACM 27th Int. Symp. Qual. Service*, 2019, pp. 1–10.
[35] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3483–3491.
[36] C. T. Leondes, *Multidimensional Systems: Signal Processing and Modeling Techniques: Advances in Theory and Applications*. Amsterdam, Netherlands: Elsevier, 1995.
[37] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, "Slime mould algorithm: A new method for stochastic optimization," *Future Gener. Comput. Syst.*, vol. 111, pp. 300–323, 2020.
[38] P. M. Chau, B. M. Duc, and D. T. Anh, "Discord discovery in streaming time series based on an improved HOT SAX algorithm," in *Proc. 9th Int. Symp. Inf. Commun. Technol.*, 2018, pp. 24–30.
[39] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, pp. 1261–1274, 2018.
[40] J. A. Lara, D. Lizcano, V. Rampérez, and J. Soriano, "A method for outlier detection based on cluster analysis and visual expert criteria," *Expert Syst.*, vol. 37, no. 5, 2020, Art. no. e12473.

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

Daguo Cheng received the BS degree in information security from the School of Computer Science and Cyberspace Security, Hainan University, Haikou, China, in 2019, and the MS degree from the College of Software, Nankai University, Tianjin, China, in 2022. He is currently working toward the PhD degree with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China. His research interests include anomaly detection, root cause localization.

Tiankai Yang received the BS degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2021. He is currently working toward the MS degree with Viterbi School of Engineering, University of Southern California, California. His research interests include anomaly detection, machine learning and data science.

Yuhe Ji received the BS degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2022. He is currently working toward the MS degree with the College of Software, Nankai University. His research interests include data science and anomaly detection.

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

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.

Man Zhu received the BS degree in electronic information science and technology from Qufu Normal University, Rizhao, China, in 2019. She is currently working toward the master's degree with the College of Software, Nankai University, Tianjin, China. Her research interests include anomaly detection and root cause localization.

Xiao Xiong received the BS degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2022. He is currently working toward the MS degree with the College of Software, Nankai University. His research interests include machine learning and failure diagnosis.

Qiliang Fan received the BS degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2021. He is currently working toward the MS degree with the College of Software, Nankai University. His research interests include anomaly detection and root cause localization.

Minghan Liang received the BS degree in software engineering from the College of Software, Nankai University, Tianjin, China, in 2021. He is currently working toward the MS degree with the College of Software, Nankai University. His research interests include anomaly detection and root cause localization.

Dan Pei (Senior Member, IEEE) received the BE and MS degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, in 1997 and 2000, respectively, and the PhD degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA), in 2005. He is currently an associate professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include network and service management in general. He is an ACM senior member.

Tianchi Ma received the BS degree from the Beijing University of Posts and Telecommunications. He has worked on Big Data area in internet companies for more than 10 years. Now he is a technical specialist in Kuaishou Technology, working on DataOps.

Yu Chen received the BS and MS degrees in computer science from Peking University, in 1998 and 2001, respectively. Then he worked as a researcher in Microsoft Research Asia, during which his research interests include distributed systems and information retrieval. He is now a SRE specialist in Kuaishou Technology, working on algorithms on AIOps.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on October 16,2023 at 05:23:06 UTC from IEEE Xplore. Restrictions apply.