Detecting Outlier Machine Instances Through Gaussian Mixture Variational Autoencoder With One Dimensional CNN

Ya Su, Youjian Zhao, Ming Sun, Shenglin Zhang, *Member, IEEE*, Xidao Wen, Yongsu Zhang, Xian Liu, Xiaozhou Liu, Junliang Tang, Wenfei Wu, *Member, IEEE*, and Dan Pei, *Senior Member, IEEE*

Abstract—Today's large datacenters house a massive number of machines, each of which is being closely monitored with multivariate time series (e.g., CPU idle, memory utilization) to ensure service quality. Detecting outlier machine instances with multivariate time series is crucial for service management. However, it is a challenging task due to the multiple classes and various shapes, high dimensionality, and lack of labels of multivariate time series. In this article, we propose DOMI, a novel unsupervised model that combines Gaussian mixture VAE with 1D-CNN, to **d**etect **o**utlier **m**achine **i**nstances. Its core idea is to capture the normal patterns of machine instances by learning their latent representations that consider the shape characteristics, reconstruct input data by the learned representations, and apply reconstruction probabilities to determine outliers. Moreover, DOMI interprets the detected outlier instance based on the reconstruction probability changes of univariate time series. Extensive experiments have been conducted on the dataset collected from 1821 machines with a 1.5-month-period, which are deployed in ByteDance, a top global content service provider. DOMI achieves the best F1-Score of 0.94 and AUC score of 0.99, significantly outperforming the best performing baseline method by 0.08 and 0.03, respectively. Moreover, its interpretation accuracy is up to 0.93.

Index Terms—Outlier machine instances, multivariate time series, service management, 1D-CNN, GMVAE

1 Introduction

Modern large datacenters usually deploy hundreds of thousands to millions of machines, including physical servers, virtual machines, dockers, to support diverse types of Internet-based services [1], [2]. About 5%∼18%[1] of these machines suffer from software bugs and/or hardware failures per year. The unexpected failures may cause data loss and resource congestion due to machines being unavailable [3], which can heavily degrade the quality of services (QoS) and reduce revenue [4]. Therefore, operation engineers carefully monitor machine metrics, such as CPU idle, memory utilization, TCP retransmission rate, to obtain a global view of each machine's status [5]. The monitoring data of each metric forms a univariate time series, and thus each machine can be represented as an entity with multivariate time series [6], [7].

A large number of outlier and anomaly detection works have been proposed over the years in order to alarm operation engineers soon after a failure occurs. They usually detect anomalies of univariate time series [8], [9], [10], [11], [12] or multivariate time series [5], [6], [13], [14], [15], [16], [17] *at each timestamp*. However, a machine anomaly at a time point does not necessarily lead to a machine/service failure due to the auto-recovery and load balancing mechanisms of today's cloud systems [7]. Therefore, it is of vital importance to detect "outlier machine instances" whose behaviors deviate from normal ones *over a long period* (e.g., one day in our scenario),[2] which can help operation engineers find the abnormal machines more likely causing machine/service failures.

Formally, a machine instance can be denoted as a "multivariate time series of Machine-X at Day-Y" (*M-X@D-Y* for short). Generally, an outlier is a data point that deviates much from other data [18]. Based on our observation, an outlier machine instance is unexpected and rare (e.g., less than 20 percent of machine instances are outliers in our scenario), and its shape is different from those of normal ones in the majority. Consequently, *detecting outlier machine instances equals to finding the machine instances whose shapes deviate from the majority in a dataset (i.e., normal machine instances).* It is inevitable that a datacenter could be unbalanced. In this case, the patterns of normal machine instances

1. Online. [Available]: https://www.statista.com/statistics/430769/annual-failure-rates-of-servers/

2. The length of this period is empirically determined by operation engineers.

• *Ya Su, Youjian Zhao, Xidao Wen, and Dan Pei are with the Department of Computer Science, Tsinghua University, Beijing 100084, China, and also with the Beijing National Research Center for Information Science and Technology, Beijing 100084, China. E-mail: su-y16@mails.tsinghua.edu.cn, {zhaoyoujian, wenxidao, peidan}@tsinghua.edu.cn.*
• *Ming Sun and Wenfei Wu are with the Department of IIIS, Tsinghua University, Beijing 100084, China. E-mail: sunm19@mails.tsinghua.edu.cn, wenfeiwu@tsinghua.edu.cn.*
• *Shenglin Zhang is with the Nankai University, Tianjin 300071, China. E-mail: zhangsl@nankai.edu.cn.*
• *Yongsu Zhang, Xian Liu, Xiaozhou Liu, and Junliang Tang are with the ByteDance, Beijing 100080, China. E-mail: {zhangyongsu, liuxian.1, liuxiaozhou, tangjunliang}@bytedance.com.*

Manuscript received 6 Apr. 2020; revised 13 Feb. 2021; accepted 7 Mar. 2021. Date of publication 9 Mar. 2021; date of current version 11 Mar. 2022. (Corresponding author: Shenglin Zhang.) Recommended for acceptance by P. Milder. Digital Object Identifier no. 10.1109/TC.2021.3065073

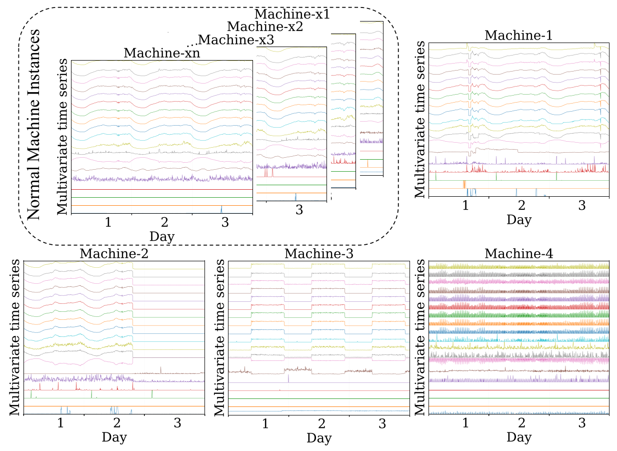


Fig. 1. Examples of normal and outlier machine instances. Each machine is with 19-metric 3-day-long multivariate time series (i.e., with 3 machine instances).

(i.e., normal patterns) would be learned from all instances in this datacenter. In this way, the machine instances which deviate much from the normal patterns are detected as outliers. Then this datacenter would be rebalanced by operation engineers. The heartbeat mechanism can detect unresponsive machines and functional machines being not utilized [19]. The failed machines which are turned down can be detected using logs. However, it is difficult, using logs, to detect the outlier machines that are not turned down but suffering from CPU, memory, network, or other problems. Machine instances can be outliers for different reasons, thus these methods are limited and the proposed method has to detect various types of outlier machine instances.

Fig. 1 shows three types of outlier machine instances: (1) *M-X@D-Y* has a similar shape with normal instances, but it has long-term abnormal shapes (which can last less than 1 day) due to software bugs or hardware failures (e.g., *M-1@D-1*, *M-2@D-2*) [4]. (2) *M-X@D-Y* has a regular shape [20], which however is different from normal ones because it may run unexpected tasks (e.g., *M-2@D-3*, *M-3* at all three days). (3) *M-X@D-Y* has an irregular shape because it is unmanaged (e.g., *M-4* at all three days). As mentioned earlier, the current outlier and anomaly detection methods can not well solve the problem in our scenario. In this work, we aim to *detect outlier machine instances*, which faces the following four major challenges:

- *Multiple classes and various shapes of multivariate time series.* The datacenter usually supports many different classes of services and thus has multiple classes of metrics of machines. It is desirable but also challenging to design a generic algorithm that can be applied to diverse types of multivariate time series with different shape characteristics [20].
- *High dimensionality of multivariate time series. M-X@D-Y* is a $\mathbb{R}^{T\times N}$ matrix, whose dimension is $T \cdot N$ and can be much high (typically ranges from 1,000 to 100,000). The curse of dimensionality could greatly degrade the performance of methods [21], [22].
- *The need of interpretation for outlier machine instances.* Outlier interpretation can help analyze outlier instances and facilitate troubleshooting [5]. It is difficult yet important for a machine learning model to interpret outliers, especially for machine instances that are with various multivariate time series.
- *Lack of labels.* Typically, labeled outlier machine instances are scarce, because it is labor-intensive and time-consuming for experienced operation engineers to label a large number of instances. Therefore, unsupervised algorithms are desirable.

To tackle the above challenges, we present DOMI, an unsupervised model to robustly detect outlier machine instances. Overall, DOMI is a reconstruction based model, which captures the normal patterns of machine instances by learning their latent representations automatically, and reconstructs the input data by the learned representations. It detects outlier instances based on the intuition that the outlier machine instance is difficult to be accurately reconstructed and its reconstruction probability is thus low [5], [8], [23]. Specifically, motivated by the idea that the shape features of images can be effectively extracted using CNN [24], DOMI extracts the shape features of multivariate time series with one dimensional CNN (1D-CNN) [7], [25], [26]. DOMI then learns the low-dimensional latent representations of multivariate time series with multiple normal patterns using Gaussian mixture variational autoencoder (GMVAE) [22], [23], [27]. Both 1D-CNN and GMVAE facilitate dimension reduction. To interpret a detected outlier machine instance, we estimate and rank the contributions (i.e., the changes of reconstruction probability) of its constituent univariate time series in outlier determination [5].

Our contributions are summarized as follows:

- This paper formally defines the problem of detecting outlier machine instances in Internet service management, where each machine instance is represented as a multivariate time series over a long period.
- To the best of our knowledge, DOMI is a new unsupervised CNN-VAE based model that integrates Gaussian mixture prior with 1D convolutional kernels for the first time.
- To better approximate the training loss, we propose a simple yet effective approximation method, multi-sample Monte Carlo integration, to improve the training performance of GMVAE based models.
- To facilitate the interpretable detection of outlier machine instances, for each outlier, we design a new interpretation method based on the reconstruction probability changes of its univariate time series, which is also applicable for other VAE based models such as OmniAnomaly[5] and Donut[8].
- Through extensive experiments on the dataset collected from 1821 machines deployed in ByteDance (a top global content service provider) with a 1.5-month-period, DOMI has been demonstrated to achieve the best F1-Score of 0.94 and AUC score of 0.99, outperforming the best baseline method by 0.08 and 0.03, respectively. The interpretation accuracy of DOMI is up to 0.93.
- We have already published our code[3] and dataset[4] on GitHub for better reproducibility of our results.

3. Online. [Available]: https://github.com/TsingHuasuya/DOMI_code
4. Online. [Available]: https://github.com/TsingHuasuya/DOMI_dataset

2 Preliminaries

In this section, we first present the motivation of our work in Section 2.1, and then formulate the problem of *detecting outlier machine instances* in Section 2.2. Lastly, we provide a basic description of 1D-CNN and GMVAE, which are two key components in our model, in Section 2.3.

2.1 Motivation

In large Internet companies, a massive number of machines work as clusters in datacenters to support Internet-based services [1], [28]. These machines process a great number of variable requests, and thus their performance is important to the quality of services (QoS) [4]. However, due to software bugs (e.g., thread crash, memory leaking), hardware failures (e.g., machine aging, hardware breakdown, correctable or uncorrectable hardware errors), malicious attacks, *etc.*, some machines are suffering poor performance and become outliers [3], [29]. In a modern Internet company requiring highly stable computing environments, an outlier machine instance can fail a whole datacenter because of a chain effect and heavily degrade the performance of services [3]. Thus, detecting outlier machine instances so as to mitigate losses is vitally important. In this paper, we mainly focus on detecting outlier machine instances, aiming to find under-performing machine instances for large Internet-based services.

In order to take a global view on the performance of machines and provide highly available services, operation engineers are carefully monitoring various types of machines' metrics (e.g., CPU idle, TCP retransmission rate). Each metric is typically collected in the form of a univariate time series. Therefore, each machine can be denoted as a multivariate time series [6], [7], [13]. The goal of detecting outlier machine instances is to determine whether a machine with multivariate time series becomes an outlier over a long period (say a day in our paper).

Since an outlier machine instance can be caused by various types of reasons, the interpretation of an outlier instance affords operation engineers a deep insight into the outlier detection results. This can facilitate a more rapid root cause analysis process.

2.2 Problem Statement

A *machine instance*, represented as *M-X@D-Y*, is a 1-day-long multivariate time series defined as $\mathbf{x} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ ($\mathbf{x} \in \mathbb{R}^{T \times N}$, where T and N are the number of time points in one day and that of univariate time series, respectively). $\boldsymbol{x}_n = \{x_n^{Y \cdot T}, x_n^{Y \cdot T+1}, \ldots, x_n^{Y \cdot T+T-1}\}$ is a univariate time series on *Day-Y* ($x_n^t \in \mathbb{R}$ is the measurement data of the nth metric at time t), representing the monitoring data of a certain machine metric. Typically, it contains successive observations that are collected with equal intervals [2], and different metrics share the same sampling interval [5]. For example, in our scenario, each machine instance has 19 metrics (i.e., $N = 19$), which are empirically selected by experienced operation engineers based on their domain knowledge. The sampling interval of these metrics is 5 minutes, and thus each day has 288 time points (i.e., $T = 288$). The detailed information of the 19 metrics, which can be classified into five categories, is listed in Table 1. Note that, disk failures can often fail the machines in practice. At ByteDance, the operation engineers designed and ran a S.M.A.R.T-based analysis pipeline (following [30]) to maintain disks' performance specifically. Therefore, from the deployment perspective at ByteDance, this work does not consider the disk-related metrics.

TABLE 1
Detailed Information About the 19 Metrics of Machine Instances Which are Classified into 5 Categories

Metric categories / count of metrics	Metrics
CPU related / 7	CPU idle (rate), CPU sintr (count of soft interrupts cycles), CPU wio (io wait), CPU system (CPU utilization at system level), CPU user (CPU utilization at user level), CPU ctxt (count of context switches), CPU nice (CPU utilization at user level with nice priority)
Memory metrics / 1	Memory utilization
VM related / 2	VM pgfault (page faults in the virtual machine), VM pgmajfault (major faults in the VM)
TCP related / 5	TCP attempt fails, TCP retransmission rate, TCP listen drops, TCP insegs (count of segments received), TCP outsegs (count of segments sent)
Network related / 4	RX bytes (count of received bytes), RX packets (count of received packets), TX bytes (count of transmitted bytes), TX packets (count of transmitted packets)

Detecting outlier machine instances is to discover the unexpected machine instances that significantly deviate from normal patterns, so as to identify the anomalous behaviors or events of machines. That is, it equals to finding the machine instances that are largely different from the majority based on the shapes of multivariate time series. Outlier and anomaly detection for univariate/multivariate time series usually determines whether a univariate time series is anomalous or not at each time point (i.e., x_n^t in $\boldsymbol{x}_n$) [8], [9], [10], [12], or whether a vector of multivariate time series is anomalous or not at each time point (i.e., $[x_1^t, x_2^t, \ldots, x_N^t]$ in $\mathbf{x}$) [5], [6], [13], [14], [15], [16], [17], based on a short period of historical data. However, it is a common practice that an automatically recovered point anomaly [31] may not eventually lead to a failure due to the robustness of cloud systems [7]. Therefore, *detecting outlier machine instances focuses on a collection of anomalies or abnormal segments in a multivariate time series over a long period* [31], i.e., formally, determining whether $\mathbf{x}$ is an outlier.

The *interpretation* of an outlier machine instance can facilitate a more rapid troubleshooting process. Generally, operation engineers usually check which metrics largely deviate from normal patterns when an outlier instance is detected, so as to infer the root cause of this outlier. For example, if the TCP related metrics and network related ones behave more abnormally than others for a long period, we can deduce that the outlier machine instance is suffering from network problems [5]. Therefore, for a detected outlier machine instance $\mathbf{x}$,

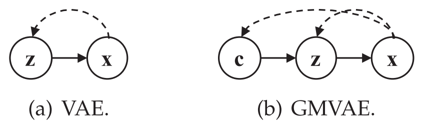


Fig. 2. Graphical models of VAE and GMVAE. $\mathbf{x}$ is input data, $\mathbf{z}$ and $\mathbf{c}$ are stochastic and categorical latent variables, respectively. The solid lines denote generative process (encoder) and the dash lines denote variational approximation (decoder).

we will interpret it by estimating and ranking the contributions of its constituent univariate time series in outlier determination (i.e., x_n in $\mathbf{x}$).

2.3 Basics of 1D-CNN and GMVAE

1D-CNN and GMVAE are two critical components of DOMI, and the basic knowledge about them is described below.

Convolutional Neural Network (CNN) is a supervised model that has three core architecture properties: local receptive fields, shared weights, and spatial or temporal sampling [32]. CNN has achieved huge success in image/video and time series tasks [7], [33], [34] as it is effective in extracting local shape features within the window of convolution. Obviously, for multivariate time series of a machine instance, there exists no strong and clear dependency among different univariate time series. As a consequence, we employ CNN with 1D convolutional kernels that move only in the temporal dimension. The main difference between 1D-CNN and 2D-CNN is that the former one uses 1D arrays rather than 2D matrices for both kernels and feature maps. 1D-CNN has shown its accurate and efficient performance for time series in the aforementioned studies [25], [26]. Thus, DOMI applies 1D-CNN to capture the shape features of multivariate time series data.

Variational autoencoder (VAE) is a popular unsupervised non-temporal dimension reduction technique based on a reconstruction architecture. It is capable of learning the low-dimensional representations of complex data, and it has been demonstrated to have excellent performance on anomaly detection for time series [5], [8], [35]. Specifically, the prior of latent representation or variable (i.e., $\mathbf{z}$ in Fig. 2a) in regular VAE is assumed as a unimodal Gaussian distribution [23], [36]. With a prior for $\mathbf{z}$, the input data $\mathbf{x}$ can be reconstructed by a decoder network $p_\theta(\mathbf{x}|\mathbf{z})$ with parameter θ. The true posterior distribution is intractable to compute, and thus VAE approximates it by an encoder network $q_\phi(\mathbf{z}|\mathbf{x})$ with parameter ϕ. A better $\mathbf{z}$ can facilitate a more accurate reconstruction [5], [23]. However, the prior assumption of regular VAE is too simple to approximate the hidden distributions of complex data [23]. Gaussian mixture VAE (GMVAE) [23], [27], [36] is a variant of VAE model. Its main idea is to use a Gaussian mixture distribution as the prior with a stochastic variable (i.e., $\mathbf{z}$ in Fig. 2b) dependent on a categorical variable (i.e., $\mathbf{c}$ in Fig. 2b). Specifically, its prior is multimodal and thus could learn more complex distributions [36]. GMVAE can be trained by maximizing the evidence of lower bound (ELBO) on the log-likelihood of input data, $\mathcal{L}(\mathbf{x})$:

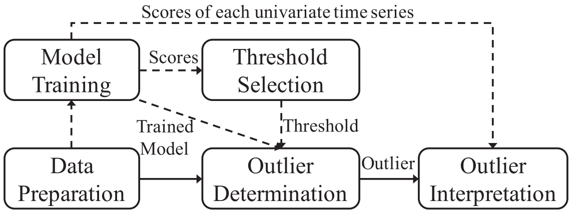


Fig. 3. Framework of DOMI. The dash lines denote offline training and the solid lines represent online detection.

$$\begin{aligned}\log(p_\theta(\mathbf{x})) &\geq \log(p_\theta(\mathbf{x})) - KL[q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})||p_\theta(\mathbf{z},\mathbf{c}|\mathbf{x})] \\ &= \mathcal{L}(\mathbf{x}) \\ &= \sum\nolimits_c \int_z q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x}) log(p_\theta(\mathbf{x}))\mathrm{d}\mathbf{z} \\ &\quad - \sum\nolimits_c \int_z q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x}) log \frac{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{c}|\mathbf{x})}\mathrm{d}\mathbf{z} \\ &= -\sum\nolimits_c \int_z q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x}) log \frac{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}{p_\theta(\mathbf{x},\mathbf{z},\mathbf{c})}\mathrm{d}\mathbf{z} \\ &= \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})} \log\left(\frac{p_\theta(\mathbf{x},\mathbf{z},\mathbf{c})}{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\right),\end{aligned} \tag{1}$$

where KL refers to Kullback-Leibler divergence, and it is used to estimate the difference between two probability distributions (i.e., $KL[q(x)||p(x)] = \int q(x)log\frac{q(x)}{p(x)}\mathrm{d}x$).

3 Design of DOMI

In this section, we first present the overall framework of DOMI in Section 3.1. After that, we introduce the network architecture of DOMI in detail in Section 3.2, followed by the description of offline model training (Section 3.3), online detection (Section 3.4), and outlier machine instance interpretation (Section 3.5).

3.1 DOMI Framework

To solve the problem of detecting outlier machine instances, an unsupervised deep learning method, DOMI, is presented in this work. The framework of DOMI is shown in Fig. 3, which consists of an offline training module and an online detection module. Data Preparation, which is shared by both two modules, standardizes machine instances by z-score normalization. Model Training captures the normal patterns of training dataset (e.g., machine instances of a cluster in the past few weeks). Threshold Selection learns the threshold of outlier instances according to the scores (i.e., reconstruction probabilities) of training dataset. The offline training is conducted periodically (e.g., per week, per day) in order to learn the latest normal patterns, and high-frequency training is not necessary for stable services. Note that when services have major updates or all machines of a datacenter suffer from concept drift at some time, which rarely occurs, we would receive the update message and DOMI will re-conduct offline training using the machine instances after the concept drift. Rapid adaption after concept drift is a challenging job, which is out the scope of this paper.

For a new machine instance $\mathbf{x}$, which consists of the monitoring data of a whole day, DOMI will calculate its score using the trained model. $\mathbf{x}$ will be determined as an outlier machine instance if its score is below the threshold. In practical applications, the metrics (i.e., univariate time series) that cause a machine instance to be an outlier is useful for troubleshooting [5]. Therefore, for a detected outlier machine instance $\mathbf{x}$, DOMI will interpret it by providing a univariate time series list ranked by their contributions (i.e., reconstruction probability changes) in outlier determination. The ranked list will be presented for operation engineers to identify and analyze the root causes of outlier machine instances.

3.2 Network Architecture

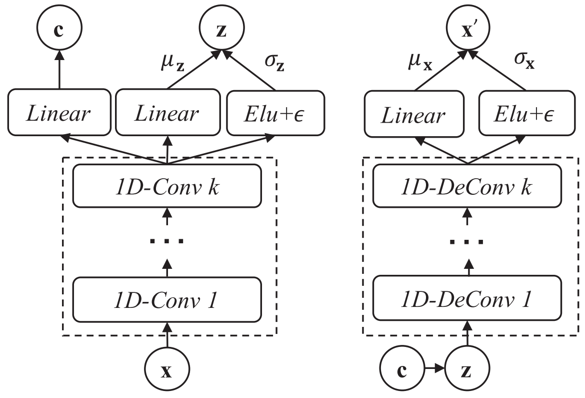
As shown in Fig. 4, DOMI is a reconstruction based model including an inference net (i.e., $q_\phi(\mathbf{z}, \mathbf{c}|\mathbf{x})$) which estimates the posterior probabilities of latent variables, and a generative net (i.e., $p_\theta(\mathbf{x}, \mathbf{z}, \mathbf{c})$) which measures the likelihood of generating data instances given latent variables. In the inference net, DOMI extracts the shape features of input data $\mathbf{x}$ with a convolutional architecture, and models the complex data distributions of $\mathbf{x}$ using a low-dimensional stochastic latent variable $\mathbf{z}$ (i.e., representation, which is a set of probability distributions). The objective of DOMI is to find a good $\mathbf{z}$ of $\mathbf{x}$, because a better $\mathbf{z}$ can facilitate a more accurate reconstruction model [23]. In the generative net, DOMI applies $\mathbf{z}$ to reconstruct $\mathbf{x}$. As there are multiple normal patterns (or classes) of metrics of machine instances in a datacenter, unlike traditional assumption that the prior of $\mathbf{z}$ is a simple Gaussian distribution [8], $\mathbf{z}$ in DOMI depends on a categorical variable $\mathbf{c}$ and the prior of its $\mathbf{z}$ is a Gaussian mixture distribution [27]. The inference net is trained together with the generative net so as to approximate the posterior distributions.

The inference and generative net are formulated in Equations (2) and (3), separately:

$$\mathbf{e}^1 = Elu(w^{e_1} * \mathbf{x} + b^{e_1}) \qquad (2a)$$
$$\mathbf{e}^k = Elu(w^{e_k} * \mathbf{e}^{k-1} + b^{e_k}) \qquad (2b)$$
$$\boldsymbol{\mu}_\mathbf{z} = w^{\boldsymbol{\mu}_\mathbf{z}}\mathbf{e}^k + b^{\boldsymbol{\mu}_\mathbf{z}} \qquad (2c)$$
$$\boldsymbol{\sigma}_\mathbf{z} = Elu(w^{\boldsymbol{\sigma}_\mathbf{z}}\mathbf{e}^k + b^{\boldsymbol{\sigma}_\mathbf{z}}) + \epsilon^{\boldsymbol{\sigma}_\mathbf{z}} \qquad (2d)$$
$$\mathbf{c} = Cat(w^{\mathbf{c}}\mathbf{e}^k + b^{\mathbf{c}}) \qquad (2e)$$
$$\mathbf{z} \sim \mathcal{N}(\mu_\mathbf{z}(\mathbf{c}), \sigma^2_\mathbf{z}(\mathbf{c})), \mathbf{c} \sim Cat(\pi) \qquad (3a)$$
$$\mathbf{d}^1 = Elu(w^{d_1}\mathbf{z} + b^{d_1}) \qquad (3b)$$
$$\mathbf{d}^k = Elu(w^{d_k}\mathbf{d}^{k-1} + b^{d_k}) \qquad (3c)$$
$$\boldsymbol{\mu}_\mathbf{x} = w^{\boldsymbol{\mu}_\mathbf{x}}\mathbf{d}^k + b^{\boldsymbol{\mu}_\mathbf{x}} \qquad (3d)$$
$$\boldsymbol{\sigma}_\mathbf{x} = Elu(w^{\boldsymbol{\sigma}_\mathbf{x}}\mathbf{d}^k + b^{\sigma_\mathbf{x}}) + \epsilon^{\boldsymbol{\sigma}_\mathbf{x}}, \qquad (3e)$$

where $*$ and respectively denote a 1D convolutional and deconvolutional operation, and the convolutional filters move only in the temporal dimension [32]. Elu is an activation function: $Elu(a) = \alpha(exp(a) - 1)|_{a<0} + a|_{a\geq 0}$ (where $\alpha = 1.0$). Cat is a categorical function and $Cat(\pi)$ is a categorical prior distribution (π represents a J-dimensional probability vector, where J is the pre-specified number of components in the Gaussian mixture) [23]. Moreover, all w^*-s, b^*-s are the parameters of the corresponding layers, and ϵ^*-s are small values to prevent numerical overflow [5], [8].

Detailedly, as shown in Fig. 4a, the input of the inference net is $\mathbf{x}$. DOMI first extracts the shape features of each univariate time series of $\mathbf{x}$ using k 1D convolution layers [26]. Then, the extracted shape feature, $\mathbf{e}^k$ in Equation (2), enters the dense layers to generate a categorical variable $\mathbf{c}$ and a



(a) Inference net. (b) Generative net.

Fig. 4. Network architecture of DOMI which is composed of two parts: inference net and generative net. $\mathbf{x}$ is the input data, k is the layers of 1D-convolution and 1D-deconvolution, $\mathbf{x}'$ is the reconstruction output, $\mathbf{z}$ and $\mathbf{c}$ are stochastic and categorical latent variables, respectively.

stochastic variable $\mathbf{z}$ (with mean $\mu_\mathbf{z}$ and standard deviation $\sigma_\mathbf{z}$). In the generative net, as shown in Fig. 4b, DOMI first generates the stochastic variable $\mathbf{z}$ using the categorical variable $\mathbf{c}$ ($\mathbf{c}$ is the component of Gaussian mixture). It then decodes the shape features using the deconvolution layers. Finally, the decoded shape feature, $\mathbf{d}^k$ in Equation (3), enters the dense layers to calculate $\mathbf{x}'$, which is the reconstruction of $\mathbf{x}$, by sampling from $\mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}^2_\mathbf{x}\mathbf{I})$. If $\mathbf{x}$ is an outlier machine instance, it will be difficult to be reconstructed accurately and $\mathbf{x}'$ would differ significantly from the original data $\mathbf{x}$. Therefore, we can apply the reconstruction probability [5], [8], [23] of $\mathbf{x}$ to detect outlier instances.

3.3 Offline Model Training

As mentioned in Section 2.3, GMVAE model is trained to maximize the ELBO (Equation (1)) [27]. Similarly, DOMI can be trained straightforwardly by optimizing the ELBO with Variational Inference for Monte Carlo Objectives (VIMCO) following [37], aiming to optimize the parameters of its network (ϕ and θ). The training objective (i.e., the loss function) of DOMI is formulated as

$$\begin{aligned}\mathcal{L}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z},\mathbf{c})}{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\right)\\ &= \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log\left(\frac{p_\theta(\mathbf{c})p_\theta(\mathbf{z}|\mathbf{c})p_\theta(\mathbf{x}|\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})q_\phi(\mathbf{c}|\mathbf{x})}\right)\\ &= \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log\left(\frac{p_\theta(\mathbf{c})}{q_\phi(\mathbf{c}|\mathbf{x})}\right) + \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log\left(\frac{p_\theta(\mathbf{z}|\mathbf{c})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)\\ &\quad + \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log\left(p_\theta(\mathbf{x}|\mathbf{z})\right),\end{aligned} \qquad (4)$$

with the mean field approximation (i.e., $\mathbf{z}$ and $\mathbf{c}$ can be partitioned and are independent) [23]: $q_\phi(\mathbf{z}, \mathbf{c}|\mathbf{x}) = q_\phi(\mathbf{z}|\mathbf{c}, \mathbf{x})q_\phi(\mathbf{c}|\mathbf{x}) = q_\phi(\mathbf{z}|\mathbf{x})q_\phi(\mathbf{c}|\mathbf{x})$. $q_\phi(\mathbf{z}|\mathbf{x})$ is obtained from the inference net, and it is assumed as a multivariate Gaussian distribution [27]: $q_\phi(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\sigma}^2_\mathbf{z}\mathbf{I})$, which aims to approximate the true posterior distribution of $\mathbf{z}$. $q_\phi(\mathbf{c} = i|\mathbf{x}) = \frac{q_\phi(\mathbf{x}|\mathbf{c}=i)q_\phi(\mathbf{c}=i)}{\sum_{j=1}^J q_\phi(\mathbf{x}|\mathbf{c}=j)q_\phi(\mathbf{c}=j)}$, where $q_\phi(\mathbf{x}|\mathbf{c} = i)$ is the probability of generating $\mathbf{x}$ using the ith Gaussian component, and $q_\phi(\mathbf{c})$ could be taken as $\mathbf{c}$-prior (i.e., $p_\theta(\mathbf{c})$).

The first and second terms in Equation (4) are the regularization (i.e., Kullback-Leibler loss) on $\mathbf{c}$ and $\mathbf{z}$, respectively. The goal is to minimize the difference between the posterior distributions ($q_\phi(\mathbf{c}|\mathbf{x})$ and $q_\phi(\mathbf{z}|\mathbf{x})$) and the prior distributions ($p_\theta(\mathbf{c})$ and $p_\theta(\mathbf{z}|\mathbf{c})$). $p_\theta(\mathbf{c}) \sim Cat(\pi)$ follows a categorical prior distribution. $\mathbf{z}$ follows a Gaussian mixture distribution and can be sampled on $p_\theta(\mathbf{z}|\mathbf{c}) \sim \mathcal{N}(\mu_\mathbf{z}(\mathbf{c}), \sigma^2_\mathbf{z}(\mathbf{c}))$ [23]. The third term is the reconstruction probability which aims to maximize the likelihood of $\mathbf{x}$, and it can be calculated as: $p_\theta(\mathbf{x}|\mathbf{z}) \sim \mathcal{N}(\mu_\mathbf{x}, \sigma^2_\mathbf{x}\mathbf{I})$.

Traditionally, for GMVAE based models, during model training, the single sample Monte Carlo estimate of the ELBO is commonly used to approximate the training loss [38], [39]. To obtain a more reliable and stable representation, in this paper, we design a simple yet effective approximation method, multi-sample Monte Carlo integration to approximate the $\mathcal{L}(\mathbf{x})$ in Equation (4):

$$\begin{aligned}\mathcal{L}(\mathbf{x}) \approx \frac{1}{L_1 \cdot L_2}\sum_{l_1=1}^{L_1}\sum_{l_2=1}^{L_2}\Bigg[\log\left(\frac{p_\theta(\mathbf{c}^{(l_2)})}{q_\phi(\mathbf{c}^{(l_2)}|\mathbf{x})}\right) \\ + \log\left(\frac{p_\theta(\mathbf{z}^{(l_1)}|\mathbf{c}^{(l_2)})}{q_\phi(\mathbf{z}^{(l_1)}|\mathbf{x})}\right) + \log\left(p_\theta(\mathbf{x}|\mathbf{z}^{(l_1)})\right)\Bigg],\end{aligned} \tag{5}$$

where $\mathbf{z}^{(l_1)}, l_1 = 1, 2, \ldots, L_1$ (L_1 is the sample size of $\mathbf{z}$) is sampled from $q_\phi(\mathbf{z}|\mathbf{x})$ and $\mathbf{c}^{(l_2)}, l_2 = 1, 2, \ldots, L_2$ (L_2 is the sample size of $\mathbf{c}$) is sampled from $q_\phi(\mathbf{c}|\mathbf{x})$.

During model training, $\boldsymbol{\sigma}_\mathbf{x}$ and $\boldsymbol{\sigma}_\mathbf{z}$ can be very small (i.e., very close to zero), which may lead to numerical problems when calculating the likelihoods of Gaussian variables. To solve this problem, we adopt the 'ϵ' trick in [8]. When backpropagating gradients through network layers, model parameters will result in NaN values (i.e., overflow) once the gradient value grows extremely large. We use gradient clipping [40] to deal with 'gradient explosion', with a limit of 10.0. Moreover, there could exist some outlier instances in training data, and we thus apply early-stopping to avoid over-fitting.

For unsupervised techniques, they are trained to capture the intrinsic properties within a dataset and then detect outliers based on the data characteristics [31]. In our model, DOMI captures the normal patterns of machine instances from the majority of training dataset in an unsupervised way [5], [8]. Typically, the majority of machine instances in the training dataset (e.g., 81 percent in our scenario) are normal. Therefore, the learning process is dominated by normal instances, and DOMI is robust to a small portion of outlier instances in the training dataset.

3.4 Online Detection

After offline training, DOMI has learned the parameters of network so as to capture the normal patterns of machine instances. Now we can determine whether an input machine instance $\mathbf{x}$ is an outlier or not using the trained model. The reconstruction probability of $\mathbf{x}$ in DOMI indicates its reconstruction difficulty [23]: the lower of its value, the harder of $\mathbf{x}$ to be accurately reconstructed. For an outlier instance, it will be hard to be accurately reconstructed and thus its reconstruction probability tends to be lower than that of normal instances [5]. Inspired by [8], in DOMI, we formally use the reconstruction probability of $\mathbf{x}$ as the outlier score: $S = \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log(p_\theta(\mathbf{x}|\mathbf{z}))$. For $\mathbf{x}$, if its outlier score S is below a threshold *TH*, it will be determined as an outlier machine instance; otherwise, it is normal. Detecting outlier machine instances is critical for operation engineers to find the underperforming machine instances and timely fix them.

In DOMI, an outlier score is a continuous value that quantifies the level of "outlierness" of a machine instance. Considering a large number of machine instances (e.g., hundreds of thousands per day in our scenario), DOMI learns the threshold *TH* automatically, which converts an outlier score into a binary result (outlier or not). As aforementioned, $\mathbf{x}$ with a lower score is more likely to be an outlier. Peaks-Over-Threshold (POT), the second theorem in Extreme Value Theory (EVT), has demonstrated its excellent performance in tuning thresholds [5]. Therefore, we apply the adjusted POT method in [5] to automatically tune DOMI's threshold *TH*.

3.5 Outlier Machine Instance Interpretation

The interpretation is to answer the question of why a machine instance becomes an outlier. The outlier score S indicates the overall performance of the whole $\mathbf{x}$. In practice, the univariate time series (i.e., $x_n \in \mathbf{x}$, where $1 \le n \le N$) that cause $\mathbf{x}$ to be an outlier instance are important for operation engineers to analyze the outlier and troubleshoot the root causes [5]. To interpret a detected outlier instance $\mathbf{x}$, we calculate the contribution (i.e., reconstruction probability) of each x_n in outlier determination, and x_n with larger contribution is more likely to infer the root cause.

According to $p_\theta(\mathbf{x}|\mathbf{z}) \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}^2_\mathbf{x}\mathbf{I})$, we can get $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{n=1}^N p_\theta(x_n|\mathbf{z})$ and $\log(p_\theta(\mathbf{x}|\mathbf{z})) = \sum_{n=1}^N \log(p_\theta(x_n|\mathbf{z}))$. That is, $S = \sum_{n=1}^N S_n$, where $S_n = \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{c}|\mathbf{x})}\log(p_\theta(x_n|\mathbf{z}))$ and it is the outlier score (i.e., reconstruction probability) of x_n. The reconstruction *difficulties* of x's different univariate time series are usually different, thus using the raw scores of univariate time series [5] is not proper. Consequently, we apply the score changes (i.e., reconstruction probability changes) of each univariate time series to interpret the detection result. Specifically, we first calculate the expected score of each univariate time series in the training dataset: $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_N$, where $\bar{S}_n$ is the expected score of x_n. Then we calculate the score changes of each univariate time series x_n: $\Delta S_n = S_n - \bar{S}_n$. Finally, for an outlier machine instance x, we rank the score changes of its N univariate time series in ascending order to form the contribution list $\boldsymbol{\Delta}\mathbf{S}$. For x_n, the higher it ranks in $\boldsymbol{\Delta}\mathbf{S}$, the larger of its score changes, the greater it contributes to outlier determination.

The top few univariate time series in $\boldsymbol{\Delta}\mathbf{S}$ can provide clues for operation engineers to understand the outlier machine instances and pinpoint the root cause. Moreover, our interpretation method is also applicable to other VAE-based methods such as Donut [8] and OmniAnomaly [5].

4 Evaluation

In this section, we first introduce the experiment setup including dataset, performance metrics and the hyper-parameters of DOMI in Section 4.1. We then compare the performance of DOMI with baseline methods in Section 4.2, followed by the evaluation of GM prior, multi-sample Monte Carlo integration and 1D convolutional kernels in Section 4.3. After that, we show the performance of interpretation in Section 4.4 and DOMI's efficiency in Section 4.5. Finally, we discuss the impact of hyper-parameters in Section 4.6.

4.1 Experiment Setup

Dataset. To demonstrate the effectiveness of DOMI, all experiments are conducted on a machine dataset collected from ByteDance, a top global content service provider. It is collected from 1821 machines in a datacenter with a 45-day-period, which provides services for four short video related applications, leading to four classes of machines. The applications have billions of daily active users and remain relatively stable with minor updates for sixth months, and thus the studied datacenter is considered stable over a long period. For each machine, the monitoring data of 19 metrics (see Table 1 for more details) is collected every five minutes. The amount of training data is determined by the length of data and the number of machines. Thus, we have 81945 (=1821 × 45) machine instances in total, where each instance is a 19 × 288 (=24 × 60/5) matrix. We divide the dataset into two parts: the former 30 days for training and the latter 15 days for testing. The ground truth has been labeled by experienced operation engineers who maintain these machines. The ratios of outlier instances in the training and testing sets are 0.19 and 0.22, respectively.

Performance Metrics. We mainly employ two performance metrics to evaluate DOMI and other baseline methods: AUC and the best F1-Score (denoted as F1_{best}). (1) Area Under the ROC Curve (AUC) indicates the overall performance by enumerating all possible thresholds. For ROC (receiver operating characteristic) curve, the closer to the upper left, the larger of AUC, the better performance of its algorithm. (2) $\text{F1}_{best} = \frac{2\times Precision\times Recall}{Precision+Recall}$, where $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, indicates the best possible performance of models with an optimal global threshold. F1_{best} and AUC are performance indicators that balance precision and recall, and thus they are more comprehensive in evaluating methods' performance than precision or recall. F1_{best} is consistent with AUC [8] and can be selected from the Precision-Recall Curve (PRC). The PRC close to upper right performs better than one close to bottom left. All evaluation experiments are one-off efforts, and we repeated them 10 times using the same hyper-parameters to calculate the average AUC and F1_{best}.

Hyper-Parameters. We set the hyper-parameters of DOMI empirically in our experiments as follows. Both the convolutional and deconvolutional architectures have four layers, the kernel sizes and strides of which are {12×1, 12×1, 6×1, 6×1} and {4×1, 4×1, 3×1, 3×1}, respectively. We apply L2 regularization with a coefficient of 10^{-4} to all layers. The dimension of $\mathbf{z}$-space variables is 10 and the number of components of $\mathbf{c}$ is 4. ϵ is set to 10^{-10}. The sample size L_1 and L_2 are both 500. For the adjusted POT method, the low quantile = 0.2 and $q = 10^{-4}$ [5]. 90 percent of the training set is used for training and the remaining 10 percent is used for validation. DOMI is trained with Adam optimizer [41], which is reduced by a factor of 0.5 every 5 epochs, with an initial learning rate of 10^{-3}. We run 10 epochs for training with early stopping, and the batch size is 32. More studies about the choice for hyper-parameters are discussed in Section 4.6.

TABLE 2
Average AUC and F1_{best} of DOMI and Baseline Methods

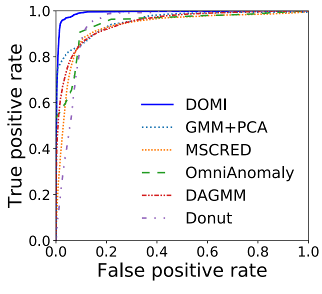
Methods	Precision	Recall	F1_{best}	AUC
DOMI	**0.9378**	**0.9418**	**0.9398**	**0.9921**
Donut	0.6995	0.9331	0.7995	0.9217
OmniAnomaly	0.6351	0.9241	0.7527	0.9214
DAGMM	0.8279	0.8031	0.8152	**0.9641**
MSCRED	0.7239	0.8338	0.7749	0.9293
GMM	0.7019	0.8371	0.7636	0.9307
GMM+PCA	0.8750	0.7933	0.8321	**0.9562**
MDDTW+DBSCAN	0.8069	0.9087	**0.8542**	N/A
DCN	0.8778	0.7953	**0.8323**	N/A
IForest	**0.9893**	0.3954	0.5650	N/A
IForest+PCA	**0.8868**	0.5157	0.6520	N/A
OCSVM	0.3936	**0.9387**	0.5548	N/A
OCSVM+PCA	0.3948	**0.9427**	0.5565	N/A

The AUC of clustering and classification based methods are not available because they have no thresholds ("N/A" denotes no results and the top 3 metrics are highlighted in bold).

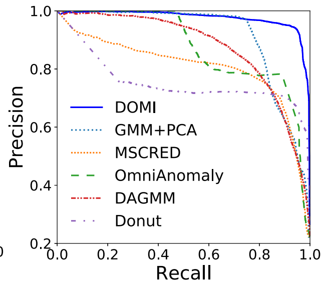
4.2 DOMI Versus Baseline Methods

To show the effectiveness of DOMI, we carefully choose the following state-of-the-art outlier and anomaly detection methods as baselines: Donut [8], OmniAnomaly [5], DAGMM [21], MSCRED [7], GMM [21], MDDTW+DBSCAN [42], DCN [43], IForest [44], OCSVM [3] (see Section 6 for more details). For the methods that cannot handle matrix data, we flatten the input matrix to vector following [41]. Moreover, to deal with the high dimensionality of input data, the density estimation and classification based methods (e.g., GMM, IForest, OCSVM) are improved by a two-stage variant [21]: (1) reduce the dimensionality of input data by PCA (2) apply the density estimation and classification based methods on the compressed data. In addition, to make OmniAnomaly work for detecting outlier machine instances, we adopt an intuitive rule to define outlier instances that *"at least T' in T time points are detected as anomalies (where T' can be tuned for best result)"*.

Table 2 lists the average precision, recall, F1_{best}, and AUC of DOMI and baseline methods. The ROC curves and PR curves shown in Fig. 5 provides more details about the performance of these algorithms. The TP/FP/TN/FN of DOMI are TP = 5726, FP = 380, FN = 354, TN = 20855, respectively. The instances having anomalies lasting short with large fluctuations are more likely to be FPs, and those having anomalies lasting long with minor fluctuations tend to be FNs. Overall, none of the baselines can extract shape features, and DOMI outperforms them because it extracts the shape features of multivariate time series with 1D-CNN



(a) ROC curves



(b) PR curves

Fig. 5. ROC curves and PR curves of DOMI and baseline models (Note that the clustering and classification based methods are not available here. Moreover, for GMM, we only select its variant GMM+PCA).

[7], [26] and successfully captures the normal patterns by GMVAE. Detailedly, DOMI exceeds the best performing baseline method by 0.03 on the AUC (i.e., DAGMM) and 0.08 on the $F1_{best}$ (i.e., MDDTW+DBSCAN, whose TP/FP/TN/FN are TP = 5526, FP = 1322, FN = 554, TN = 19913, respectively). That is, MDDTW+DBSCAN generates 942 more FPs and 200 more FNs comparing to DOMI. The large number of FPs will waste operators much more time on checking false alarms, and the large number of FNs may cause operators to falsely ignore outlier instances and in turn lead to bad user experience. Next, we will analyze these methods in detail.

Extracting Shape Features is Important. For MSCRED, its inputs are signature matrices constructed by multivariate time series instead of raw values, which concern about the shape similarity among different pairs of *univariate* time series, rather than the overall shapes of multivariate time series. For "MDDTW+DBSCAN", MDDTW calculates pairwise distances among instances based on their shape similarity, rather than extracts the shape features. It performs the best among all baselines, which further proves the importance of considering shape characteristics of time series. However, its computational complexity is as high as $O(n^2)$, where n is the number of instances. Applying it for detecting outlier machine instances will cost lots of computational resources and is much time-consuming.

Dimension Reduction Should be Done in an End-to-End Manner. Neither GMM nor IForest achieves comparable performance to their variants (i.e., combine them with PCA), which demonstrates the importance of dimension reduction. However, PCA can not greatly improve the performance of OCSVM. This could be because that, for two-stage models, dimension reduction in the first step is unaware of the follow-ups, and the key information of data could be lost during dimension reduction and in turn lead to suboptimal performance [21]. Different from the two-stage models, DOMI is an end-to-end model that combines dimension reduction and pattern learning simultaneously to achieve optimal performance.

VAE-Based Reconstruction is More Robust. Compared with the AE-based reconstruction methods (e.g., DAGMM and MSCRED), the VAE-based reconstruction method, DOMI, infers the variability of distributions in the latent variables using variational inference [35], which has been proved to be more principled [23]. Although Donut is based on VAE, its prior is a simple Gaussian distribution, which degrades its performance for multivariate time series with diverse types of distributions [22]. The Gaussian mixture prior of DOMI enables its latent representations to learn the complex distributions of data well.

Anomaly Detection for Multivariate Time Series is not Proper for the Problem of Detecting Outlier Machine Instances. OmniAnomaly detects anomalies for multivariate time series at each time point, which mainly focuses on the starting point rather than the entire segment of long-lasting anomalies [5]. Moreover, even if it could be applied for outlier machine instance detection, the number of anomalous time points (e.g., T' in our case) is almost infeasible to be accurately predefined in practice.

A Reasonable Latent Representation is Needed. For the density estimation method (e.g., GMM), it has been demonstrated to be very difficult to perform reasonable density estimation for complex data [21], and thus it suffers poor performance in our scenario. In addition, for a well-trained AE, there may exist no significant differences among the reduced dimensions, and thus the clustering method of DCN cannot distinguish them accurately. On the contrary, the well-learned latent representations in DOMI have been demonstrated to be intuitive and reasonable, so that it achieves good performance (will discuss later in Section 5.2).

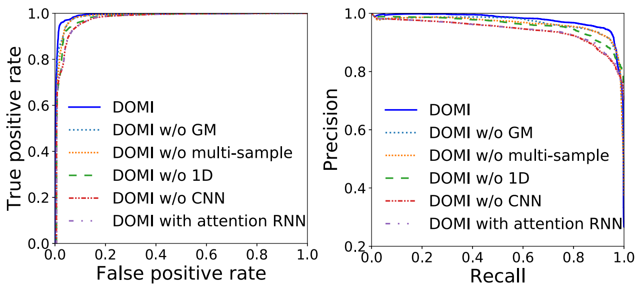
TABLE 3
Average AUC and $F1_{best}$ of DOMI and its Variants

Methods	Precision	Recall	$F1_{best}$	AUC
DOMI	0.9378	0.9418	0.9398	0.9921
DOMI w/o GM	0.9053	0.9336	0.9192	0.9829
DOMI w/o multi-sample	0.9057	0.9316	0.9184	0.9826
DOMI w/o 1D	0.9283	0.8648	0.8954	0.9786
DOMI w/o CNN	0.9102	0.8535	0.8809	0.9753
DOMI with attention RNN	0.9222	0.8523	0.8858	0.9760

4.3 Evaluation of GM Prior, Multi-Sample Monte Carlo Integration and 1D Convolutional Kernels

There are three major techniques in DOMI: GM prior, multi-sample Monte Carlo integration and 1D convolutional kernels. To evaluate their performance, we design five variants of DOMI: (1) "DOMI without (w/o) GM" denotes that the Gaussian mixture prior in DOMI is replaced by a simple Gaussian prior. (2) "DOMI w/o multi-sample" means that the approximation method is replaced by the single sample Monte Carlo integration. (3) "DOMI w/o 1D" represents that the 1D-CNN is replaced by 2D-CNN. (4) "DOMI w/o CNN" indicates that the convolutional architecture is replaced by full connected layers. (5) "DOMI with attention RNN" represents that the 1D-CNN is replaced by an attention based RNN.

From Table 3 and Fig. 6, we observe that "DOMI w/o GM" does not perform as well as DOMI. This could be explained by the fact that the prior with the Gaussian mixture distribution, which is more complex than the simple Gaussian distribution, can better model high dimensional data [23], [27]. Without the multi-sample Monte Carlo approximation, one sample of $\mathbf{c}$ is not sufficient for computing the ELBO and the value of $\mathbf{c}$ is neither reliable nor stable. Additionally, unlike the matrix data of images, there exists no clear dependency among different univariate time series. Therefore, 1D-CNN is more intuitive than 2D-CNN for shape feature extraction on



(a) ROC curves (b) PR curves

Fig. 6. ROC curves and PR curves of DOMI and its variants.

multivariate time series, and replacing 1D-CNN with 2D-CNN degrades the performance of DOMI. Besides, compared with 2D-CNN [25], 1D-CNN has simpler architecture and lower computational complexity, which makes the model easier to be trained [26]. "DOMI with attention RNN" leads to worse performance compared to DOMI. It is because attention-based RNNs only capture the temporal features of time series, while 1D-CNN can well capture both temporal and shape features. Lastly, the improvement of DOMI and "DOMI w/o 1D" over "DOMI w/o CNN" and "DOMI with attention RNN" further proves the necessity of shape feature extraction on time series [7], [26].

In real-world scenarios, it is necessary to automatically tune the threshold for outlier machine instances. Here we show the performance of the adjusted POT method [5] for automatic threshold tuning in DOMI. The F1-Score obtained through the adjusted POT is 0.9395, which is only slightly lower than F1_{best} (0.0003), indicating that the adjusted POT method is practical for threshold tuning in DOMI.

4.4 Evaluation of Interpretation

Interpretation of the detected outlier machine instances is important for troubleshooting. To achieve this goal, DOMI applies score changes (i.e., reconstruction probability changes) rather than raw scores of univariate time series for interpretation, because the reconstruction difficulty varies among different univariate time series. For example, the normal patterns of the 5th metric in Fig. 1 are more difficult to be learned than the other metrics because it is variable in nature. Specifically, its expected score $\bar{S}_5 \approx 1$, which is much lower than others (about 5 to 30).

We apply HitRate@P% (P can be 100 or 120) to evaluate the interpretation accuracy using raw scores and score changes, respectively. HitRate@P% indicates the overlapping ratio between **GT** (ground truth list with the metrics that indeed lead to outlier instances) and the top $\lfloor P\% \times |\mathbf{GT}| \rfloor$ metrics in list $\mathbf{\Delta S}$ provided by DOMI, where $|\mathbf{GT}|$ is the length of **GT**. Here we give a toy example to understand HitRate@P%. For an outlier machine instance **x** with 10 univariate time series (i.e., $N = 10$), its $\mathbf{\Delta S}$ is {2, 7, 3, 6, 8, 9, 1, 5, 10, 4} (the number 1 to 10 means the names of 10 univariate time series) and **GT** is {2, 3, 6, 9, 10}. Therefore, HitRate@100% = 0.6 and HitRate@120% = 0.8.

From Table 4, we observe that DOMI using score changes provides better interpretation results for outlier instances than that using raw scores. Moreover, the average interpretation accuracy of DOMI is higher than Donut and OmniAnomaly, because its **z**-space representations better capture the shape features and complex distributions of multivariate time series. The experiments also show that applying score changes brings better performance to Donut and OmniAnomaly than using raw scores, demonstrating that this idea can be applied to other VAE-based anomaly/outlier detection methods beyond DOMI.

4.5 Evaluation of Computational Efficiency

To demonstrate the computational efficiency of DOMI, we test it on two different servers: (a) a CPU server with (24-core Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz with 64 GB RAM) and (b) a GPU server: Nvidia GeForce RTX 2080-Ti. The average execution time for offline training and online testing is shown in Table 5. Specifically, each step in the model training procedure costs 0.09 and 0.04 seconds on the CPU server and GPU server, respectively. Because each epoch has about 1,500 steps for our training set, it consumes about two minutes and one minute on the CPU server and GPU server, respectively. Since we run 10 epochs for the whole training set, the total training time is less than half an hour. In the online testing procedure, we detect the instances in a batch fashion and measure the average testing time per instance (i.e., testing time per instance = testing time for a batch/batch size). It takes 2.7 and 1.1 milliseconds for the CPU server and GPU server to test a machine instance, respectively. That is, DOMI can detect outliers for ten thousand machine instances every 30 seconds using either a CPU server or a GPU server. The computational complexity of DOMI is $O(mn)$, where m is the training epoch and n is the number of instances. Note that, the testing time (i.e., the inference time) does not cover the data processing procedure including data aggregation and data sending. The data processing time is about 3.1 milliseconds per instance. Based on on-site interviews with operation engineers, they are quite satisfied with DOMI's computational efficiency.

TABLE 4
Average Interpretation Accuracy of Two Interpretation Methods for DOMI and Two VAE-Based Models (Donut and OmniAnomaly)

Interpretation methods	Raw scores		Score changes	
	HitRate @100%	HitRate @120%	HitRate @100%	HitRate @120%
DOMI	0.6534	0.8001	0.8953	0.9302
Donut	0.5163	0.7220	0.8039	0.9079
OmniAnomaly	0.6717	0.8080	0.8625	0.9130

4.6 Evaluation of Hyper-Parameters

In this section, we mainly discuss three hyper-parameters of DOMI that can significantly impact its performance: the dimension of hidden stochastic variables (i.e., **z**), the number of components of Gaussian mixture (i.e., **c**), and the number of epochs.

z and **c** are two main components of DOMI. A larger **z** can reduce the effect of dimension reduction, while a smaller **z** can lead the model to be under-fitting [8]. Moreover, a smaller **c** can reduce the complexity of the distributions of **z**, degrading the performance of **z** in modeling high dimensional data. It has been demonstrated that a model's performance becomes steady when **c** is large enough [45].

TABLE 5
Average Execution Time (Seconds) of DOMI on CPU and GPU, Respectively

	CPU	GPU
Training time per step	0.0892	0.0370
Training time per epoch	137.2048	57.7789
Testing time per instance	0.0027	0.0011

Each epoch has about 1500 steps for training.

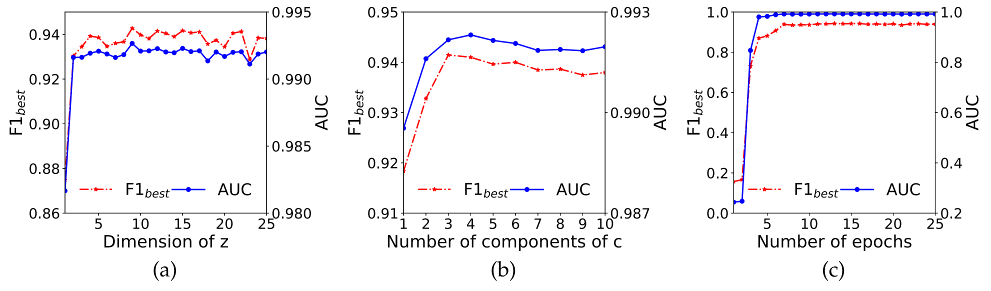


Fig. 7. The average F1$_{best}$ and AUC results of DOMI by varying three hyper-parameters.

Therefore, we first empirically set the number of components of $\mathbf{c}$ as a large value (e.g., 6) to show the impact of $\mathbf{z}$'s scale to DOMI's performance. From Fig. 7a, we can see that both F1$_{best}$ and AUC achieve the best result when the dimension of $\mathbf{z}$ is 10, and they become steady when the dimension of $\mathbf{z}$ is larger than 3. It demonstrates that DOMI is quite robust to the dimension of $\mathbf{z}$. Therefore, we set this hyper-parameter as 10. Fig. 7b shows the average F1$_{best}$ and AUC of DOMI as a function of the number of components of $\mathbf{c}$. Obviously, DOMI's performance becomes stable when it is larger than 3, demonstrating that DOMI is also robust to this hyper-parameter. We thus set it as 4 in the evaluation experiments.

The number of epochs, i.e., the number of complete passes through the training set, is one of the hyper-parameters of gradient descent. A smaller number of epochs can prevent a model from being fully trained. However, a larger number of epochs can bring larger computational complexity. Fig. 7c shows the average F1$_{best}$ and AUC of DOMI as the number of epochs varies. Similarly, we can see that when the number of epochs is larger than 7, the performance of DOMI becomes steady, and thus DOMI is robust to this number. Therefore, we set it as 10 in the evaluation experiments to ensure more robust performance, and its small value indicates that DOMI can be well trained within a short time.

Overall, DOMI is robust to the three hyper-parameters, and thus operation engineers do not have to worry about how to tune their values. Moreover, we also provide guidance of tuning hyper-parameters: the larger number of dimensions of input data, the larger of $\mathbf{z}$; the more classes of machines, the larger of $\mathbf{c}$; the smaller of the training instances, the larger number of the epoch. Note that tuning hyper-parameters automatically for DOMI is difficult and out the scope of this paper.

5 Discussion

In this section, we first discuss the model fit of DOMI in Section 5.1, and then explain how DOMI works for detecting outlier machine instances through visualizing $\mathbf{z}$-space representations in Section 5.2. After that, we present the deployment architecture and three case studies of DOMI in ByteDance in Section 5.3. Finally, we conclude some lessons learned from DOMI in Section 5.4.

5.1 Model Fit

It is necessary for a deep learning model to be appropriately fit in the training procedure, the goal of which is to capture general patterns of training data and not simply memorize the dataset. A deep learning model can be under-fitting when the loss of training set $\ll$ the loss of validation set. On the contrary, it is over-fitting if the loss of training set $\gg$ the loss of validation set. Fig. 8 shows the loss curves of DOMI on the training set and validation set in the training procedure, respectively. We can see that, when the number of steps is smaller than about 4,000, both the two loss curves keep decreasing, which means DOMI is learning patterns in the training set and generalizing better but not well enough on the validation set. After that, the loss of DOMI on both training set and validation set becomes stable, which not only proves that DOMI converges well on the dataset and successfully learns a generalized model, but also indicates that DOMI indeed maximizes ELBO during model training [11] and the training data is sufficient. In addition, the loss on the training set and validation set is roughly equal for each step, intuitively demonstrating that DOMI is an appropriate-fit model.

5.2 Visualization of z-Space Representations

Here we try to demonstrate how DOMI works by analyzing its latent representations. As aforementioned, DOMI is a reconstruction based method. In the training procedure, it captures the normal patterns of training data by learning their low-dimensional representations. Specifically, for a machine instance $\mathbf{x}$, DOMI first compresses it to a low-dimensional representation $\mathbf{z}$, and then reconstructs it (i.e., $\mathbf{x}'$) based on the learned $\mathbf{z}$. Whether $\mathbf{x}$ is an outlier or not, this learned $\mathbf{z}$ is as "normal" as possible [5], [8]. Therefore, for an outlier $\mathbf{x}$, its $\mathbf{x}'$ is greatly different from it and its reconstruction probability is thus low, based on which DOMI can successfully perform the outlier detection for machine instances. In Fig. 9, we visualize the 3-dimensional $\mathbf{z}$-space representations of the testing set. All representations are sampled from $q_\phi(\mathbf{z}, \mathbf{c}|\mathbf{x})$. As shown in Fig. 9a, the $\mathbf{z}$ representations of normal and outlier instances generated by DOMI are "overlapped" and appeared to be identical, demonstrating that they share very similar distributions. That is, an outlier $\mathbf{x}$'s $\mathbf{z}$ and $\mathbf{x}'$ are "normal", and its reconstruction probability is thus low.

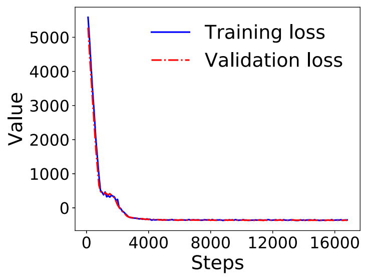


Fig. 8. The loss curves of DOMI on the training set and validation set.

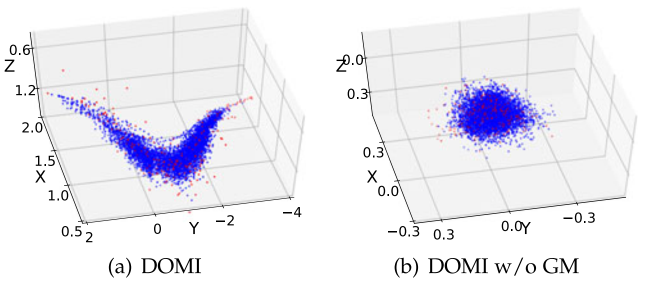

(a) DOMI (b) DOMI w/o GM

Fig. 9. 3-dimensional $\mathbf{z}$-space representations of (a) DOMI and (b) "DOMI w/o GM", where red and blue points represent outlier and normal instances, respectively.

To further understand the $\mathbf{z}$-space representations of DOMI, we compare them with the $\mathbf{z}$ representations of "DOMI w/o GM" as shown in Fig. 9b. Obviously, the distributions of DOMI's $\mathbf{z}$ are more complex than those of "DOMI w/o GM"'s. That is because the Gaussian mixture prior of $\mathbf{z}$ in DOMI is more complex than the simple Gaussian prior in "DOMI w/o GM". This further explains why DOMI performs better than "DOMI w/o GM": the isotropic distribution of simple Gaussian fails to model the intrinsic multimodality of high dimensional data [22].

5.3 Deployment and Case Study

In ByteDance, the System Technologies and Engineering (STE) team is responsible for the reliability of machines. To help the operation engineers in STE find the under-performing and unmanaged machine instances, we implemented the prototype of DOMI in Python v3.6 and Tensorflow v1.14.0. DOMI has been deployed in 19 datacenters housing 90,000+ machines in ByteDance for more than 4 months, and it has successfully detected about 450 outlier machine instances per day. These datacenters are with diverse types of workloads. DOMI achieves an excellent precision of 0.91, demonstrating that it is generic enough to be applied to other datacenters. We acknowledge that it is difficult to obtain the recall of DOMI because labeling a large number of false negatives, which is key to calculate recall, requires very significant human efforts and is impractical.

Fig. 10 presents the deployment framework of DOMI. The machines support online services in the companies. Their metrics are carefully being monitored and collected as the format of multivariate time series, then processed and stored by Kafka in a streaming manner, and created a data API for easy access. We train one DOMI model for each datacenter, which runs a variety of services and houses thousands of machines with multiple classes. For different datacenters, the hyper-parameters of DOMI were set the same without any tuning. Moreover, to avoid the negative impact of outlier instances during model training, we adopt an outlier skipping strategy, that first recorded the outlier instances detected by the trained DOMI model and then retrained a new model based on the dataset with these outlier instances being filtered out. During online detection, operation engineers would receive the detected outlier results with their interpretations via emails, and can see them on a visualization platform. In Fig. 11, we present three real-world cases that happened in ByteDance.

In Case A, as shown in Fig. 11a, the machine instance *machine-A@13-March-2020* was detected by DOMI as an

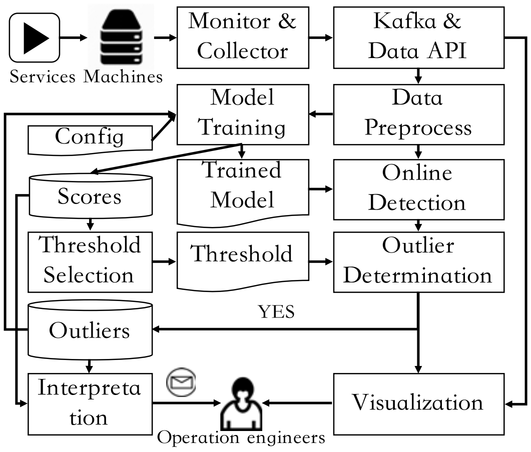
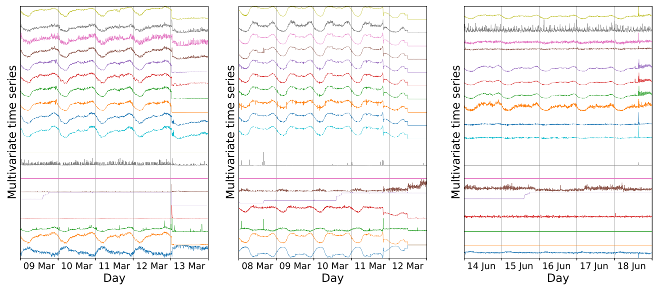


Fig. 10. Deployment framework of DOMI.

outlier. Its score was -90.82 (while the scores of other machine instances from 09-March-2020 to 12-March-2020 ranged from 150 to 200), which was lower than the threshold 117.05 that was automatically selected by the adjusted POT method. The top 5 metrics in $\Delta\mathbf{S}$ provided by DOMI were {CPU wio, CPU sintr, CPU idle, Memory utilization, CPU ctxt}, which were mainly CPU and Memory related metrics. More specifically, compared with the top 5 metrics, both Network related metrics and TCP related ones behaved more normal and they ranked latter in $\Delta\mathbf{S}$. After investigating this case, operation engineers found that an uncorrectable ECC memory error happened on that machine at 13-March-2020. The $\Delta\mathbf{S}$ of DOMI provided operation engineers clues to analyze this accident.

In Case B, as shown in Fig. 11b, the machine instance *machine-B@12-March-2020* was detected as an outlier with the score of -585.21. Moreover, mostly all metrics were abnormal based on $\Delta\mathbf{S}$. Operation engineers found, based on investigations, that no services were running on this machine and it became unmanaged during that period. DOMI successfully detected this machine instance because it believed that the metric patterns of this instance deviated from the normal patterns for a long period.

In Case C, as shown in Fig. 11c, the machine instance *machine-C@18-June-2020* was detected as an outlier with the score of -81.32. Detailedly, the TCP and network related metrics were abnormal based on $\Delta\mathbf{S}$, indicating that the machine was suffering a network problem. The ground truth was that there occurred network congestion at switches at 18-June-2020. The $\Delta\mathbf{S}$ provided by DOMI helped operation engineers troubleshoot this failure more quickly.



(a) Case A (b) Case B (c) Case C

Fig. 11. Three real-world cases about outlier machine instances detected by DOMI in ByteDance.

From these three cases, we learn that DOMI could be applied to detect outlier machine instances for real-world services, and its interpretation can provide operation engineers a deeper insight into the root causes of outlier instances.

5.4 Lessons Learned

In this section, we show the following five lessons learned from DOMI. (1) Compared with 2D-CNN, 1D-CNN is more effective in capturing the shape features of multivariate time series because of its 1D convolutional kernels and feature maps, which also reduces computational complexity for model training and makes the model easier to be trained. (2) VAE-based models with Gaussian mixture prior perform better than those with simple Gaussian prior, especially on the dataset with complex characteristics, because they can well model the intrinsic multimodality of data by obtaining complex latent representations. (3) Due to the curse of dimensionality for multivariate time series, it is necessary to reduce the dimensionality of high dimensional data. (4) An end-to-end model is better than a two-stage model in our scenario, because it deals with dimension reduction and pattern capturing simultaneously so as to achieve optimal performance. (5) For VAE-based models, using reconstruction probability changes of univariate time series can better interpret outlier machine instances compared with using construction probability itself.

6 Related Works

Generally speaking, both outlier and anomaly detection methods aim to determine whether a data instance stands out as being dissimilar to all others and deviates from normal patterns [31]. In recent years, outlier/anomaly detection, particularly for time series, has emerged as a widely researched problem in both machine learning and deep learning area [31]. Moreover, it has become one of the fundamental problems in the field of Artificial Intelligence for IT Operations (AIOps). Among these detection methods, supervised models usually need manual labels, which are difficult to obtain, for model training [12], and their performance is usually sub-optimal due to class imbalance [31] and can only identify the outliers of known types. As a consequence, they are hard to be applied in practice and unsupervised methods are generally preferred. Although no specific work was designed for *detecting outlier machine instances* previously, a collection of unsupervised outlier/anomaly detection algorithms [3], [5], [7], [8], [15], [16], [17], [21], [42], [43], [44] have been proposed, which can be adopted or improved to solve this problem. They can be divided into three categories as follows.

(1) *Reconstruction methods* are based on an encoder-decoder architecture and apply reconstruction errors or probabilities to perform outlier/anomaly detection. [8] proposed Donut, a VAE-based model, to detect anomalies for seasonal *univariate* time series in web applications, and it used stochastic latent variables to learn the normal patterns of input data. OmniAnomaly [5] detected *anomalous time points* in multivariate time series for industry devices using stochastic Recurrent Neural Network. MSCRED [7] proposed a multi-scale convolutional recurrent encoder-decoder, and constructed signature matrices to represent the shape similarity among different pairs of *univariate* time series in complex systems. It mainly focuses on the inter-correlations rather than the overall performance of multivariate time series, which is different from detecting outlier machine instances by definition. DAGMM [21] detected anomalies in high dimensional data, which used an autoencoder to reduce dimensionality and applied a Gaussian mixture model to estimate the density distribution of low-dimensional space. Both OmniAnomaly and Donut applied reconstruction probabilities to detect anomalies, while MSCRED and DAGMM used reconstruction errors as detection scores.

(2) *Density estimation and clustering based methods.* Gaussian Mixture Model (GMM) [21] estimated the density of instances, and determined those in low density as outliers. "MDDTW +DBSCAN" [42] used a multi-dimensional dynamic time warping (DTW) to measure the shape similarity among instances, and applied DBSCAN to cluster these instances. Deep Clustering Network (DCN) [43] combined AE for dimensionality reduction and K-means for clustering. Those small and outlier clusters were determined as outliers.

(3) *Classification based methods*, as detecting outlier machine instances is a binary classification problem. Isolation Forest (IForest) [44] isolated instances by partitioning the value of features, and believed that outliers were those close to the root. One Class Support Vector Machines (OCSVM) [3] learned a boundary surrounding normal instances, and those outside the boundary were considered as outliers.

In comparison, none of the above methods can well address the challenges lying in detecting outlier machine instances, i.e., multivariate time series with various shapes and high dimensionality, and DOMI performs better than them, as discussed in Section 4.2.

7 Conclusion

This paper formally defines the problem of *detecting outlier machine instances*, which plays a critical role in monitoring systems and is extremely important for Internet service management. To solve this problem, we propose a novel end-to-end unsupervised method, DOMI, which combines GMVAE with 1D-CNN. It learns the normal patterns of machine instances by learning their low-dimensional representations, fully capturing multivariate time series' shapes and various data distributions. Moreover, to better approximate the training loss, we propose a multi-sample Monte Carlo integration approximation method for GMVAE based models. For detected outlier machine instances, DOMI further provides an effective interpretation based on the reconstruction probability changes of univariate time series. Extensive experiments have been carried out on a real-world dataset collected from ByteDance (a top global content service provider), demonstrating DOMI's excellent performance. The code of DOMI and the dataset used in our experiments have been publicly published on Github for the reproducibility of our results.

The ultimate objective of machine management is failure detection, localization, and mitigation in a real-time fashion automatically. This paper mainly focuses on the first step, i.e., outlier detection, and automatic mitigation is our further work. In our efforts to contribute to this field, we plan to construct a failure library containing the detailed information, such as failure types, root causes, and solutions (such as turning down the bad machines, or migrating the processes running in a bad machine to other machines), of historical machine failures. With this library, for each detected outlier machine instance, we will build models to automatically learn the correlations between failure patterns, root causes, and solutions. Note that DOMI can be considered as a generic approach to outlier detection for entities with multivariate time series beyond machines. In the future, we would like to apply it to other areas, e.g., robot (with multiple monitored sensors) failure detection, health monitoring (using biomedical signals).

ACKNOWLEDGMENTS

The authors would like to thank Yuqing Liu, Ping Liu, and Zhihan Li for their helpful discussions on this work. They also thank Juexing Liao for proofreading this article. This work was supported in part by the National Key R&D Program of China under Grant 2019YFB1802504, in part by the National Natural Science Foundation of China under Grant 61902200 and Grant 62072264, in part by the China Postdoctoral Science Foundation under Grant 2019M651015, and in part by the Beijing National Research Center for Information Science and Technology (BNRist) Key Projects.

REFERENCES

[1] V. Nair *et al.*, "Learning a hierarchical monitoring system for detecting and diagnosing service issues," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 2029–2038.
[2] Y. Su *et al.*, "CoFlux: Robustly correlating KPIs by fluctuations for service troubleshooting," in *Proc. Int. Symp. Qual. Service*, 2019, Art. no. 13.
[3] Y.-L. Lee, D.-C. Juan, X.-A. Tseng, Y.-T. Chen, and S.-C. Chang, "DC-prophet: Predicting catastrophic machine failures in datacenters," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2017, pp. 64–76.
[4] M. Gabel, R. Gilad-Bachrach, N. Bjorner, and A. Schuster, "Latent fault detection in cloud services," Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2011–83, 2011.
[5] 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.
[6] 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. Discov. Data Mining*, 2018, pp. 387–395.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] D. Liu *et al.*, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proc. Internet Meas. Conf.*, 2015, pp. 211–224.
[13] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1544–1551, Jul. 2018.
[14] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," 2016, *arXiv:1607.00148*.
[15] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online detection of utility cloud anomalies using metric distributions," in *Proc. IEEE Netw. Operations Manag. Symp.*, 2010, pp. 96–103.
[16] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manag. Workshops*, 2011, pp. 385–392.
[17] C. Wang *et al.*, "Performance troubleshooting in data centers: An annotated bibliography?," *ACM SIGOPS Operating Syst. Rev.*, vol. 47, no. 3, pp. 50–62, 2013.
[18] C. C. Aggarwal, "Outlier analysis," in *Data Mining*. Berlin, Germany: Springer, 2015, pp. 237–263.
[19] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 139–152.
[20] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 2016, *arXiv:1603.06995*.
[21] B. Zong *et al.*, "Deep autoencoding Gaussian mixture model for unsupervised anomaly detection," in *Proc. Int. Conf. Learn. Representations*, 2018.
[22] W. Liao, Y. Guo, X. Chen, and P. Li, "A unified unsupervised gaussian mixture variational autoencoder for high dimensional outlier detection," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 1208–1217.
[23] Y. Guo, W. Liao, Q. Wang, L. Yu, T. Ji, and P. Li, "Multidimensional time series anomaly detection: A GRU-based Gaussian mixture variational autoencoder approach," in *Proc. Asian Conf. Mach. Learn.*, 2018, pp. 97–112.
[24] G. Dai, J. Xie, and Y. Fang, "Siamese CNN-BiLSTM architecture for 3D shape representation learning," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 670–676.
[25] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D convolutional neural networks and applications: A survey," 2019, *arXiv: 1905.03554*.
[26] S. Kiranyaz, A. Gastli, L. Ben-Brahim, N. Al-Emadi, and M. Gabbouj, "Real-time fault detection and identification for MMC using 1D convolutional neural networks," *IEEE Trans. Ind. Electron.*, vol. 66, no. 11, pp. 8760–8771, Nov. 2019.
[27] W. Shi, H. Zhou, N. Miao, and L. Li, "Dispersed exponential family mixture VAEs for interpretable text generation," 2020, *arXiv: 1906.06719*.
[28] M. P. Papazoglou and W.-J. Van Den Heuvel, "Service oriented architectures: Approaches, technologies and research issues," *VLDB J.*, vol. 16, no. 3, pp. 389–415, 2007.
[29] Y. Chen, S. Wang, S. Lu, and K. Sankaralingam, "Applying hardware transactional memory for concurrency-bug failure recovery in production runs," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 837–850.
[30] M. M. Botezatu, I. Giurgiu, J. Bogojeska, and D. Wiesmann, "Predicting disk replacement towards reliable data centers," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 39–48.
[31] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019, *arXiv: 1901.03407*.
[32] C. Lin *et al.*, "Early diagnosis and prediction of sepsis shock by combining static and dynamic information using convolutional-LSTM," in *Proc. IEEE Int. Conf. Healthcare Informat.*, 2018, pp. 219–228.
[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[34] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *J. Syst. Eng. Electron.*, vol. 28, no. 1, pp. 162–169, 2017.
[35] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture IE*, vol. 2, no. 1, pp. 1–18, 2015.
[36] N. Dilokthanakul *et al.*, "Deep unsupervised clustering with Gaussian mixture variational autoencoders," 2017, *arXiv:1611.02648*.
[37] A. Mnih and D. J. Rezende, "Variational inference for Monte Carlo objectives," 2016, *arXiv:1602.06725*.
[38] M. Collier and H. Urdiales, "Scalable deep unsupervised clustering with concrete GMVAEs," 2019, *arXiv: 1909.08994*.
[39] C. Robert and G. Casella, *Monte Carlo Statistical Methods*. Berlin, Germany: Springer, 2013.

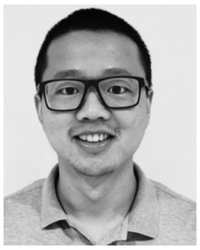
[40] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *Proc. 30th Int. Conf. Int. Conf. Mach. Learn.*, 2013, pp. III-1310–III-1318. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042817.3043083
[41] S. Basu, K. Wagstyl, A. Zandifar, L. Collins, A. Romero, and D. Precup, “Analyzing Alzheimer’s disease progression from sequential magnetic resonance imaging scans using deep 3D convolutional neural networks,” in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2018.
[42] J. Mei, M. Liu, Y.-F. Wang, and H. Gao, “Learning a Mahalanobis distance-based dynamic time warping measure for multivariate time series classification,” *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1363–1374, Jun. 2016.
[43] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, “Towards K-means-friendly spaces: Simultaneous deep learning and clustering,” in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 3861–3870.
[44] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation-based anomaly detection,” *ACM Trans. Knowl. Discov. Data*, vol. 6, 2012, Art. no. 3.
[45] S. Jiang, Y. Chen, J. Yang, C. Zhang, and T. Zhao, “Mixture variational autoencoders,” *Pattern Recognit. Lett.*, vol. 128, pp. 263–269, 2019.

Ya Su received the BS degree in software engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2016. He is currently working toward the PhD degree with the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include AIOps, service management, data mining, and machine learning.

Youjian Zhao received the BS degree from Tsinghua University, Beijing, China, in 1991, the MS degree from the Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1995, and the PhD degree from Northeastern University, Shenyang, China, in 1999. He is currently a professor at CS Department, Tsinghua University. His research mainly focuses on high speed internet architecture, switching and routing, and high-speed network equipment.

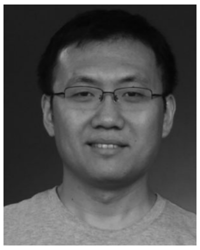
Ming Sun received the BS degree from Tsinghua University, Beijing, China, in 2019. He is currently working toward the MS degree with the Department of IIIS (Institute for Interdisciplinary Information Sciences), Tsinghua University, Beijing, China. His current research insterests include data analysis and machine learning.

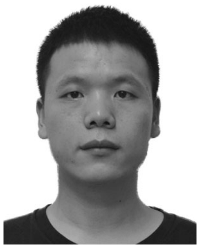
Shenglin Zhang (Member, IEEE) received the BS degree from the School of Computer Science and Technology, Xidian University, Xi’an, China, in 2012, and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2017. He is currently an assistant professor with the College of Software, Nankai University. His research interests include AIOps.


Xidao Wen received the PhD degree in information science from the School of Computing and Information, University of Pittsburgh, Pittsburgh, Pennsylvania, in 2019. He is currently a post-doc researcher at Tsinghua University. His research interests include data mining, machine learning, and artificial intelligence for IT operations.

Yongsu Zhang received the BS degree from the Huazhong University of Science and Technology, Wuhan, China, in 2009, and the MS degree from the Beijing University of Post and Telecommunications, Beijing, China, in 2011. He is currently a technical leader with ByteDance.

Xian Liu received the BS degree from Shanghai Jiao Tong University, Shanghai, China, in 2006, and the MS degree from Nanjing University, Nanjing, China, in 2008. He is currently a software engineer with ByteDance. His research interests include hardware and software performance and system management architect.


Xiaozhou Liu received the BS degree from Peking University, Beijing, China, in 2005. He is currently a software engineer with ByteDance. His research interests include data analysis, Linux, and AIOps.


Junliang Tang received the BS degree from Nanjing Tech University, Nanjing, China, in 2016, and the MS degree from the Huazhong University of Science and Technology, Wuhan, China, in 2018. He is currently a software engineer with ByteDance.

Wenfei Wu (Member, IEEE) received the BS degree from Beihang University, Beijing, China, in 2010, and the PhD degree from the Computer Sciences Department, University of Wisconsin-Madison, Madison, Wisconsin, in 2015. He is currently an assistant professor of IIIS Department, Tsinghua University. His research interest include networked systems.

Dan Pei (Senior Member, IEEE) received the BS and MS degrees from Tsinghua University, Beijing, China, in 1997 and 2000, respectively, and the PhD degree from the University of California, Los Angeles (UCLA), California, in 2005. He is currently an associate professor at Tsinghua University. His current research interests include management and improvement of the performance and security of networked services. Right now, he is focusing on the field of AIOps.