

Practical Anomaly Detection over Multivariate Monitoring Metrics for Online Services

Jinyang Liu[†], Tianyi Yang[†], Zhuangbin Chen^{†**}, Yuxin Su[‡],
Cong Feng[§], Zengyin Yang[§], Michael R. Lyu[†]

[†]The Chinese University of Hong Kong, Hong Kong SAR, China, {jyliu, tyang, lyu}@cse.cuhk.edu.hk

[‡]School of Software Engineering, Sun Yat-sen University, Zhuhai, China, {chenzhh36, suyx35}@mail.sysu.edu.cn

[§]Computing and Networking Innovation Lab, Huawei Cloud Computing Technology Co., Ltd, China,
{fengcong5, yangzengyin}@huawei.com

Abstract—As modern software systems continue to grow in terms of complexity and volume, anomaly detection on multivariate monitoring metrics, which profile systems' health status, becomes more and more critical and challenging. In particular, the dependency between different metrics and their historical patterns plays a critical role in pursuing prompt and accurate anomaly detection. Existing approaches fall short of industrial needs for being unable to capture such information efficiently. To fill this significant gap, in this paper, we propose CMAAnomaly, an anomaly detection framework on multivariate monitoring metrics based on collaborative machine. The proposed collaborative machine is a mechanism to capture the pairwise interactions along with feature and temporal dimensions with linear time complexity. Cost-effective models can then be employed to leverage both the dependency between monitoring metrics and their historical patterns for anomaly detection. The proposed framework is extensively evaluated with both public data and industrial data collected from a large-scale online service system of Huawei Cloud. The experimental results demonstrate that compared with state-of-the-art baseline models, CMAAnomaly achieves an average F1 score of 0.9494, outperforming baselines by 6.77% ~ 10.68%, and runs $10\times \sim 20\times$ faster. Furthermore, we also share our experience of deploying CMAAnomaly in Huawei Cloud.

Index Terms—Anomaly Detection; Multivariate Monitoring Metrics; Software Reliability

I. INTRODUCTION

As modern software systems, e.g., online service systems, have grown to an unprecedented scale, failures become inevitable, leading to significant revenue loss and user dissatisfaction [1], [2], [3], [4], [5]. To understand the health status of an online service system, various monitoring metrics such as network traffic, server response delay, and CPU usage rate are closely monitored [6], [7], [8], [9], [10]. Anomaly detection over the monitoring metrics is a crucial approach to ensure the reliability and availability of the system, which aims to discover unexpected events or rare items in data. Many efforts [11], [12], [13], [14] have been devoted to univariate metric anomaly detection, which deals with only one single type of metric. However, more often than not, different types of monitoring metrics collectively can indicate the occurrence of anomalies more precisely [2]. Different system components (e.g., microservices for different functionalities) are tightly

coupled, and failures tend to trigger anomalies in multiple monitoring metrics. For example, a problematic load balance server is often accompanied by a burst on both round-trip delay and in-bound traffic rate, which will further increase CPU utilization. However, a sudden rise in CPU utilization alone could be caused by regular service upgrades, which should not be regarded as an anomaly. Compared to univariate metric, anomaly detection on multivariate monitoring metrics is more challenging. We have identified three main reasons:

Demanding industrial requirements. In modern software systems, minutes of downtime could cause an expensive drain on company revenue. Therefore, suspicious anomalies should be quickly identified. Moreover, as systems continuously undergo feature upgrades and system renewal, the patterns of multivariate monitoring metrics may shift accordingly. To accommodate such ever-changing pattern shifts, the anomaly detection model must support fast model retraining.

Large-volume and noisy data. Although terabytes and even petabytes of metric data are being generated everyday, most of them do not contain much valuable information. For example, a significant portion of monitoring metrics only records plain system runtime behaviors. Moreover, a low signal-to-noise ratio is also a critical issue. Thus, multivariate metric anomaly detection with the presence of noise is challenging.

Sparsity of monitoring metrics' dependency. In industrial systems, hundreds or even thousands of monitoring metrics are being monitored. The dependencies among monitoring metrics are very sparse, i.e., most monitoring metrics are not or weakly dependent on other monitoring metrics. Therefore, how to automatically learn the dependencies among different monitoring metrics is critical towards efficient multivariate metric anomaly detection.

In the literature, many studies have shifted to anomaly detection on multivariate monitoring metrics, which mainly resorts to different deep learning-based models. For example, OmniAnomaly [15] proposes to learn the normal patterns of multivariate time series by modeling data distribution through stochastic latent variables. Anomalies are then determined by reconstruction probabilities. Similarly, Malhotra et al. [16] used an LSTM-based (long short-term memory) encoder-decoder network to learn time series's normal patterns and

**Zhuangbin Chen is the corresponding author.

Zhang et al. [17] used an attention-based convolutional LSTM network. Although tremendous progress has been made, we still observe two major limitations of existing approaches: 1) the interactions among monitoring metrics are not explicitly modeled, and 2) the efficiency falls behind industrial needs. Specifically, previous approaches [15], [16], [18] detect anomalies on multivariate monitoring metrics mainly by stacking different types of monitoring metrics into a feature matrix and feeding it to sophisticated neural network models. In this manner, each metric corresponds to one feature, and monitoring metrics' interactions are implicitly learned at the cost of complicated model design, resulting in inevitable false alarms. More recent studies tackle this problem by constructing an $m \times m$ metric inner-product matrix [17] or a complete graph [19] for m different monitoring metrics to capture the pairwise metric interaction, both of which yield an $\mathcal{O}(m^2)$ computation complexity. Similarly, heavy models such as graph neural networks [20] are employed to learn monitoring metrics' dependencies by feeding the matrix or training on the graph. Different from them, we argue that by properly modeling the interactions of monitoring metrics along with feature and temporal dimensions, cost-effective neural network models can be leveraged for anomaly detection.

In this paper, to overcome the aforementioned limitations, we propose CMAAnomaly, an efficient unsupervised model for anomaly detection over multivariate monitoring metrics. Specifically, we propose a collaborative machine to conduct cross-feature and cross-time metric interactions by taking the inner product of pairwise monitoring metrics and pairwise timestamps. Moreover, to capture the existing sparse dependency, we assign a weight to each metric (or timestamp), and the importance of a pair of dependencies is measured by the product of corresponding weights. However, such a weighted interactions mechanism still costs quadratic computation as previous approaches do. To alleviate this problem, inspired by factorization machines [21], we reformulate the proposed collaborative machine and reduce the computation from quadratic time complexity to linear complexity, which ensure the efficiency of CMAAnomaly. After that, to consider both the feature and temporal factors simultaneously, the results of interactions from both aspects are concatenated and fed to a cost-effective multi-layer perceptron (MLP). Particularly, the model is trained by learning from the historical normal patterns of monitoring metrics and predicting their future states. Anomalies are then identified based on prediction errors. Such learning paradigms have been proven to be effective in many related studies [22], [19]. To sum up, in this paper, we make the following major contributions:

- We propose CMAAnomaly, an efficient unsupervised model for anomaly detection over multivariate monitoring metrics (§ III). CMAAnomaly explicitly models the interactions among monitoring metrics along both temporal and feature dimensions. More importantly, we reduce the computation of our model from quadratic to linear complexity. In doing this, anomaly detection can be performed efficiently to process a large volume of data in the real world.

- We conduct extensive experiments with both public and industrial data collected from a large-scale online service provider, Huawei Cloud (§ IV). The experimental results demonstrate that CMAAnomaly achieves 0.9494 F1 score with 4.45% improvement over the state-of-the-art approaches. Moreover, CMAAnomaly can run $10\times \sim 20\times$ faster than the baseline models.
- We have successfully incorporated CMAAnomaly into the troubleshooting system of the Huawei Cloud. Feedback from on-site engineers confirms its practical benefits conveyed to various online services and products. We also share the hands-on experience story to benefit the community (§ VI).

II. BACKGROUND

A. Dependency of Multivariate Monitoring Metrics

In recent years, the complexity and scale of modern software systems are growing at a rapid speed. Various types of monitoring metrics that profile an entity's health status need to be closely monitored [15]. Particularly, an entity can be a logical one like a service or a physical one like a computing server. They provide system operators and engineers with the most prompt and direct way to understand the system. As different system components are tightly-coupled and work collaboratively, there exists some dependency among monitoring metrics, i.e., Multivariate monitoring metrics. Consequently, when failures are encountered, the dependent monitoring metrics are likely to exhibit abnormal behaviors simultaneously. Such dependencies reflect some intrinsic properties of the system and thus provide more reliable evidence for multivariate monitoring metric anomaly detection.

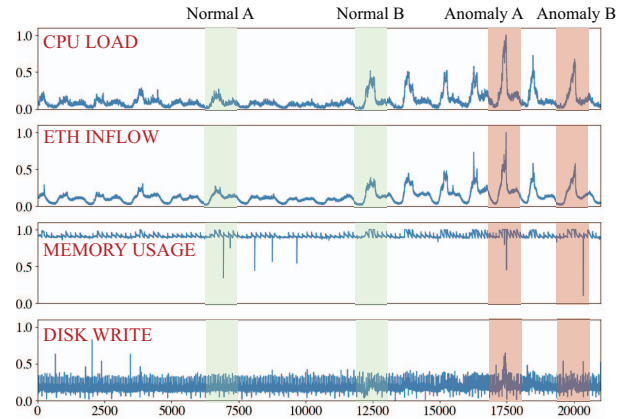


Fig. 1. Multivariate monitoring metrics Snippet from Server Machine Dataset

A real-world example is provided in Figure 1, which is from a public dataset released by [15]. The four monitoring metrics in the figure constitute a typical set of dependent monitoring metrics. Particularly, *CPU LOAD* and *ETH INFLOW* are strongly correlated as their curves exhibit a highly similar trend. Without considering such dependency, we will miss a complete picture of the system's health status. For example, in the segment marked as *Normal A*, we can see a clear spike

in *MEMORY USAGE*, which would be flagged as an anomaly without a glance at the other three monitoring metrics. Similar situation happens to segment *Normal B*, where boots can be clearly seen in both *CPU LOAD* and *ETH INFLOW*. Therefore, we need to consider the full set of multivariate monitoring metrics to pursue an accurate anomaly detection, as shown in segment *Anomaly A* and *Anomaly B*. Besides the dependency between monitoring metrics, we can also leverage historical monitoring metric patterns to reduce false positives. Specifically, in Figure 1, all monitoring metrics have witnessed some abnormal spikes in history. However, they do not necessarily indicate the occurrence of failures. By learning such historical patterns, our model will not be too sensitive to spikes and will be able to distinguish benign change patterns from anomalous ones.

B. Problem Statement

Anomaly detection on system monitoring metrics is crucial for proactive and prompt troubleshooting, aiming to discover abnormal status exhibited by the monitoring metrics. Such anomalies often indicate that a system encounters faults or attacks. In this paper, we focus on anomaly detection for an entity based on the multivariate monitoring metrics collected from it at equal-space timestamps [15]. The problem can be formally defined as follows.

The input of multivariate monitoring metrics can be represented as $X \in \mathbb{R}^{n \times m}$, where n is the number of different monitoring metrics and m is the number of observations. The t^{th} row of X , denoted as $x_t = [x_t^1, x_t^2, \dots, x_t^m]$, is a m -dimensional vector containing the observation of each monitoring metric at timestamp t . Similarly, the k^{th} column of X , denoted as $x^k = [x_1^k, x_2^k, \dots, x_n^k]$, is a n -dimensional vector containing the observations of the k^{th} monitoring metric. Particularly, we denote $x_{i:j}^k = [x_i^k, x_{i+1}^k, \dots, x_j^k]$ as a consecutive sequence of x^k from timestamp i to j .

The objective of anomaly detection for multivariate monitoring metrics is to determine whether a given x_t is anomalous, i.e., whether the entity is in abnormal status at timestamp t . For each timestamp t , our model calculates an anomaly score $s_t \in [0, 1]$, which represents the probability of x_t being anomalous. If s_t is larger than a pre-defined threshold θ , x_t will be predicted as an anomaly. The ground truth $\mathbf{y} \in \mathbb{R}^n$ is an n -dimensional vector consisting 0 and 1, where 0 indicates a normal point, and 1 indicates an anomalous one.

III. METHODOLOGY

In this section, we introduce CMAAnomaly, our anomaly detection framework for multivariate monitoring metrics. We will begin with an overview of CMAAnomaly, followed by a detailed explanation.

A. Overview of CMAAnomaly

In today's software systems, e.g., online service systems, engineers face an overwhelming number of monitoring metrics, which are being generated in a 24×7 non-stop basis. It would be labor-intensive and error-prone to manually investigate

each metric for failure detection and diagnosis. CMAAnomaly facilitates this process by providing automated and effective multivariate metric anomaly detection.

The overall framework of CMAAnomaly is illustrated in Figure 2, which consists of four phases, i.e., *data preprocessing*, *collaborative machine*, *model training*, and *anomaly detection*. The first phase preprocesses the data by applying normalization and window sliding. Particularly, the input types of monitoring metrics can vary depending on the application scenario. These monitoring metrics can be unified to the same range by normalization. In the subsequent phase, the preprocessed data is input into the collaborative machine, CMAAnomaly's core component, which effectively captures interactions among multivariate monitoring metrics across both feature and temporal dimensions. Next, we train a forecasting-based anomaly detection model [22], [23] that identifies anomalies via prediction errors. The collaborative machine enables the use of a cost-effective model architecture, with both phases crucial for a highly accurate and efficient anomaly detector. Finally, the trained model is used to detect anomalies in new observations.

B. Preprocessing

Different monitoring metrics may have distinct scales, for example, the metric monitoring the CPU execution, i.e., *CPU USAGE*, is in the range of 0% to 100%. However, the metric monitoring the network traffic, i.e., *INBOUND PACKAGE RATE* can range from zero to millions of kilobytes. Therefore, data normalization is performed for each individual metric to ensure the robustness of our model. After that, we conduct a sliding window to generate input to the model. Each window preserves the local pattern of multivariate monitoring metrics.

1) *Data Normalization*: We apply max-min normalization to each individual metric, i.e., x^k , as follows:

$$x_{norm}^k = \frac{x^k - \min(x^k)}{\max(x^k) - \min(x^k)}, \quad (1)$$

where the values of $\max(x^k)$ and $\min(x^k)$ are computed in the training data, which will then be used for test data normalization. For simplicity, we omit the "norm" subscript in the following elaboration.

2) *Sliding Window*: The sliding window is to partition monitoring metrics along the temporal dimension. Particularly, it consists of two attributes, i.e., window size ω and stride τ . The stride indicates the forwarding distance of the window along the time axis to generate multivariate metric windows. As the stride is often smaller than the window size, there exists overlapping between two consecutive windows. We denote the s^{th} sliding window as:

$$X_s = [x_{s\tau}, x_{s\tau+1}, \dots, x_{s\tau+\omega-1}] \quad (2)$$

where $s \in [0, 1, 2, \dots]$. X_s together with the observations at the next timestamp of the window, i.e., $\hat{x}_s = x_{s\tau+\omega}$, constitute a pair $(X_s, \hat{x}_s)$, where $X_s \in \mathbb{R}^{\omega \times m}$ and $\hat{x}_s \in \mathbb{R}^m$.

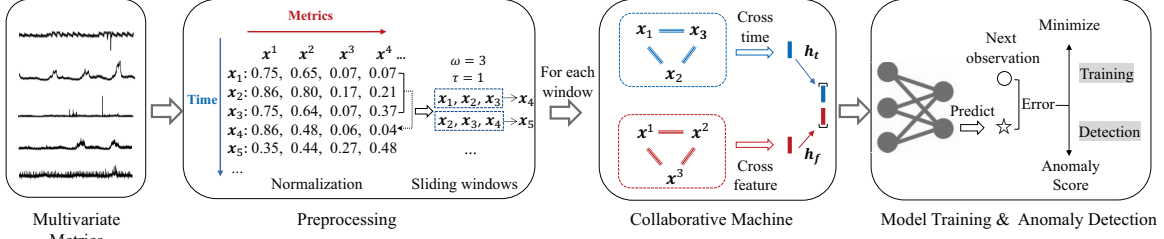


Fig. 2. Overall Framework of CMAnomaly

C. Collaborative Machine

As depicted in Figure 1, accurate anomaly detection on multivariate monitoring metrics requires consideration of metric dependencies. Historical patterns also offer crucial anomaly detection clues, especially if historical spikes have not resulted in reported failures. By learning these benign change patterns, our model can effectively reduce noise in monitoring metrics.

In this section, we propose the collaborative machine, a mechanism to efficiently capture the interactions of multivariate monitoring metrics along with both the feature and temporal dimensions. Specifically, we allow cross-feature and cross-timestamp interactions by taking the inner-product of pairwise monitoring metrics and pairwise timestamps in a multivariate metric window as shown in Figure 2. The results are two types of vectors, i.e., metric feature vector and temporal vector, which captures the dependencies and historical patterns of monitoring metrics, respectively. Inspired by the factorization machine [21], we reformulate metric and timestamp inner-product pairs to achieve linear time complexity, reducing computational complexity. Both types of features are then concatenated and fed into a cost-effective model for training.

1) *Multivariate Monitoring Metrics Interactions:* To explicitly capture the dependency between multivariate monitoring metrics and their historical patterns, for each sliding window, denoted as $X_s \in \mathbb{R}^{\omega \times m}$, we calculate the pairwise inner product of all metric feature vectors, i.e., $x_{s\tau, s\tau+\omega-1}^k, k \in [1, m]$, and temporal vectors, i.e., $x_t, t \in [s\tau, s\tau + \omega - 1]$. Particularly, for notation simplification, we remove the subscript of feature vectors.

$$h_f = b_0 + \sum_{i=1}^m w_i x^i + \sum_{i=1}^m \sum_{j=i+1}^m \langle x^i, x^j \rangle v_i v_j \quad (3)$$

$$h_t = \hat{b}_0 + \sum_{i=1}^{\omega} \hat{w}_i x_i + \sum_{i=1}^{\omega} \sum_{j=i+1}^{\omega} \langle x_i, x_j \rangle \hat{v}_i \hat{v}_j \quad (4)$$

The cross-feature and cross-time metric interactions, denoted as h_f and h_t , are formulated as Equation 3 and 4, respectively. Particularly, we only elaborate on the Equation 3 as Equation 4 shares the same format. In Equation 3, $b_0, w_i, v_j, v_j \in \mathbb{R}$ are trainable parameters, $x^i, x^j \in \mathbb{R}^{\omega}$ are the i^{th} and j^{th} column of X_s with each column representing all the observations of a metric in the corresponding window, and $\langle \cdot, \cdot \rangle$ is the operation of inner product. The equation is composed of three terms: the first term is a trainable bias, the second term is a weighted sum of all monitoring metrics

without explicit interaction, and the third term is the core part of the proposed collaborative machine, which models the pairwise metric interactions. Specifically, each pair of metric interactions is modeled by the inner product of the corresponding two monitoring metrics. However, not all the pairs share the same importance. To model the importance of each metric and its interaction with other monitoring metrics, we assign a trainable weight to each particular metric, e.g., v_i for the i^{th} metric. After that, while interacting with other monitoring metrics, the product of the weights can measure the importance of the combination, e.g., $v_i v_j$ denotes the importance of the interaction between the i^{th} and j^{th} monitoring metrics.

2) *Efficient Computation:* As all pairwise feature and temporal interactions require to be computed, the operations shown in Equation 3 and 4 are computationally expensive, i.e., $\mathcal{O}(m^2)$ and $\mathcal{O}(\omega^2)$, respectively. They are not practical for industrial scenarios as a small increase in the number of multivariate monitoring metrics or observation step size, i.e., larger m and ω , will quickly increase the computation complexity. To capture the feature and temporal interactions more efficiently, inspired by the work of Rendle et al. [21], we reformulate the pairwise interaction term in Equation 3 and 4, which could greatly reduce the computation cost, leading to a linear time complexity. Particularly, we only demonstrate the reformulation of the feature interaction term in Equation 3, as the temporal version in Equation 4 can be derived similarly:

$$\sum_{i=1}^m \sum_{j=i+1}^m \langle x^i, x^j \rangle v_i v_j \quad (5)$$

$$= \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \langle x^i, x^j \rangle v_i v_j - \frac{1}{2} \sum_{i=1}^m \langle x^i, x^i \rangle v_i v_i \quad (6)$$

$$= \frac{1}{2} \left(\sum_{i=1}^m \sum_{j=1}^m \sum_{r=1}^{\omega} x_r^i x_r^j v_i v_j - \sum_{i=1}^m \sum_{r=1}^{\omega} x_r^i x_r^i v_i v_i \right) \quad (7)$$

$$= \frac{1}{2} \sum_{r=1}^{\omega} \left(\left(\sum_{i=1}^m x_r^i v_i \right) \left(\sum_{j=1}^m x_r^j v_j \right) - \sum_{i=1}^m (x_r^i)^2 v_i^2 \right) \quad (8)$$

$$= \frac{1}{2} \sum_{r=1}^{\omega} \left(\left(\sum_{i=1}^m x_r^i v_i \right)^2 - \sum_{i=1}^m (x_r^i)^2 v_i^2 \right) \quad (9)$$

Equation 9 is mathematically equivalent to the third term of Equation 3. The time complexity for Equation 5 is $\mathcal{O}(m^2 \omega)$ since the item $\langle x^i, x^j \rangle$ needs ω times computations. After

our reformulation, the computation complexity for capturing metric interactions is decreased from $\mathcal{O}(m^2\omega)$ to $\mathcal{O}(m\omega)$ as shown in Equation 9. Moreover, the computation involved in Equation 9 can be implemented as matrix multiplications, which can be accelerated by GPU (graphics processing unit) for better efficiency. By substituting the interaction terms in both Equation 3 and 4, we can capture the pairwise cross-feature and cross-time interactions to produce h_f and h_t in a highly efficient manner.

D. Model Training and Anomaly Detection

The last two phases of CMAAnomaly are model training and anomaly detection. The anomaly detection model of CMAAnomaly works in a forecasting manner. To be more specific, the model is trained to predict the next metric values given preceding observations. Anomaly is then detected based on prediction errors, as shown in Figure 2. In the training phase, as most multivariate monitoring metrics would reflect the normal status of an entity, the model will learn the normal patterns of monitoring metrics, i.e., what the next observations would be given previous ones. Although there could be anomalies in the training data, they tend to be forgotten by the model as they rarely appear. Consequently, in the detection phase, the model will predict "normal" metric values based on the learned patterns. If the real observations deviate from the predicted ones by a significant margin, an anomaly may happen, i.e., the entity is not in its normal status. Particularly, the deviation measures the likelihood of the occurrence of the anomaly.

Our framework supports various types of neural network models for anomaly detection. However, thanks to the formulation of metric interactions in Equation 3 and 4, we find a cost-effective model architecture, i.e., multi-layer perceptron (MLP), is sufficient to achieve competitive performance. The anomaly detection model can be formulated as follows:

$$\tilde{h}_{i+1} = \sigma(\tilde{h}_i \tilde{X}_i + \tilde{b}_i), i = 0, 1, \dots, L-1, \quad (10)$$

where L is the number of layers of the MLP model, $\tilde{W}_i, \tilde{b}_i$ are trainable parameters with customizable size, and $\sigma(x) = \max(0, x)$ is the ReLU activation function. Particularly, we simultaneously consider the cross-feature and cross-time metric interactions by concatenating h_f and h_t , which is the input to the model, i.e., $\tilde{h}_0 = \text{concat}(h_f, h_t)$. $\hat{y} = \tilde{h}_L \in \mathbb{R}^m$ is the prediction result produced by the last layer of the MLP model, which contains the predicted values for all monitoring metrics at the next timestamp.

1) *Training*: We train the anomaly detection model by minimizing the following mean square error (MSE) loss $\mathcal{L}$ between the predictions and ground truth observations:

$$\mathcal{L} = \sum_{i=1}^N \|\hat{y}_i - \hat{x}_i\|_2, \quad (11)$$

where N is the number of training sliding windows. $\hat{y}_i \in \mathbb{R}^m$ and $\hat{x}_i = x_{i\tau+\omega} \in \mathbb{R}^m$ are the predicted and ground truth ob-

servations for the i^{th} window, respectively. The minimization of $\mathcal{L}$ is conducted using the Adam optimizer [24]. Particularly, we stop the training process when the loss $\mathcal{L}$ is relatively stable, i.e., the change of $\mathcal{L}$ is less than 10^{-5} . With the minimization of loss $\mathcal{L}$ during training, CMAAnomaly can learn from the normal patterns in the training data by updating all trainable parameters, e.g., $v_i v_j$ denoting the interaction weights.

2) *Detection*: After the model is trained, we compute an anomaly score for each window X_i in the testing data. Specifically, we first calculate the MSE between the predicted and ground truth observations, and then apply the sigmoid function to rescale the score to the range $[0, 1]$, which represents the probability of the occurrence of an anomaly:

$$s_i = \phi\left(\frac{1}{m} \|\hat{y}_i - \hat{x}_i\|_2\right) \quad (12)$$

where $\phi(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

To determine whether an anomaly has happened, a threshold θ should be defined for the anomaly score. The timestamps with a large anomaly score, i.e., $s_i \geq \theta$, should be regarded as anomalous points. In reality, the threshold can be set by on-site engineers based on their experience. A large threshold imposes a strict anomaly detection policy, which may miss important system failures, i.e., low recall. However, a small threshold increases the sensitivity to metric changes, resulting in false alarms, i.e., low precision. In our experiments, we empirically set the threshold that yields the best experimental results.

IV. EXPERIMENTS

In this section, we evaluate CMAAnomaly using both public data and industrial data. Particularly, we aim to answer the following research questions (RQs):

RQ1: How effective is CMAAnomaly in anomaly detection over multivariate monitoring metrics?

RQ2: How much the proposed collaborative machine mechanism contributes to the overall effectiveness of CMAAnomaly?

RQ3: How efficient is CMAAnomaly compared with existing approaches?

A. Experimental Setup

1) *Dataset*: To evaluate the effectiveness of CMAAnomaly, we conduct experiments on three public datasets. Moreover, to confirm its practical significance, we collect an industrial dataset from the Networking service of the Huawei Cloud.

Public dataset. SMD (Server Machine Dataset). SMAP and MSL are two real-world datasets released by NASA. They are collected from the running spacecraft and contain a set of telemetry anomalies corresponding to actual spacecraft issues involving various subsystems and channel types [22]. SMD dataset is collected from a large Internet company containing a 5-week-long monitoring metrics of 28 machines released by Su et al. [15].

Industrial dataset. Besides the public dataset, we also collected real-world monitoring metrics from a global cloud

service provider Huawei Cloud to conduct a more comprehensive evaluation. The online services of the Huawei Cloud have been supporting tens of millions of users worldwide. Therefore, to provide a stable 24×7 service, the status of each online service is closely monitored with monitoring metrics. The engineers can fix problems timely if the anomalies of monitoring metrics can be automatically detected and reported in real-time. To evaluate our method in a practical scenario, we collected a 30-day-long monitoring metrics dataset with 13 online services. The dataset is termed as *Industry* in Table I. Each of the online services has 15~30 different types of monitoring metrics. We use the first 20 days of monitoring metrics as the training data and the rest as the testing data. For labeling the dataset, we rely on corresponding issue reports, which contain the start and end times of problems identified by on-site engineers or customers of an online service. Anomalies are designated during periods when the online service is deemed problematic, while the remaining data is considered normal. We also open-source this dataset to facilitate future studies in this field¹.

The statistics of the datasets are listed in the Table I. *#Train* and *#Test* denote the number of observations in the training and testing dataset, respectively. *#Entities* denotes the number of entities in a dataset. Specifically, in SMAP and MSL, the entity is a spacecraft that can be viewed as a complex software system composed of multiple interconnected components, each serving a specific functionality. In SMD, the entity is a server machine. In our industrial dataset, each entity serves as an online service, e.g., load balance, machine learning, deep learning, remote storage, and authentication. *#Dim.* stands for the number of monitoring metrics associated with an entity. *Anomaly Ratio* is the ratio of the number of anomalous points to the number of observations.

TABLE I
DATASET STATISTICS

Dataset	#Train	#Test	#Entities	#Dim.	Anomaly Ratio (%)
SMAP	135,183	427,617	55	25	13.13
MSL	58,317	73,729	27	55	10.72
SMD	708,405	708,420	28	38	4.16
Industry	1,728,000	864,000	13	15~30	1.16

2) *Experimental Environment*: We run all experiments on a Linux server with Intel Xeon Gold 6148 CPU @ 2.40GHZ and 1TB RAM, GeForce RTX 2080 Ti, running Red Hat 4.8.5 with Linux kernel 3.10.0. In addition, the proposed model is implemented under the PyTorch framework and runs on the GPU.

3) *Evaluation Metrics*: As anomaly detection is a binary classification problem, we employ Precision (PC), Recall (RC), and F1 score (F1) for evaluation, as calculated by Equation 13. Specifically, TP (true positive) denotes the number of anomalous samples that the model correctly predicts as an anomaly. FP (false positive) is the number of normal samples

that the model mistakenly recognizes as an anomaly. FN (false negative) is the number of normal samples that are predicted as normal. F1 score is the harmonic mean between precision and recall. A higher F1 score indicates a better model. To show the best performance of our method, we manually tune the anomaly threshold to achieve the best F1 score as done by [25].

$$PC = \frac{TP}{TP + FP} \quad RC = \frac{TP}{TP + FN} \quad F1 = 2 \cdot \frac{PC \cdot RC}{PC + RC} \quad (13)$$

In practice, failures often last for a certain period, which results in consecutive anomalies within a time range. Therefore, it is practical to apply the *point adjustment* process as done in [15], [25], [11], which consider that a model makes a correct prediction for an anomalous segment if at least one point in the segment is successfully predicted as an anomaly. The following evaluation results are obtained after applying point adjustment.

B. RQ1: Effectiveness of CMAnomaly

To study the effectiveness of CMAnomaly, we compare its performance with various state-of-the-art baseline models on both the public datasets and the industrial dataset collected from Huawei Cloud. We first train CMAnomaly on the training data until convergence. Then, we compute the anomaly score for each observation of the testing data. After that, we manually try thresholds θ in the range of $[0, 1]$ with a step size of 0.1 to produce the prediction. In particular, for the baselines with automatic threshold selection mechanism, we replace such mechanism with manual selection. In doing this, we follow a recent work [25] to obtain the best performance of all models for a fair comparison. Finally, the predictions of all models are adjusted to compute PC, RC, and F1.

Performance on public datasets. We select six unsupervised multivariate metric anomaly detection methods as the baselines, which are isolation forest (IF) [26], LSTM-NDT [22], DAGMM [27], LSTM-VAE [18], autoencoder (AE) [25], OmniAnomaly [15] and USAD [25].

The overall performance is shown in Table II, where we mark the best F1 score in bold and underline the second-best ones. We also report the average metrics on all datasets in the "Average" columns. We have the following observations: (1) IF achieves the worst performance compared with other baseline models. Because IF tries to isolate the anomalous timestamp independently, which loses the temporal dependency. (2) Other LSTM-based or AE-based methods including LSTM-NDT, DAGMM, LSTM-VAE, AE perform much better than IF and achieve $0.6861 \sim 0.8283$ F1 score because these models take a window of observations as input, which helps to retain valuable historical information. (3) OmniAnomaly and USAD outperform other baselines by a large margin and achieve nearly 0.88 f1 score. These two methods introduce different mechanisms to ensure robust anomaly detection. OmniAnomaly models MTS through stochastic variables and uses reconstruction probabilities to determine anomalies. Differently, USAD utilizes the generative adversarial network (GAN) to train autoencoders, which allows the model to detect anomalies close to normal data. (4) The proposed method CMAnomaly achieves the highest F1 score on all datasets. The average F1 score is improved to 0.9494 from the second-best, i.e., 0.8892, achieved by USAD. The cost-effective design of CMAnomaly allows it to incur fewer trainable parameters than USAD. As a consequence, overfitting to the training data could be alleviated. In other words, only normal patterns with relatively high occurrence are learned by CMAnomaly during training. When making anomaly detection on the test data, CMAnomaly can be less sensitive and only raise the anomaly score when a real anomaly happens, avoiding more false alarms than USAD. The experimental results indicate that the

¹<https://github.com/OpsPAI/CMAnomaly>

TABLE II
ACCURACY COMPARISON OF DIFFERENT ANOMALY DETECTION METHODS ON PUBLIC DATASETS

Methods	SMAP			MSL			SMD			Average		
	PC	RC	F1	PC	RC	F1	PC	RC	F1	PC	RC	F1
IF	0.4423	0.5105	0.4671	0.5681	0.6740	0.5984	0.5938	0.8532	0.5866	0.5347	0.6792	0.5507
LSTM-NDT	0.8965	0.8846	<u>0.8905</u>	0.5934	0.5374	0.5640	0.5684	0.6438	0.6037	0.6861	0.6886	0.6861
DAGMM	0.6334	0.9984	0.7124	0.7562	0.9803	0.8112	0.6730	0.8450	0.7231	0.6875	0.9412	0.7489
LSTM-VAE	0.7164	0.9875	0.7555	0.8599	0.9756	0.8537	0.8698	0.7879	0.8083	0.8154	0.9170	0.8058
AE	0.7216	0.9795	0.7776	0.8535	0.9748	0.8792	0.8825	0.8037	0.8280	0.8192	0.9193	0.8283
OmniAnomaly	0.7585	0.9756	0.8054	0.9140	0.8891	0.8952	0.9809	0.9438	<u>0.9441</u>	0.8845	0.9362	0.8816
USAD	0.7697	0.9831	0.8186	0.8810	0.9786	<u>0.9109</u>	0.9314	0.9617	0.9382	0.8607	0.9745	<u>0.8892</u>
CMAAnomaly	0.9989	0.8916	0.9187	0.9903	0.9690	0.9782	0.9658	0.9379	0.9512	0.9850	0.9328	0.9494

PC of CMAAnomaly is significantly higher than USAD (0.9850 vs 0.8607), which implies fewer false alarms.

TABLE III
ACCURACY COMPARISON ON INDUSTRIAL DATASET

Methods	Industry		
	PC	RC	F1
OmniAnomaly	0.6639	0.8382	0.7283
LSTM-VAE	0.8273	0.7436	0.7560
CMAAnomaly	0.9179	0.8202	0.8368

Performance on industrial dataset. To evaluate the effectiveness of CMAAnomaly in the industrial application, we apply two most effective anomaly detection methods whose source code is publicly available, i.e., LSTM-VAE [18] and OmniAnomaly [15] on the industrial dataset for comparison.

The experimental results are shown in Table III. In particular, the PC of OmniAnomaly is the lowest, but the RC is the highest because the complex architecture of OmniAnomaly incurs more trainable parameters, which makes it easier to overfit the training data. Therefore, OmniAnomaly is more sensitive to capture more anomalies but has the most false positive alarms. Different from the results in Table II, LSTM-VAE outperforms OmniAnomaly on this dataset in terms of F1. LSTM-VAE has a more light-weight design than OmniAnomaly, so LSTM-VAE suffers less overfitting. As a result, LSTM-VAE only raises the anomaly score when the new observation deviates more from the prediction. In this case, though higher PC is achieved, LSTM-VAE has the lowest RC because it cannot effectively find all possible anomalies. CMAAnomaly can balance PC and RC better and achieves the best F1 score, ~ 0.08 higher than the second-best one achieved by LSTM-VAE. The collaborative machine facilitates CMAAnomaly to capture the dependency of the training monitoring metrics effectively. Therefore, CMAAnomaly avoids overfitting the noisy points existing in training data e.g., usual spikes as shown in Figure 1. Instead, CMAAnomaly reports a higher anomaly score only when the dependent monitoring metrics are anomalous, thus achieves the highest precision. Moreover, CMAAnomaly keeps the sensitivity to detect more true positive samples thanks to the ability of capturing the dependency.

C. RQ2: Effectiveness of Collaborative Machine

To evaluate the effectiveness of the proposed collaborative machine used in CMAAnomaly. We firstly produce a variant of CMAAnomaly by removing the collaborative machine (CM). Specifically, we use the average of all the monitoring metrics across the time dimension as the input of the MLP instead of hidden vectors computed by Equation 3 and Equation 4. In doing this, explicit interactions are removed. Then we apply the variant to perform anomaly detection on three public datasets SMAP, MSL and SMD. Then, we compare the

F1 score obtained by the variant with CMAAnomaly. Figure 3 shows the accuracy comparison between the variant and CMAAnomaly. The variant is denoted as "CMAAnomaly w/o CM" in the figure. We can observe that the accuracy of the variant drops on all three datasets. Especially, the average accuracy decreases by ~ 0.07 from 0.9494 to 0.8789. This can be explained by the loss of valuable temporal and feature information of the variant. On the one hand, without the collaborative machine, the sequential dependency and feature dependency cannot be captured, which hinders the model to learn from the normal patterns in the training data. As a consequence, when making the prediction in the anomaly detection stage, the new normal observation cannot be accurately predicted thus larger anomaly score may be produced. Therefore, more false alarms are reported.

D. RQ3: Efficiency of CMAAnomaly

In this section, we evaluate the efficiency of CMAAnomaly compared with the baseline models. We select the OmniAnomaly and LSTM-VAE for comparison because they are the most effective methods that are open-source. We conduct the comparison based on the official implementation of OmniAnomaly² and LSTM-VAE³. Next, we perform these methods on the industrial dataset and record training time and prediction time. Training time defines as the total time of feeding the preprocessed data to the model, and prediction is the total time used to predict the next observation on all the windows in the test set. Specifically, for a fair comparison, all the models are trained in a batch-wise manner for one epoch, and the same batch size is used. In real-world practice, we need to apply different window sizes to detect anomalies under different granularities. Therefore we increase the window size ω in the range of [16, 32, 64, 128, 256] to evaluate the efficiency of different methods. Stride is set as $\tau = 5$ on the training data and $\tau = 1$ on the test data because anomaly detection on test data should be conducted under finer granularity to alleviate missing anomalies. All the experiments run on the same computation machine on a GPU of GeForce RTX 2080 Ti.

Figure 4(a) and 4(b) show the efficiency comparison of the selected models in the training and prediction phases, respectively. According to the results, we can see that (1) the computational cost of LSTM-VAE and OmniAnomaly increases quickly with a larger window size, because both of the methods adopt LSTM as a core part to handle sequential input. Therefore, when we increase the window size, the depth of the LSTM increases, thus incurs more computation. (2) OmniAnomaly costs more time (nearly $2\times$) than LSTM-VAE though they share similar LSTM and VAE architecture. Nevertheless, OmniAnomaly introduces additional stochastic variable connection and planar normalizing flow to model the monitoring metrics, which are more computationally expensive. (3) CMAAnomaly is the most efficient method. To be more specific, CMAAnomaly is

²<https://github.com/NetManAI/Ops/OmniAnomaly>

³<https://github.com/TimyadNyda/Variational-Lstm-Autoencoder>

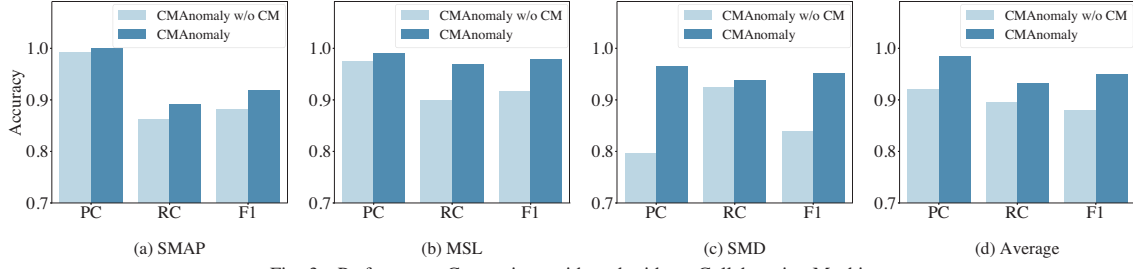


Fig. 3. Performance Comparison with and without Collaborative Machine

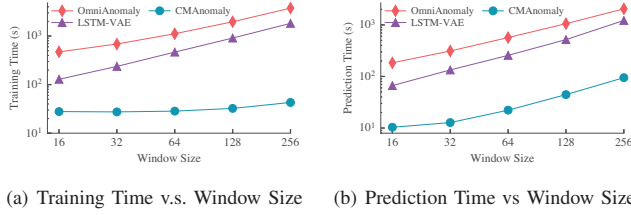


Fig. 4. Efficiency of CMAAnomaly

nearly 10x faster than LSTM-VAE and 20x than OmniAnomaly. Significantly, the training and prediction time do not increase a lot when varying the window size. Mainly, since τ on the test set is set to 1 and less than that of the training data, more test sliding windows are generated. Therefore, the test time of CMAAnomaly is slightly larger than the training time. This only happens on CMAAnomaly because the other two methods need more computation on the back-propagation during training. The cost-effective design and reduction of the computation complexity of CMAAnomaly benefit its efficiency. Moreover, both the collaborative machine and MLP of CMAAnomaly can be well accelerated by the asynchronous computing architecture of GPUs. Therefore, CMAAnomaly could conduct efficient training and prediction.

We want to emphasize that the efficiency of CMAAnomaly can meet the demanding requirements of industrial scenarios. (1) Generally, a large-scale system is closely monitored by a large number of monitoring metrics collected every second. Such a large quantity of monitoring metrics cloud be processed by CMAAnomaly in real-time. (2) CMAAnomaly can be fast retrained to learn the new patterns of monitoring metrics after the system upgrades. With this advantage, the deployment of an online anomaly detection service will not be suspended for too long.

E. Case Study

To make a more concrete evaluation of CMAAnomaly, we provide a case study on how CMAAnomaly computes anomaly scores for given multivariate monitoring metrics. This case study is conducted on a segment of the industrial dataset. The monitoring metrics monitor the traffic of connected network nodes supporting a service. After the model is trained, we apply the model to consecutively predict the monitoring metrics and detect anomalies. The results are shown in Figure 5. In this figure, the first four rows show the real observation of the monitoring metrics and the prediction made by CMAAnomaly. The last row shows the anomaly score of the entity and a threshold selected. If we consider the monitoring metrics individually, the first two monitoring metrics have spikes at both segments marked as "Normal" and "Anomaly" so anomalies would be alarmed. Differently, our method computes a much lower anomaly score of the "Normal" segment than the "Anomaly" segment by simultaneously considering the dependency of all the monitoring metrics. As a result, the anomaly score at the true positive segment can be magnified. In fact, the "Normal" spikes are caused by a typical increase of user requests to a port of the service. However, in the

"Anomaly" segment, a physical machine failure happens, which leads to anomalous patterns of the monitoring metrics.

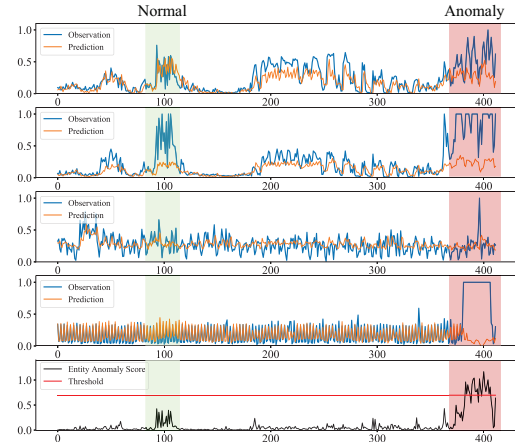


Fig. 5. Case Study on Industrial Dataset

F. Threat to Validity

We have identified the following major threats to validity.

Labeling noise. To evaluate the practical usage of CMAAnomaly, we conduct experiments on an industrial dataset collected from Huawei Cloud. The evaluation requires on-site engineers to manually inspect the monitoring metrics and label the anomalies. As the labeling is done manually, label noises, i.e., false positives and false negatives, may be introduced. However, the engineers we invited have rich domain knowledge and years of system troubleshooting experience. They are required to discuss and confirm the labeling results with each other. Moreover, the failures we selected are typical to the service systems. Engineers can quickly and confidently do the labeling work. Therefore, we believe the amount of noise is small.

Model implementation and parameter settings. The implementation of our model and baseline models as well as parameter settings are important internal threats to the validity. To reduce this threat, we collect the source code released by each work or directly employ the results employed in the paper. For our model, we employ peer code review. Specifically, all authors of this paper are required to check the code carefully. Moreover, all experiments are repeated multiple times and the average results are reported. In terms of parameter settings, we conduct a series of comparative experiments and empirically select the best results. As our model possesses the merit of high efficiency, the best results can be quickly obtained.

V. RELATED WORK

In the literature, anomaly detectors over multivariate metrics can be categorized to *implicit modeling* and *explicit modeling* schemas.

Implicit Modeling. The major challenge of multivariate anomaly detection comes from the effective modeling of complex temporal dependence and stochasticity of Multivariate monitoring metrics. Much previous work attempts to capture the normal temporal dependence by

modeling hidden states implicitly. For example, Hundman et al. [22] leverages LSTM for anomaly detection in spacecraft time-series metrics based on prediction errors. Malhotra et al. [16] proposes an LSTM-based encoder-decoder network that reconstructs “normal” time series and detects anomalies through reconstruction errors. Another way to model normal patterns is to learn the distribution of input data like deep generative models [28] and deep Bayesian network [29]. Donut [12] employs Variational AutoEncoder (VAE) to generate the normal hidden state of seasonal monitoring metrics without expert-labeled data. With solid theoretical explanation, Donut successfully detects anomalies in seasonal monitoring metrics with various patterns and data quality, but enjoys high time complexity in the training phase. To reduce the training complexity, DAGMM [27] simplify the hidden state as a combination of several Gaussian distributions. USAD [25] improves the autoencoder framework by incorporating adversarial samples to speed up the training phase. OmniAnomaly [15] uses a stochastic recurrent neural network to simulate the normal distribution of multivariate time-series data. Anomalies are identified when the input data reconstruction has low probability. However, the generalization capability of these generative approaches with implicitly modeling is degraded when they encounter severe noise in temporal monitoring metrics, which is very common in industrial production systems.

Explicit Modeling. To reduce the negative effect of noisy data in multivariate monitoring metrics, the temporal dependencies such as the inter-correlations between different parts of monitoring metrics should be captured in an explicit way. Zhang et al. [17] proposes a convolutional recurrent encoder-decoder framework called MSCRED to characterize multiple levels of the normal states in different steps of monitoring metrics. MSCRED employs a convolutional encoder and convolutional LSTM network to capture the monitoring metric interactions and temporal patterns, respectively. Finally, it generates different levels of anomaly scores based upon the severity of different incidents. Similar to our approach, Zhao et al. [19] consider each univariate monitoring metric as an individual feature to capture the complex dependencies of multivariate monitoring metrics from temporal and feature perspective. Their proposed method incorporates feature-oriented and time-oriented graph attention mechanisms to obtain mixed hidden representation from a combination of forecasting-based and reconstruction-based frameworks. However, the overall framework enjoys high time-complexity and is hard to address the demanding industrial requirements for quick response and adjustment.

VI. INDUSTRIAL EXPERIENCE

A. Success Story

In Huawei Cloud, CMAAnomaly has been successfully incorporated into the troubleshooting system for two years, serving several essential services that have a large number of customers worldwide. We conduct field interviews with on-site engineers to collect feedback. Based on the feedback, we have seen it shedding light on highly accurate and efficient anomaly detection on multivariate monitoring metrics. Due to the complex system architecture, in Huawei Cloud, the number of monitoring metrics being monitored is extremely large. Existing anomaly detection algorithms either report too many false positives or cannot meet the efficiency requirements, i.e., real-time detection and fast model retraining. Owing to the proposed mechanism of collaborative machines and cost-effective model architecture, CMAAnomaly can achieve superior performance.

B. Lessons Learned

Model Maintenance. Monitoring the accuracy of a deployed model is crucial in its industrial application. It is common that the accuracy of a model decreases over time due to the notorious *concept drift* problem [7]. In our scenario, the accuracy drop can usually happen when a new version of an online service is launched, which

incurs different anomaly patterns in the monitoring metrics. An effective solution to this problem is to retrain the anomaly detection model with newly collected data. At Huawei Cloud, a substantial number of monitoring metrics are generated on a daily basis. Consequently, it becomes necessary for the model to be retrained within a short period of time, allowing for the rapid redeployment of a new version. Thanks to the lightweight design of CMAAnomaly, we are able to establish an efficient pipeline for updating the model. This pipeline encompasses various stages, including data collection, retraining, evaluation, and deployment.

Utilizing Expert Knowledge. On-site engineers often possess decent knowledge about the systems. Such expert knowledge can facilitate model design and development. For example, given thousands of monitoring metrics at hand, they can quickly select the important ones that best characterize the systems’ health status. Moreover, instead of finding dependent monitoring metrics from scratch as done in this paper, engineers can help identify a small set of correlated monitoring metrics first. A more accurate model can then be pursued on top of these monitoring metrics. However, such valuable knowledge is often not well accumulated, organized, and documented. For incident management in cloud systems, automated knowledge extraction [30] has been proposed. Efforts should also be devoted to system troubleshooting based on monitoring metrics.

Building data collection pipeline. For modern software systems, IT operations play a crucial role for system reliability assurance [3], [31], [32], [33], [34]. Since it is data-driven by nature, sufficient and high-quality data are the foundation for model development in many applications, e.g., anomaly detection, failure diagnosis, fault localization. However, common data quality issues include insufficient labels, data noise, etc. To alleviate this issue, a complete and efficient pipeline should be constructed for monitoring data (e.g., monitoring metrics and logs [35], [36], [37]) collection and storage. Specifically, when failure is detected, there should be tools to facilitate the labeling work for engineers. The labeled data should also be properly stored in a database which supports easy query in future. Once there are sufficient labels, possible supervised learning models can be explored.

VII. CONCLUSION

In this paper, we propose CMAAnomaly, an anomaly detection framework for multivariate monitoring metrics based on collaborative machine. Particularly, the proposed collaborative machine can learn the pairwise cross-feature and cross-time interactions between monitoring metrics with linear time complexity. Thus, CMAAnomaly can quickly obtain a big picture of a system’s health status for anomaly detection. We have conducted experiments with three public datasets and one industrial dataset collected from a large online service of Huawei Cloud. For public datasets, CMAAnomaly has achieved an average F1 score of 0.9494, outperforming the existing best approach by 6.77%. Similarly, a 10.68% of accuracy gain is achieved in the industrial dataset. More importantly, for model training and prediction, CMAAnomaly runs up to 20x faster than the baseline methods, demonstrating that CMAAnomaly is capable of meeting the demanding industrial requirements in terms of effectiveness and efficiency. Our framework has been successfully incorporated into the troubleshooting system of the Huawei Cloud. Feedback from on-site engineers confirms its practical usefulness.

VIII. ACKNOWLEDGEMENT

The work described in this paper was supported by the Key-Area Research and Development Program of Guangdong Province (No. 2020B010165002) and the Key Program of Fundamental Research from Shenzhen Science and Technology Innovation Commission (No. JCYJ20200109113403826). It was also supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund) and the National Natural Science Foundation of China (No. 62202511).

REFERENCES

- [1] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu, "Towards intelligent incident management: why we need it and how we make it," in *Proceedings of the 28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (ESEC/FSE)*, 2020, pp. 1487–1497.
- [2] X. Zhang, Q. Lin, Y. Xu, S. Qin, H. Zhang, B. Qiao, Y. Dang, X. Yang, Q. Cheng, M. Chintalapati, Y. Wu, K. Hsieh, K. Sui, X. Meng, Y. Xu, W. Zhang, F. Shen, and D. Zhang, "Cross-dataset time series anomaly detection for cloud systems," in *Proceedings of the Annual Technical Conference, (USENIX ATC)*, 2019, pp. 1063–1076.
- [3] Y. Dang, Q. Lin, and P. Huang, "Aioops: real-world challenges and research innovations," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, (ICSE)*, 2019, pp. 4–5.
- [4] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *CoRR*, vol. abs/2009.07237, 2020.
- [5] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (ESEC/FSE) FSE*. ACM, 2018, pp. 60–70.
- [6] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "Eadro: An end-to-end troubleshooting framework for microservices on multi-source data," *CoRR*, vol. abs/2302.05092, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2302.05092>
- [7] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu, "Adaptive performance anomaly detection for online service systems via pattern sketching," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 61–72.
- [8] J. Liu, S. He, Z. Chen, L. Li, Y. Kang, X. Zhang, P. He, H. Zhang, Q. Lin, Z. Xu *et al.*, "Incident-aware duplicate ticket aggregation for cloud systems," *arXiv preprint arXiv:2302.09520*, 2023.
- [9] J. Liu, Z. Jiang, J. Gu, J. Huang, Z. Chen, C. Feng, Z. Yang, Y. Yang, and M. R. Lyu, "Prism: Revealing hidden functional clusters from massive instances in cloud systems," *arXiv preprint arXiv:2308.07638*, 2023.
- [10] Y. Li, X. Zhang, S. He, Z. Chen, Y. Kang, J. Liu, L. Li, Y. Dang, F. Gao, Z. Xu *et al.*, "An intelligent framework for timely, accurate, and comprehensive cloud incident detection," *ACM SIGOPS Operating Systems Review*, vol. 56, no. 1, pp. 1–7, 2022.
- [11] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining, (KDD)*, 2019, pp. 3009–3017.
- [12] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, (WWW)*. ACM, 2018, pp. 187–196.
- [13] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," *CoRR*, vol. abs/2004.00433, 2020.
- [14] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2250–2267, 2014.
- [15] 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 *Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining, (KDD)*, 2019, pp. 2828–2837.
- [16] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *CoRR*, vol. abs/1607.00148, 2016.
- [17] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the 33rd Applications of Artificial Intelligence Conference, (AAAI)*, 2019, pp. 1409–1416.
- [18] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *CoRR*, vol. abs/1711.00614, 2017.
- [19] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *Proceedings of the 20th International Conference on Data Mining, (ICDM)*, 2020, pp. 841–850.
- [20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.
- [21] S. Rendle, "Factorization machines," in *Proceedings of the 10th International Conference on Data Mining, (ICDM)*, 2010, pp. 995–1000.
- [22] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Söderström, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining, (KDD)*, 2018, pp. 387–395.
- [23] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 Conference on Computer and Communications Security, (CCS)*. ACM, 2017, pp. 1285–1298.
- [24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of 3rd International Conference on Learning Representations, (ICLR)*, 2015.
- [25] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: unsupervised anomaly detection on multivariate time series," in *Proceedings of the 26th SIGKDD Conference on Knowledge Discovery and Data Mining, (KDD)*. ACM, 2020, pp. 3395–3404.
- [26] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *Proceedings of the 8th International Conference on Data Mining (ICDM)*, 2008, pp. 413–422.
- [27] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proceedings of the 6th International Conference on Learning Representations, (ICLR)*, 2018.
- [28] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th Conference on Neural Information Processing Systems 2014, (NeurIPS)*, 2014, pp. 2672–2680.
- [29] H. Wang and D. Yeung, "Towards bayesian deep learning: A framework and some existing methods," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3395–3408, 2016.
- [30] M. Shetty, C. Bansal, S. Kumar, N. Rao, N. Nagappan, and T. Zimmermann, "Neural knowledge extraction from cloud service incidents," *CoRR*, vol. abs/2007.05505, 2020.
- [31] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, (ICSE-SEIP)*, 2019, pp. 111–120.
- [32] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, K. Sui, and D. Pei, "Real-time incident prediction for online service systems," in *Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (ESEC/FSE)*. ACM, 2020, pp. 315–326.
- [33] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang, Y. Wu, F. Zhou, W. Zhang, K. Sui, and D. Pei, "Understanding and handling alert storm for online service systems," in *Proceedings of the 42nd International Conference on Software Engineering, Software Engineering in Practice, (ICSE-SEIP)*, 2020, pp. 162–171.
- [34] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J. Lou, C. Li, Y. Wu, R. Yao, M. Chintalapati, and D. Zhang, "Predicting node failure in cloud service systems," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (ESEC/FSE)*, 2018, pp. 480–490.
- [35] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd International Conference on Automated Software Engineering, (ASE)*, 2018, pp. 178–189.
- [36] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the 22nd Symposium on Operating Systems Principles, (SOSP)*, 2009, pp. 117–132.
- [37] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering, (ICSE)*, 2016, pp. 102–111.