CTF: Anomaly Detection in High-Dimensional Time Series with Coarse-to-Fine Model Transfer

Ming Sun*, Ya Su*, Shenglin Zhang†, Yuanpu Cao†, Yuqing Liu†, Dan Pei*,
Wenfei Wu*, Yongsu Zhang‡, Xiaozhou Liu‡, Junliang Tang‡
*Tsinghua University †Nankai University ‡ByteDance

***Abstract*—Anomaly detection is indispensable in modern IT infrastructure management. However, the dimension explosion problem of the monitoring data (large-scale machines, many key performance indicators, and frequent monitoring queries) causes a scalability issue to the existing algorithms. We propose a coarse-to-fine model transfer based framework CTF to achieve a scalable and accurate data-center-scale anomaly detection. CTF pre-trains a coarse-grained model, uses the model to extract and compress per-machine features to a distribution, clusters machines according to the distribution, and conducts model transfer to fine-tune per-cluster models for high accuracy. The framework takes advantage of clustering on the per-machine latent representation distribution, reusing the pre-trained model, and partial-layer model fine-tuning to boost the whole training efficiency. We also justify design choices such as the clustering algorithm and distance algorithm to achieve the best accuracy. We prototype CTF and experiment on production data to show its scalability and accuracy. We also release a labeling tool for multivariate time series and a labeled dataset to the research community.**

I. Introduction

Anomaly detection always plays an important role in IT infrastructure management, which helps operators identify the misbehaving machines and reduce corresponding revenue loss [1]–[3]. A recent trend is to introduce deep learning (DL) based algorithms for anomaly detection (as a typical scenario of AIOps [4]). Compared with rule-based algorithms, such DL-based ones (*e.g.,* OmniAnomaly [1], LSTM-VAE [5], and DAGMM [6]) have demonstrated the advantages of automation, robustness, and operability (*i.e.,* less dependency on the operators' experience).

However, the DL-based algorithms are facing the challenge of *dimension explosion* when being applied to modern large-scale infrastructures. The dimension increasing is threefold. First, (machine domain) the scale of a modern data center increases fast in the past few years, and it can consist of a few million servers now. Second, (KPI domain) each machine in the infrastructure is monitored at a fixed period with many key performance indicators (KPIs), *i.e.,* 10X KPIs about CPU, memory, and network statistics; each KPI can be formulated as a univariate time series, and thus, each machine can be represented as a multivariate time series (MTS). Third, (time domain) each machine is monitored in a fine-grained frequency, *e.g.,* with a 30-second monitoring interval, there would be 2880 time points to monitor a machine each day.

Wenfei Wu is the corresponding author.

Typical DL algorithms [1], [2], [5]–[11] train an anomaly detection model for each machine. However, they face the scalability issue when being applied to a data center with 1X million machines. Applying the DL algorithms to individual machines and training per-machine models could result in unacceptable time (10X minutes per model $\times$ 1X million machines); training one DL model for all machines would bound the accuracy by nature (one model is not expressive enough for diverse machines).

For anomaly detection of large-scale machines, some solutions propose to cluster machines first [12], and conduct anomaly detection within each cluster. However, the high dimension of the KPI and time domains makes the clustering algorithms inefficient: they need to compute the pairwise distance of 1X million machines' MTS, each of which has 100K dimensions (10X KPIs $\times$ 1X thousand time points), and thus, it could not be complete in practical time.

We observe that a class of unsupervised DL algorithms [1], [5] — Recurrent Neural Network - Variational Auto Encoder (RNN-VAE) — usually transforms the original MTS input into low-dimensional latent representations (details in §II-A), which actually compresses the dimensions in the KPI domain. While these RNN-VAE based algorithms use latent representations to improve the accuracy and robustness (*i.e.,* denoise [1]), we got the inspiration to use the low-dimensional latent representations for clustering to solve the dimension explosion issue. Our approach synthesizes an RNN-VAE based algorithm and a clustering algorithm so that it takes both algorithms' advantages in handling high-dimensional KPIs and large-scale machines.

We further overcome three challenges to build the synthetic framework. First, there is a logical dependency between RNN-VAE model training and the clustering — clustering on the latent representations requires a trained RNN-VAE model to transform the original MTS into the latent representations, but training an RNN-VAE model (before clustering machines) based on the non-clustered large-scale multi-machine MTS can hardly be accurate and efficient (as discussed in paragraph 3). We propose a *coarse-to-fine model transfer* framework, avoiding this dilemma (§II-B).

Second, we still face the challenge in the time domain — the latent representation is still a high-dimensional time sequence (2880 time points per day), which is not friendly for clustering. We propose to transform the time sequence into a distribution, which improves both efficiency and accuracy (§II-B).

Third, we need to make several design choices under the synthetic framework, including a model transfer method (*e.g.,* fine-tuning strategy), a clustering algorithm (*e.g.,* DBSCAN, Hierarchical Agglomerative Clustering), and a distance metric (*e.g.,* KL divergence, JS divergence, or Wasserstein distance). We carefully reason the properties of the operational dataset and the advantages of each option and make the design choice (§II-C). Specifically, we are the first to apply a fine-tuning strategy during model transfer for RNN-VAE models, validating its usefulness in improving efficiency and accuracy. We also validate the design choices in experiments (§V-C).

The final synthetic framework is named CTF. CTF's offline model training has four steps. First, it samples an all-machine dataset to train a coarse-grained model; second, it uses the coarse-grained model to transform per-machine MTS to low-dimensional latent representations; third, it clusters machines based on the distribution of latent representations; finally, it copies the coarse-grained model to each cluster and fine-tunes a per-cluster model. CTF's online anomaly detection uses per-cluster models to generate an anomaly score for per-machine real-time MTS and uses POT [13] to decide the threshold whether to report an anomaly. Moreover, CTF is generic to unsupervised DL algorithms whose model learns a low-dimensional time-specific latent representation [8].

We architect the CTF framework and use it for a state-of-the-art well-performed RNN-VAE algorithm OmniAnomaly [1]. We deploy CTF in an Internet company; the whole system preprocesses monitoring data, executes offline model training, and performs online real-time anomaly detection. We also build a labeling tool for network engineers to label MTS data as the ground truth and verify the accuracy of CTF's results. Our evaluation shows that when applying CTF to OmniAnomaly, CTF can reduce the model training time from about two months (estimated by the number of machines $\times$ per model training time) to 4.40 hours for one hundred thousand machines within a data center using six computing servers; it achieves an F1-Score (accuracy) of 0.830, with only 0.012 performance loss (compared to per-machine models). The contributions of this paper can be summarized as follows.

- We propose a coarse-to-fine model transfer based framework, which can perform anomaly detection on a huge operation dataset with high dimensions on machine, KPI, and time domains. Its core techniques are as follows.
  - It first synthesizes model training and machine clustering and accelerates both of them.
  - It first uses the distribution of the latent representations for clustering, which accelerates the pairwise distance computation.
  - It first applies a fine-tuning strategy to RNN-VAE models, validating its benefits to both accuracy and efficiency.
- We implement and evaluate CTF with a state-of-the-art anomaly detection algorithm on a large-scale dataset from a top global Internet company, demonstrating CTF's effectiveness and scalability in practical infrastructure.

TABLE I
DETAILED INFORMATION OF THE 49 KPIS OF MACHINES WHICH ARE CLASSIFIED INTO FIVE CATEGORIES.

KPI categories / counts	KPIs
CPU / 15	CPU idle rate, CPU busy rate, CPU utilization at user or system level, CPU load, etc.
Memory / 10	Memory usage or free or available rate, etc.
Sockets / 6	Sockets established or closed or orphaned, etc.
UDP / 7	count of UDP packets sent or received, count of UDP buffer errors sent or received, etc.
TCP / 11	TCP retransmisstion rate, TCP listen drops, TCP listen overflows, TCP delayed ACK locked, etc.

TABLE II
SYMBOLS AND THEIR CORRESPONDING MEANINGS AND VALUES.

Symbols	Meanings	Values
M	No. of machine entities, indexed by i	10^5
L	No. of KPI vector's dimensions, indexed by j	49
T	No. of collected KPI vectors during a period (*e.g.,* 13 days in our dataset), indexed by t	37440
K	No. of clusters (§III-B)	50
C	No. of a latent representation's dimensions (§III-B)	3
T_0	The length of time window in anomaly detection	100
T_m	The time of model training per machine	315s
T_f	The time of feature extraction per machine	0.3s
$\mathbf{x}_{i,j,t}$	The i-th machine's j-th KPI at time t	—
$\mathbb{P}_i$	The latent representation distribution of the i-th machine entity	—

- We release a labeling tool for MTS* and a labeled dataset† to the community for research and deployment.

II. BACKGROUND

A. Problem Statement

Data Format. In the infrastructure, each machine is monitored on several KPIs (in Table I, about CPU, memory, etc.), and these KPIs are collected at a fixed interval (*e.g.,* 30s in our system). Thus, each machine's behavior is formulated as an MTS, which is also named a *machine entity*.

One time monitoring of the KPIs would generate a vector of L dimensions (Table I). The monitoring system would generate T KPI vectors during a period (*e.g.,* $T = 2880$ for one day, and $T = 20160$ for one week). We denote $\mathbf{x}_{i,j,t}$ as the i-th machine's j-th KPI at time t. Then, the i-th machine entity $\mathbf{x}_{i,:,:}$ is a $L \times T$ matrix ($\mathbf{x}_{i,:,:} \in \mathbb{R}^{L \times T}$, and symbols in Table II). Fig. 1 visualizes a few examples of machine entities.

An anomaly refers to KPI vector at a time point that significantly differs from the normal data [1], [2]. We use T_0 to denote the number of consecutive KPI vectors within a time window. For example, $\{\mathbf{x}_{i,:,t-T_0+1}, \mathbf{x}_{i,:,t-T_0+2}, ..., \mathbf{x}_{i,:,t}\}$ denotes the KPI vectors at time t. We also name these T_0 KPI vectors as a "data instance" in the following context.

RNN-VAE Based Algorithms for Anomaly Detection. In this paper, we focus on RNN-VAE based anomaly detection algorithms, which are unsupervised Deep Learning algorithms. They need less operators' experience compared with

*https://github.com/NetManAIOps/label-tool
†https://github.com/NetManAIOps/CTF_data

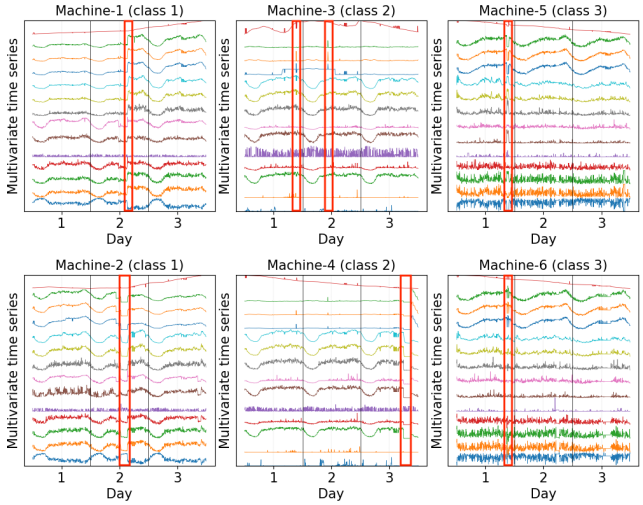

Fig. 1. Examples of machine entities from three classes, each of which is a 14-metric 3-day-long multivariate time series. The anomalous regions are marked in the rectangles.

statistical-based solutions [14], need no manual data labeling compared with supervised solutions [15], and outperform other unsupervised solutions (*e.g.,* DAGMM [6]) [1] (details in §VI).

In a dataset, most KPI vectors are usually normal, and minority ones are anomalous (*i.e.,* outliers). RNN-VAE based algorithms train a model to remember the features of majority and denoise the minority. For time series modeling, a data instance (containing the historical values) is input to the model for understanding the last KPI vector, and the model would reconstruct another data instance as output. After training, as the model remembers the majority (normal points) features, the model output represents a normal behavior — if the input significantly differs from the output, the last KPI vector is identified as an anomaly; otherwise, it is normal.

RNN-VAE models have an encoder and a decoder (Fig. 2). At a time point, a KPI vector is input to the encoder. The encoder uses RNNs (storing states from predecessor KPI vectors) to extract the temporal features and dense layers to compress them to a latent space (*i.e.,* from $\mathbf{x_t}$ to $\mathbf{z_t}$). The decoder reverses the process — it uses RNNs to recover the sequence in the latent space and dense layers to recover the dimensions (*i.e.,* from $\mathbf{z_t}$ to $\mathbf{x'_t}$). The probability density of the input (*i.e.,* $\mathbf{x_t}$) in the output's distribution (*i.e.,* $\mathbf{x'_t}$ distribution) is defined as the reconstruction probability. The loss function used for model training, the evidence lower bound (ELBO), is composed of the reconstruction probability and regularization [16]. Its mathematical format is as follows, and the first term corresponds to the reconstruction probability.

$$\mathcal{L}(\mathbf{x_t}) = \mathbb{E}_{q_\phi(\mathbf{z_t}|\mathbf{x_t})}[\log p_\theta(\mathbf{x_t}|\mathbf{z_t})] - D_{KL}[q_\phi(\mathbf{z_t}|\mathbf{x_t})||p_\theta(\mathbf{z_t})]$$

During the training stage, data instances are input to an RNN-VAE algorithm, and it trains the model by maximizing ELBO. During the prediction stage, a real-time data instance is an input to the RNN-VAE model, and the reconstruction probability is used to judge whether the current KPI vector is an anomaly.

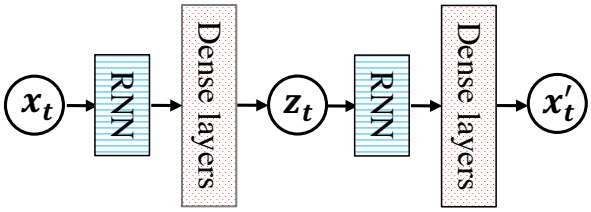

Fig. 2. The network architecture of RNN-VAE models at time t. Latent representation $\mathbf{z_t}$ and reconstruction $\mathbf{x'_t}$ are stochastic variables. RNNs are frozen, and dense layers are fine-tuned when training fine-grained models.

Scalability Challenge. As discussed in §I, the dimensions in the domains of machine, KPI, and time explode. The dimensions and their values in operation are listed in Table I and Table II. Such high dimensions make the following three approaches with existing solutions hard to scale. (1) As training a model usually takes 10X minutes, training individual models for each machine entity would cost about two months (*i.e.,* 10X minutes per model $\times$ 1X million machines). (2) Using a subset of the whole machine's population (*e.g.,* sampling) would bound the training time, but training one model to represent and detect diverse machine entities can hardly achieve accuracy. Fig. 1 shows that different machines have different anomaly patterns (*e.g.,* there are short-term anomalies in Machine-3, Machine-5, and Machine-6, a long-term anomaly in Machine-2, and "stepped" anomalies[‡] in Machine-1 and Machine-4), and one model can hardly describe the diverse behaviors accurately. (3) Naively clustering machines and training per-cluster models face the difficulty in the clustering stage because the pairwise distance computation in the clustering needs to handle high-dimensional data in KPI and time domains (about 100K dimensions).

Goal. Thus, our goal is to devise a framework for anomaly detection in high-dimensional time series, and with the framework, the anomaly detection should be complete (especially the training time) within an acceptable time and sacrifice limited accuracy.

B. Intuition and Analysis

Intuition. We observe that the RNN-VAE based algorithms could compress each high-dimensional KPI vector (L) into a low-dimensional latent representation (C) using the dense layer, a typical structure from VAE.

With the KPI domain compressed, pairwise distance computation on the latent representations $\mathbf{z_t}$ could be faster than that on the original input $\mathbf{x_t}$. Thus, we take the approach of clustering machines and training a per-cluster model. There are still three more challenges to overcome for this approach.

Challenge 1: The mutual dependency between the clustering and the model training. Clustering on the latent representations depends on a trained model to transform $\mathbf{x_t}$ to $\mathbf{z_t}$, but without clustering, we face the dilemma of training per-machine models (costly) and training one model for all machines (inaccurate).

[‡]KPIs' distribution changes significantly.

Solution (reducing M and reusing partial results): We design a synthetic framework, where we consider the efficiency in each step and gradually improve the total accuracy. (1) We first sample the all-machine dataset to pre-train a coarse-grained model. (2) Then we use the coarse-grained model to transform per-machine MTS to the latent representations. (3) And then, we use the distribution of the latent representations to classify machines into K clusters. (4) Finally, we transfer the coarse-grained model to each cluster and fine-tune the per-cluster fine-grained models. Clustering machines in Step 3 reduces M models to K models, and model transfer in Step 4 reuses the pre-trained model, both of which improve the efficiency.

Challenge 2: The high dimension of the time domain. Even each KPI vector is compressed, each machine entity is still a long series of latent representations. If the clustering algorithm computes the distance of two long series, the execution time is still not practical.

Solution (reducing T): We sample the latent representation sequence to get distribution and use the distribution for distance computation in clustering. Specifically, we use Wasserstein distance [17] to compute the pairwise distance. Using distribution to represent a machine entity is intuitive: the whole company has relatively fixed businesses to run on the machines, and each machine usually executes one kind of business (corresponding programs); thus, samples can represent the whole time series, and distribution is more suitable than a temporal sequence of latent representations for clustering (analysis in §III-B).

Challenge 3: The design choices of the neural network (NN) training methods. We transfer the coarse-grained model to train per-cluster models, but model transfer has several choices, *e.g.,* training all NN layers of the old model in a new dataset, training partial layers of the old model on the new dataset (*i.e.,* fine-tuning). We should make the right choice to guarantee efficiency and accuracy (or make the tradeoff).

Solution (faster training): Inspecting the NN architecture in Fig. 2, we find that the RNN layers are shallow and deterministic, which extract the general time-series features. In contrast, the dense layers are deep and stochastic for learning better representations of a new dataset. Thus, we conduct the first fine-tuning strategy for RNN-VAE networks: freezing RNNs for feature generalization [18] [19] and tuning the dense layers for better latent representation and reconstruction. It further saves the model training time (freezing RNNs) and improves accuracy (feature generalization).

C. Preliminaries

The text above briefly describes concepts of RNN-VAE used in CTF; we further list details of the other methods in CTF.

Wasserstein Distance. Most clustering algorithms need to compute pairwise distances. In CTF, a machine entity is transformed into distribution in §II-B; thus, we need a metric to measure the distance between distributions. Wasserstein distance is a suitable choice for CTF. Its intuition is to measure the least distance of "moving" one distribution to the other. Compared with other choices (*e.g.,* Kullback-Leibler divergence or Jensen-Shannon divergence), Wasserstein distance is particularly useful when there is less or no overlap between two distributions (the other two would degenerate to meaningless constants). Its mathematical format is below [17].

$$W(\mathbb{P}_1, \mathbb{P}_2) = \inf_{\gamma \in \prod(\mathbb{P}_1, \mathbb{P}_2)} \mathbb{E}_{(\mathbf{z}_1, \mathbf{z}_2) \sim \gamma}[||\mathbf{z}_1 - \mathbf{z}_2||],$$

where $\mathbb{P}_1$ and $\mathbb{P}_2$ are the two distributions of random variable z_1 and z_2, $\prod(\mathbb{P}_1, \mathbb{P}_2)$ denotes the set of all possible joint distributions, and $\gamma(\mathbf{z}_1, \mathbf{z}_2)$ is one distribution in the set whose marginals are $\mathbb{P}_1$ and $\mathbb{P}_2$.

Hierarchical Agglomerative Clustering. In CTF, we adopt hierarchical agglomerative clustering (HAC) [20]. Based on the distance matrix, HAC iteratively merges the closer pair of clusters and moves up the hierarchy until all machine entities are merged into one cluster. The final clustering result can be determined according to the number of required clusters or per-cluster observations. Compared to other clustering algorithms DBSCAN [12] and K-medoids [21], the HAC algorithm has three advantages. First, it needs no initial parameters (*e.g.,* the number of clusters or distance thresholds). Second, it is not sensitive to the distance measurement algorithms because it clusters on the rank of distances rather than the value. Third, the relationships among different hierarchies are apparent, so it is convenient for us to visualize the clustering results.

Model Transfer. If a model is trained on an old dataset and needs to be applied to a new dataset, the model needs to be further trained using the new dataset. This process is called model transfer. Model transfer can have the model preserve previous training results to avoid training from scratch and adapt to a new dataset to improve accuracy [22]. In this paper, we use the whole dataset (all machine entities) as the old dataset to get a coarse-grained model and the per-cluster dataset to fine-tune a per-cluster fine-grained model.

III. Design

We apply the CTF framework to an RNN-VAE algorithm, and it has three steps — data preprocessing, offline model training, and online anomaly detection. The offline model training synthesizes the methods in §II-B as a whole workflow.

A. Data Preprocessing

In the infrastructure, each machine is monitored periodically (30 seconds), and the per-machine KPIs are collected and stored. These raw data cannot be used directly in the later model training and anomaly detection, and they need to be preprocessed with two steps.

Filling in missing data. In practical operation, some data could be missing because of the problems of data collection or transmission in the monitoring system. When a value is missing, we use its previously observed value to fill in its position.

Data normalization. Different KPIs have different units (*e.g.,* GB for memory, % for CPU) and different ranges. DL algorithms usually perform better when the values in different dimensions are within an approximate range. Thus, we conduct

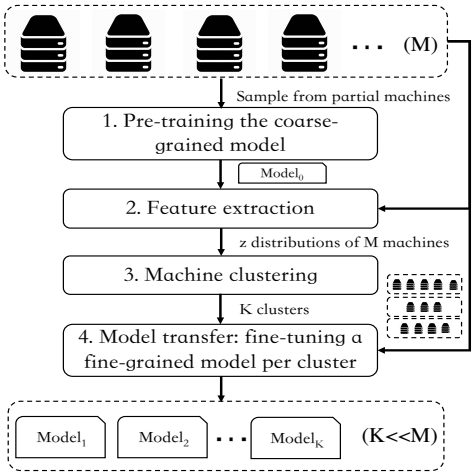

Fig. 3. The workflow of CTF's offline model training.

data normalization. In detail, the i-th machine's j-th KPI at time t is normalized by the mean and standard deviation of the same machine and KPI dimension ($\mu_{i,j}$ and $\sigma_{i,j}$), *i.e.,*

$$\hat{\mathbf{x}}_{i,j,t} = \frac{\mathbf{x}_{i,j,t} - \mu_{i,j}}{\sigma_{i,j}},$$

where $\mu_{i,j}$ and $\sigma_{i,j}$ are updated periodically using recently collected data.

B. Offline Model Training

The synthetic model training has the following four steps. Fig. 3 depicts the workflow of the offline training.

Step 1: Pre-training Coarse-grained Model M_0. CTF pre-trains a coarse-grained model for clustering and model training. As the whole dataset contains $M \times T$ data instances, CTF samples a portion of them to train the model. CTF follows two principles to make the samples representative of the whole dataset.

First, we consider the extra physical attributes of machines, *e.g.,* physical racks, running applications, and machine models, and sample data instances from each group of machines. Second, in the time domain, we uniformly randomly sample data instances from each machine's time series.

The sampled data instances are fed into an RNN-VAE algorithm for model training; the parameters in the RNN-VAE model are initialized and updated and converge to a coarse-grained model M_0. This pre-training is efficient because the sampled dataset is much smaller (0.0001X) than the whole original dataset.

Step 2: Feature Extraction. With M_0, each machine entity is transformed into a briefer representation distribution in the latent space, named $\mathbf{z_t}$ distribution. Each machine has T KPI vectors, and each KPI vector corresponds to a data instance. For each machine entity, CTF first samples a portion of the KPI vectors from the whole time series, and then uses the encoder of the RNN-VAE model to transform the KPI vector to its latent representation (a vector of size C, and $C < L$, see Table II). Thus, a per-machine KPI vector series is transformed into distribution of fewer low-dimensional latent representations, *i.e.,* $\mathbf{z_t}$ distribution.

There is a subtle argument that we should pay more attention to the distribution of the latent representations than the temporal order of that. For infrastructure operation data, the distribution is more suitable than time series. For example, machine-one may execute tasks A and B sequentially, and machine-two executes the two tasks in reverse order; in practice, the two tasks are probably affiliated to the same application/service, the two machines should belong to the same cluster; in this case, two time series would be different but their distributions would be the same. Furthermore, our experimental results §V-C validate this hypothesis.

Step 3: Machine Clustering. The machine clustering is executed on the machines' latent representation distributions, *i.e.,* their $\mathbf{z_t}$ distributions. First, CTF computes the pairwise Wasserstein distance of $\mathbf{z_t}$ distributions among machine entities and gets a distance matrix of all machine entities. The distance measures the similarities between entities — closer distributions should more likely belong to the same class. Then CTF uses HAC to cluster machines based on the distance matrix, and each machine would finally belong to one specific cluster.

In our practice, the number of machine entities is still too large to compute the pairwise distance ($M = 10^5$), and we further improve the clustering algorithm based on [12]. We randomly sample a subset (about 10K) of the machine entities, and run HAC on the subset to get the classes. For each of the remaining machine entities, we compute the distances from it to all clustered machine entities, calculate the average distance within each class, and assign it to the class with the least average distance.

Step 4: Model Transfer. The coarse-grained model M_0 is copied to each cluster; within each cluster, the model is further trained (*i.e.,* fine-tuned) using the per-cluster data. The data sampling is the same as that in Step 1, but the training procedures are customized in this step.

In the model fine-tuning, the RNN layers are frozen, and the model parameter update is only on the dense layers. The reason is that RNN layers are shallow and deterministic, and thus, they extract general time-series features in the coarse-grained model, which could contribute to the model generalization (similar to BERT [23] for text feature extraction and VGG [24] for image feature extraction); but the dense layers are deep and stochastic, and thus, they need to be fine-tuned to learn a better latent representation for each cluster. Moreover, freezing RNN layers also saves the training time in this step. Finally, the M_0 would evolve to individual models M_i for each cluster.

C. Online Anomaly Detection

In the runtime, each machine's data instance is input to the machine's cluster model, and the model outputs a reconstruction data instance. As the model remembers the majority behavior, the output indicates the "normal" behavior of machines. The reconstruction probability between $\mathbf{x_t}$ and $\mathbf{x'_t}$ indicates the

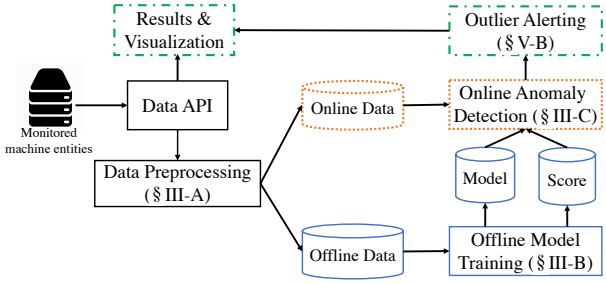

Fig. 4. System architecture. The blue modules with solid lines denote offline model training, and the orange modules in dash lines represent online anomaly detection, while the green ones with dot-dash lines show the notification system.

probability of the input KPI vector approaching the normal behavior. A smaller reconstruction probability would imply that the input is anomalous.

In operation, the reconstruction probability is also named *anomaly score*. The online module of CTF compares the anomaly score with a threshold — for each real-time KPI vector, an anomaly score lower than the threshold would be considered as an anomaly; otherwise, it is normal.

In the threshold selection, CTF uses Peaks-Over-Threshold (POT) [13]. Given a set of historical anomaly scores (also called samples in the following text), POT has two steps (details in [13]): first, POT filters out the samples below a certain "low quantile" of the whole population, fits these samples with a Generalized Pareto Distribution (GPD), and gets the GPD function; second, it uses the function and an anomaly quantile among the whole population (denoted as q) to identify the threshold.

Compared with other choices of threshold selection methods (*e.g.,* state-based thresholding [5], Dynamic Error Thresholds [7]), POT has three advantages. First, it makes no assumption of the whole population's distribution, because it filters out the low quantile samples and fits them with GPD. Second, it requires only two parameters (low quantile and q), which is robust in parameter selection. Third, POT is efficient in practical operations (0.015 seconds for each machine's one week's historical score).

In CTF operation, we apply the same POT parameters to each cluster and use the historical scores of each machine within the cluster to fit the GPD. Because the statistical features of machines' scores in the same cluster are similar, they can share the same parameters, avoiding unnecessary parameter tuning.

IV. Implementation

A. System Architecture

The architecture of CTF with an anomaly detection algorithm is shown in Fig. 4. CTF fetches monitoring data via the Data API and preprocesses the KPI data, as described in §III-A. The offline module runs the logic in §III-B and stores the per-cluster model and historical scores. The online model runs the logic in §III-C and outputs the result (anomalous or not) to the notification system (alerting and visualization). The online and offline modules are implemented in Python with Tensorflow, which has about 500 lines of code (mainly about the CTF framework); the visualization system is implemented based on Grafana.

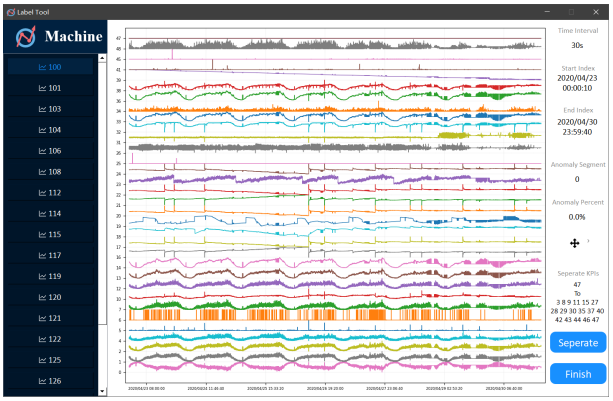

Fig. 5. The interface of the labeling tool.

B. A Labeling Tool for Experiment

While CTF with an anomaly detection algorithm is unsupervised, which does not need labeled data, we still need ground truth (*i.e.,* a dataset with data labeled anomalous or normal) to validate its accuracy. We develop a labeling tool with graphical user interface (GUI) for network engineers to label a multivariate time series. The labeling tool is illustrated in Fig. 5, and it has the following functions. (1) It can load, visualize, drag, zoom in/out the time series so that the user can overview the shape of the whole sequence and locate the details of a segment. (2) It can fold and unfold a few KPI dimensions, providing a better view to the users. (3) It can label or cancel anomalies. Users can select the beginning and end of an interval and save it as an anomaly. (4) It collects and updates statistics of anomaly intervals (*e.g.,* count, percentage) in real-time. This labeling tool is implemented in Python with PyQt and matplotlib, with about 600 lines of code.

V. Evaluation

Combining with a state-of-the-art well-performed RNN-VAE based algorithm (*i.e.,* OmniAnomaly [1] in this paper), we set up experiments (dataset and environment) to evaluate CTF's performance (scalability and effectiveness). We compare CTF with its variants to validate its synthetic framework, compare CTF with other baselines to validate the design choices (*e.g.,* clustering objects, clustering algorithms), and show the deployment results.

A. Experiment Setup

Dataset. We collect a dataset from a top global Internet company, where geo-distributed data centers serve global users. The businesses running on the infrastructure are typical Internet services (*e.g.,* news, advertisement, videos).

The experimental dataset contains 533 machine entities, and each is monitored with 49 KPIs. KPIs are collected every 30s spanning 13 days (from April 18th to April 30th). In the

TABLE III
THE EXECUTION TIME OF EACH STEP UNDER DIFFERENT NUMBERS OF MACHINE ENTITIES.

M	533	10^3	10^4	10^5	10^5(6 servers)
Pre-training	5493	5493	5493	5493	5493
Feature extraction	166	311	3113	31130	5292
Clustering	3	6	232	576	576
Model transfer	2238	2238	4475	22375	4475
Total	7900	8048	13313	59574	15836
Average	14.822	8.048	1.331	0.596	0.158

experiment, we use the first five days' data for training and the latter eight days' data for testing.

Performance Metrics. The online module of CTF would output the detection result, which is compared with the labeled results. We denote false positives (FP) as the normal KPI vectors that are reported as anomalies by the algorithm, false negatives (FN) as anomalous KPI vectors reported as normal ones, true positives (TP) and true negatives (TN) as the anomalous or normal KPI vectors reported correctly. We use Precision, Recall, and *F1-Score* (F1 for short) to evaluate the anomaly detection accuracy, where

$$Precision = TP/(TP + FP), Recall = TP/(TP + FN)$$

$$F1 = 2 \times Precision \times Recall/(Precision + Recall)$$

F1 is a performance indicator with considering both Precision and Recall, and it is more important than Precision and Recall. In this paper, all scores (*i.e.,* F1, Precision, and Recall) are the average of all machine entities. In the experiment, we also measure the *execution time* of CTF and other methods, which denotes the efficiency. All the time is in seconds.

Hyper-parameters of models. In the experiment, we keep the original hyper-parameters of OmniAnomaly as [1]. The offline module of CTF would sample 100 machine entities and 10% data instances per-machine series in Step 1, and 30 and 10% in Step 4. The number of clusters in Step 2 is 5. In the offline module, we use empirical values of low quantile 0.01, 0.02, and 0.03 and $q = 10^{-5}$.

Environment. All the experiments are run on a server with a 64-core Intel(R) Xeon(R) Gold 6130 CPU @2.10GHz and 376GB RAM.

B. Overall Performance

Scalability. We deploy the CTF framework, vary the number of machine entities, and measure the execution time of each step, and the results are in Table III. The numbers of clusters (an empirical value) for 10^3, 10^4, and 10^5 machines are 5, 10, and 50, respectively. For the case of 10^5 machines, we use six CPU servers for the model training (Step 2 and 4) and list the accumulated time and the actual parallel execution time of six servers. We got the following observations.

First, the pre-training time is a marginal fixed time (5493s). Second, the feature extraction time is proportional to the number of machines (311s for 10^3 and 3113s for 10^4), but this step can be paralleled among machine entities. On average, it takes 0.3s for each machine. Third, the clustering algorithm needs first to compute the pair-wise distance and then run HAC. However, this step is significantly smaller (0.1X, *e.g.,* 576s v.s. 4000+s for 10^5) than other steps. Using $\mathbf{z_t}$ distribution in the latent space to represent a machine entity is efficient in accelerating the clustering speed (comparison with other methods are in §V-C). Finally, the fine-tuning time is proportional to the number of clusters (Table III), but it can also be paralleled among clusters. Overall, CTF is scalable to process data of the size of a data center (about 1X million machines).

TABLE IV
F1, PRECISION, AND RECALL SCORES OF CTF WITH OMNIANOMALY WITHOUT AND WITH ALERTING.

Methods	F1	Precision	Recall
Without alerting	0.830	0.785	0.881
With alerting	0.892	0.907	0.877

TABLE V
COMPARISON WITH MODEL VARIATIONS.

Methods	F1	Precision	Recall	Training time
CTF	0.830	0.785	0.881	7900
One model/machine[a]	0.842	0.820	0.864	168150
One model for all	0.796	0.791	0.802	5493
CTF w/o transfer	0.798	0.758	0.843	8413

[a] We evaluate 10% machine entities in this method.

Effectiveness and Operational Tuning. Applying CTF with OmniAnomaly to the dataset, we got F1/Precision/Recall of 0.830/0.785/0.881, which can significantly reduce the operators' workload.

In practical operation, we observe that some anomalies lasted for a short time (*e.g.,* less than 2 minutes). Thus, we tune alerting policy cross-time: at least N consecutive anomalous points will be considered as anomalies (*e.g.,* $N = 5$ in our scenario). With this alerting policy, the Precision of CTF is further improved (F1 from 0.830 to 0.892, see Table IV).

Validating the choice of the Synthetic Framework. CTF uses a synthetic framework to conduct machine clustering and model training. We compare it with a few other model variations and list the results in Table V. "One model/machine" denotes training one model for each machine. We use all five days' data for model training because the sampled data is insufficient. This method can fit the machine's behavior to the best and is the most effective one, but its training time is not acceptable. In the experiment, we could evaluate only 10% machine entities on our test dataset, and the training time of 100,000 machines is estimated to be about two months. In practice, it could be hardly paralleled on each machine because it costs many computing resources and may affect more important business tasks. The time complexity of this approach is $O(M \cdot T_m)$, where T_m is per-model training time, and that of CTF is $O(M \cdot T_f) + O(K \cdot T_m)$, where T_f is feature extraction time, and K is the number of clusters. With $T_f \ll T_m$ and $K \ll M$, the total time is significantly reduced. In practice, CTF needs 15,836s (*i.e.,* 4.40h) with six servers

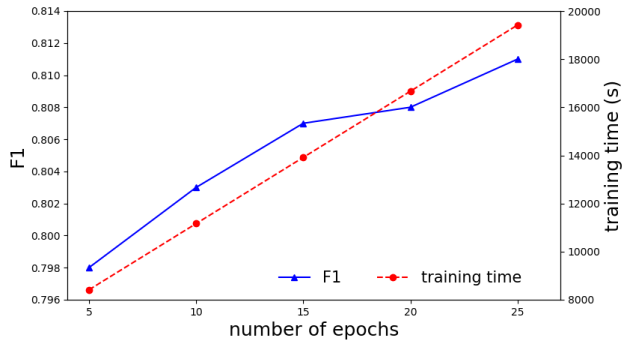

Fig. 6. F1 and training time under different numbers of epochs for CTF w/o transfer.

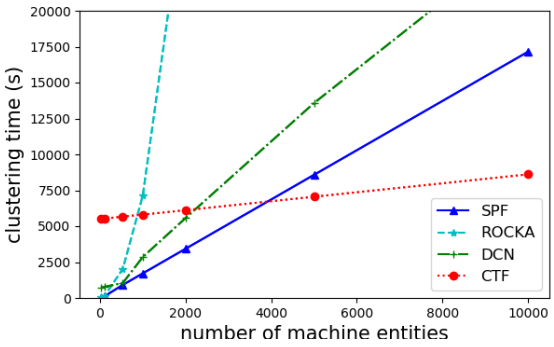

Fig. 7. The clustering time of CTF and baseline algorithms.

to train models for 100,000 machines (0.158s per machine).

"One model for all" indicates training one model for all machine entities. It is equivalent to the pre-trained coarse-grained model M_0 in CTF. It costs the least training time, but its anomaly detection accuracy is the lowest. Because the machine entities are diverse, one single model can hardly extract features from their mixed patterns (the latent $\mathbf{z_t}$ distribution and reconstruction $\mathbf{x'_t}$ may be inaccurate).

"CTF without (w/o) transfer" is similar to CTF in the workflow, and the difference is in Step 4: each cluster retrains a model from scratch (not from M_0). Note that CTF freezes the RNNs in Step 4, but "CTF w/o transfer" needs to train RNNs from scratch. "CTF w/o transfer" performs worse in both effectiveness and efficiency compared with CTF. Fig. 6 shows the F1 and training time of "CTF w/o transfer", and we can observe that more epochs can achieve higher F1 but need more time. However, even if training 25 epochs (19,417 seconds), the F1 is still lower than that of CTF. Because in CTF, the pre-trained M_0 in CTF captures the general features [19], its RNNs do not need to be trained further, and CTF benefits from model generalization; but in "CTF w/o transfer", the initialization of network parameters is easier to disturb the per-cluster model in Step 4 (ineffectiveness), and training RNNs costs extra time (2238s v.s. 2751s).

C. Validating Design Choices

Choice of Clustering Objects. The clustering object of CTF is $\mathbf{z_t}$ distribution, and we compare this choice with other state-of-the-art algorithms: SPF [25] and ROCKA [12] with the original input, and DCN [26] with AE latent representation (§VI). They cannot handle matrix input, and we flatten the matrix to a vector [27]. After clustering, we train one OmniAnomaly [1] model within each cluster. Fig. 7 represents the clustering time and Fig. 8(a) shows the effectiveness.

Overall, CTF's clustering object ($\mathbf{z_t}$ distribution) outperforms the other three on both efficiency and effectiveness.

(1) ROCKA's execution time increases at a square of the number of machines, and DCN, SPF, and CTF increase linearly. CTF has a marginal cost (*i.e.,* the coarse-grained model training time), but the time to machine ratio is small. When the number of machines is 5000 or more, CTF costs the least clustering time. Thus, CTF is able to handle the anomaly detection of millions of machines.

(2) CTF has a higher F1 score than the other three algorithms. SPF and ROCKA are not resistant to noises in the dataset; DCN theoretically could resist noises, but it is hard to fully converge (its NN has a few billion parameters), and thus its latent representation may be inaccurate. CTF's $\mathbf{z_t}$ distribution is robust to noises and low-dimensional to converge in the experiment, which is thus the most accurate.

Choice of Distance Measures. We compare CTF's Wasserstein distance algorithm with other distance measures: Kullback-Leibler (KL) divergence (between distributions), Jensen-Shannon (JS) divergence (between distributions), and mean squared error (between sequences). We replace the distance algorithm with the other three and run CTF experiments again. The results are shown in Fig. 8(b).

Wasserstein distance outperforms the other three. Compared with MSE, we conclude that distribution is a better representation than sequence for machine clustering (as analyzed in §III-B). Compared with KL and JS divergences, we conclude that the distance between neighboring values (in Wasserstein) is a better measure than the binary 0/1 indicator of whether two values overlap (in KL and JS divergences).

Choice of Clustering Algorithms. We also compare the HAC algorithm with other popular clustering algorithms: K-medoids and DBSCAN. We replace HAC with each of them in CTF, fine-tune their parameters, and show the results in Fig. 8(c). HAC performs best because it is based on the rank of distances and robust to extreme values. However, the other two algorithms are sensitive to extreme values in the distances' distribution and parameters, especially for DBSCAN. Thus, the HAC algorithm is more suitable in real scenarios.

VI. Related Work and Discussion

A. Anomaly Detection Algorithms

The anomaly detection algorithms for time series can be roughly classified into three classes — statistical-based algorithms and machine learning algorithms (supervised and unsupervised) [28]. They have different advantages in scalability, operability, efficiency, and accuracy.

Statistical-based algorithms rely on extensive domain knowledge (*e.g.,* parameter selection). A classic one is K-Sigma algorithm [14], which used the values deviating K times

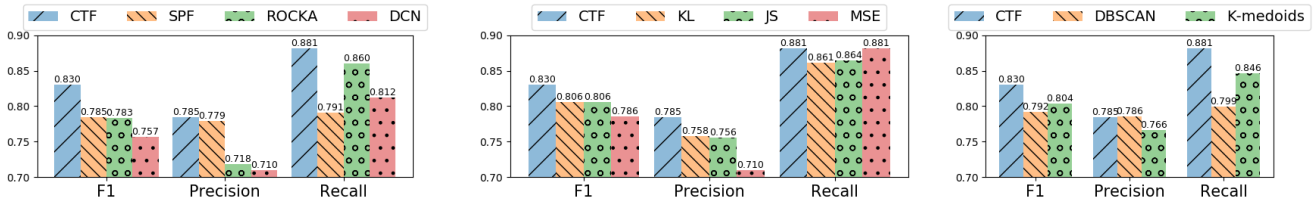

Fig. 8. F1, Precision, and Recall scores under (a) algorithms for different clustering objects, (b) different distance measures, (c) different clustering algorithms.

the standard derivation from the average as thresholds and conducted anomaly detection. This kind of algorithms can be efficient and scalable, but usually need experienced operators to set parameters (*e.g.,* anomaly threshold).

Supervised algorithms [3], [15], [29] require manual labeling. The famous one is EGADS in Yahoo [15], which used a collection of machine learning methods and threshold selection algorithms for anomaly detection on large-scale univariate time series. When there are no tremendous labors to devote to labeling, this class of algorithms can hardly be applied to millions of machines.

Unsupervised algorithms [1], [5]–[9] aim to find "outliers" among all data instances. Among them, RNN-VAE based algorithms, combining RNN for time-series feature extraction and VAE for learning a robust latent representation, demonstrate the best potential [1]. CTF is a framework that scales them to millions of machines.

B. Clustering Algorithms

The clustering algorithms can be categorized according to their clustering objects.

Cluster on univariate time series. ROCKA [12] aims to detect anomalies for large-scale univariate time series. It clustered them using DBSCAN with normalized cross-correlation distance and trained one anomaly detection model DONUT [2] per cluster. Symbolic Pattern Forest (SPF) [25] is the state-of-the-art algorithm on univariate time series clustering. It extracted the symbolic patterns from time series, clustered on them using different trees, and finally assembled the clustering results. They do not contain mechanisms to compare the importance of KPI dimensions in an MTS and cannot be directly applied.

Cluster on multivariate time series (MTS). Multi-dimensional dynamic time warping (MDDTW) [30] measures the shape similarity among MTS. Then a clustering algorithm (*e.g.,* DBSCAN) is combined with it for clustering. However, its computational complexity of one pairwise distance is $O(LT^2)$, higher than $O(LT \log T)$ of ROCKA [12]. Thus, it costs more time than $\mathbf{z_t}$ distribution clustering (§V-C).

Cluster on the latent representation (*e.g.,* z) in Auto Encoder (AE) models. DCN [26] combines AE for dimensionality reduction and K-means on the encoded $\mathbf{z}$ for clustering and trains the whole model end-to-end. CTF is inspired by this approach. It uses distribution instead of the whole latent representation, improving both effectiveness and efficiency.

C. Model Transfer Strategies

During model transfer, the old model is trained on a new dataset. This process has several choices, *e.g.,* continuing to train all NN layers, training partial layers (namely fine-tuning, *e.g.,* Convolution layers or RNN layers). Jason, et al. [18] studied the transferability of different layers in deep neural networks. They find that the shallow layers contain more general information and benefit for diverse data. In contrast, the deep layers are specific, which need more fine-tuning to fit new tasks. Moreover, Long, et al. [19] applied this method to image classification. In the infrastructure operation domain, we are the first to exercise the fine-tuning on RNN-VAE models. The evaluation results show that this is the right choice in our scenario.

D. Lessons Learned

We learn the following lessons. (1) Our coarse-to-fine framework synthesizes clustering and model training in large-scale infrastructure, which could also be applied to other scalable scenarios, *e.g.,* large-scale time series prediction or classification. (2) For anomaly detection methods with the CTF framework, data within a shorter time window (a few days instead of months) may be sufficient to train models for more machine entities, *e.g.,* CTF with OmniAnomaly uses five days' data to train models and achieves equally high F1-Score compared with eighteen days' data in OmniAnomaly only.

VII. Conclusion

We built CTF to perform anomaly detection for large-scale and high-dimensional data center infrastructures. CTF is a synthetic framework with four steps — pre-training, feature extraction, machine clustering, and model transfer. The synthetic framework takes advantage of low-dimensional $\mathbf{z_t}$ distribution clustering, model reuse, and fine-tuning in model transfer to boost the training efficiency. We carefully justify the design choices, such as the clustering algorithm and distance measure, to improve the accuracy to the best. We exercise CTF prototype on production data and demonstrate the scalability and effectiveness of its design. We also release a labeling tool for MTS and a labeled dataset to the research community.

ACKNOWLEDGMENT

This project is supported by National Natural Science Foundation of China Grant No. 61802225, the Zhongguancun Haihua Institute for Frontier Information Technology, National Natural Science Foundation of China Grant No. 61902200 and China Postdoctoral Science Foundation (2019M651015).

REFERENCES

[1] 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 ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2828–2837.

[2] 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 *WWW*, 2018.

[3] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 211–224.

[4] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 4–5.

[5] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[6] 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 *International Conference on Learning Representations*, 2018.

[7] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 387–395.

[8] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[9] 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 AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1409–1416.

[10] 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 ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 3009–3017.

[11] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[12] Z. Li, Y. Zhao, R. Liu, and D. Pei, "Robust and rapid clustering of kpis for large-scale anomaly detection," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[13] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1067–1075.

[14] E. W. Grafarend, *Linear and nonlinear models: fixed effects, random effects, and mixed models*. de Gruyter, 2006.

[15] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1939–1947.

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[17] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[19] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International Conference on Machine Learning*, 2015, pp. 97–105.

[20] F. Murtagh and P. Legendre, "Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion?" *Journal of classification*, vol. 31, no. 3, pp. 274–295, 2014.

[21] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[22] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*. Springer, 2018, pp. 270–279.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[25] X. Li, J. Lin, and L. Zhao, "Linear time complexity time series clustering with symbolic pattern forest," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 2930–2936.

[26] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3861–3870.

[27] 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." NIPS, 2018.

[28] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[29] X. Zhang, J. Kim, Q. Lin, K. Lim, S. O. Kanaujia, Y. Xu, K. Jamieson, A. Albarghouthi, S. Qin, M. J. Freedman *et al.*, "Cross-dataset time series anomaly detection for cloud systems," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 1063–1076.

[30] 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 transactions on Cybernetics*, vol. 46, no. 6, pp. 1363–1374, 2015.