Robust System Instance Clustering for Large-Scale Web Services

Shenglin Zhang[*†]
Nankai University;
HL-IT; TKL-OS
Tianjin, China

Dongwen Li
Nankai University
Tianjin, China

Zhenyu Zhong
Nankai University
Tianjin, China

Jun Zhu
Nankai University
Tianjin, China

Minghan Liang
Nankai University
Tianjin, China

Jiexi Luo
Nankai University
Tianjin, China

Yongqian Sun[†‡]
Nankai University;
TKL-OS
Tianjin, China

Ya Su
Kuaishou Technology
Beijing, China

Sibo Xia
Nankai University
Tianjin, China

Zhongyou Hu
Nankai University
Tianjin, China

Yuzhi Zhang[*†]
Nankai University;
HL-IT; TKL-OS
Tianjin, China

Dan Pei[§]
Tsinghua University;
BNRist
Beijing, China

Jiyan Sun
Chinese Academy of
Sciences
Beijing, China

Yinlong Liu
Chinese Academy of
Sciences
Beijing, China

ABSTRACT

System instance clustering is crucial for large-scale Web services because it can significantly reduce the training overhead of anomaly detection methods. However, the vast number of system instances with massive time points, redundant metrics, and noise bring significant challenges. We propose *OmniCluster* to accurately and efficiently cluster system instances for large-scale Web services. It combines a one-dimensional convolutional autoencoder (1D-CAE), which extracts the main features of system instances, with a simple, novel, yet effective three-step feature selection strategy. We evaluated *OmniCluster* using real-world data collected from a top-tier content service provider providing services for one billion+ monthly active users (MAU), proving that *OmniCluster* achieves high accuracy (NMI=0.9160) and reduces the training overhead of five anomaly detection models by 95.01% on average.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Networks** → **Network services**.

KEYWORDS

Multivariate time series, Clustering, 1D-CAE

[*]HL-IT: Haihe Laboratory of Information Technology Application Innovation
[†]TKL-OS: Tianjin Key Laboratory of Operating System
[‡]Yongqian Sun is the corresponding author (sunyongqian@nankai.edu.cn).
[§]BNRist: Beijing National Research Center for Information Science and Technology

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3511983

ACM Reference Format:
Shenglin Zhang, Dongwen Li, Zhenyu Zhong, Jun Zhu, Minghan Liang, Jiexi Luo, Yongqian Sun, Ya Su, Sibo Xia, Zhongyou Hu, Yuzhi Zhang, Dan Pei, Jiyan Sun, and Yinlong Liu. 2022. Robust System Instance Clustering for Large-Scale Web Services. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3485447.3511983

1 INTRODUCTION

Cloud-native platforms allow developers to quickly build new application architectures that are resilient, elastic, and agile, and thus an increasing number of Web services are applying microservice architecture [19]. Web service in the microservice system can have several to thousands of instances running on different containers, virtual machines, or physical machines [18, 28, 32, 45, 46]. Therefore, there are a vast number of system instances, *e.g.*, service instances, containers, virtual machines, physical machines, switches, routers, in a large-scale Web service. The reliability of these system instances is of vital importance to Web services because their anomalous behavior may degrade the availability of Web services, impact user experience and even lead to economic loss [32, 41, 44, 48].

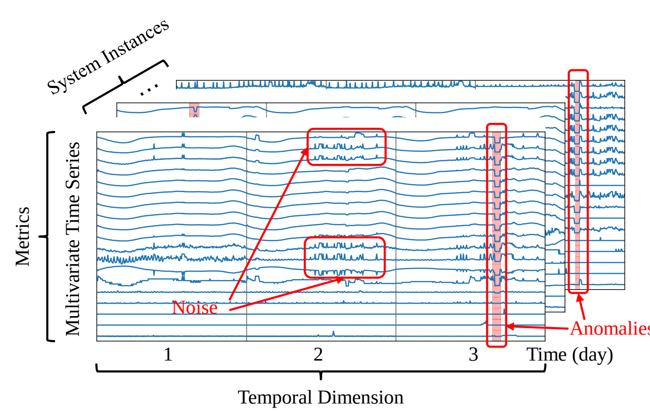


Figure 1: The MTS of system instances in large-scale Web services.

For proactively detecting the anomalous behaviors of system instances and timely mitigate system failures, operators configure diverse types of system-level metrics, *e.g.*, CPU utilization, memory utilization, disk I/O, network throughput, and user-perceived metrics, *e.g.*, average response time, error rate, page view count, and continuously collect their monitoring data at predefined time intervals. The monitoring metrics of a system instance thus form a multivariate time series (MTS), as shown in Figure 1. Recently, a collection of deep learning-based methods have been proposed for MTS anomaly detection because they can accurately learn the expressive representations of complex and massive MTS data [13, 40, 47, 49]. Typically, an MTS anomaly detection model learns the normal patterns of a system instance and determines that the instance becomes anomalous when its behavior deviates from the learned normal patterns. However, on the one hand, there are a huge number of system instances in large-scale Web services (*e.g.*, millions of system instances Alibaba [28] and ByteDance [41]), and training an MTS anomaly detection model for each system instance will consume a lot of computational resources (see Table 1). On the other hand, training one anomaly detection model for all system instances is not expressive enough for diverse system instances and will degrade the accuracy [42] (see Table 5). Therefore, it is pretty challenging to deploy these MTS anomaly detection methods in large-scale Web services. Although CTF [42] moves the first step to address the anomaly detection problem for high-dimensional time series, it only improves the performance of RNN+VAE-based model (*i.e.*, OmniAnomaly [40]), and is noise-sensitive (see Table 6).

After investigating thousands of real-world system instances, we observe that: 1) Because many system instances share similar patterns of normal metrics, most of the system instances, except for outliers, can be grouped into different clusters according to their metric patterns. 2) Although the system instances housing the same microservice have similar patterns, a system instance usually houses several microservices, and different system instances house different combinations of microservices. Therefore, it is impractical to group system instances according to the combinations of microservices deploying on them. Now, it is intuitive to automatically group system instances into different clusters, such that the system instances of each cluster have similar normal patterns of metrics. In this way, we can train an MTS anomaly detection model for each cluster instead of each system instance, significantly reducing the training overhead since the number of clusters is much smaller than system instances. For a system instance, its MTS represents the normal patterns of its metrics, and the problem is transformed into an MTS clustering problem.

Over the years, several MTS clustering methods have been proposed [17, 20, 23, 24, 36]. However, none of them can address the three challenges lying in large-scale Web services: 1) There are a vast number of system instances with massive time points containing noise and anomalies. 2) A system instance usually has redundant and non-periodic metrics, which can degrade the performance of MTS clustering. 3) A labeling tool is needed to efficiently cluster MTS manually for model evaluation and improvement (§ 2.2).

In this paper, we propose *OmniCluster* to accurately and efficiently cluster system instances for large-scale Web services. *OmniCluster* utilizes one-dimensional convolutional autoencoder (1D-CAE) to embed high-dimensional data into low-dimensional data, extracting the main features of MTS. Additionally, it applies a simple, novel, yet effective strategy to select periodic and representative features. *OmniCluster* is task-agnostic, and it can be applied for any type of MTS anomaly detection model.

The contributions of this paper are summarized as follows:

(1) We apply 1D-CAE to embed high-dimensional data into low dimensional data, which not only reduces clustering overhead but also eliminates the impact of noise and anomalies, addressing the first challenge. To the best of our knowledge, we are among the first to apply 1D-CAE for MTS clustering (§ 3.3).
(2) We propose a novel strategy to select periodic and representative features, which prevents some features from interfering with MTS clustering, addressing the second challenge (§ 3.4).
(3) We conducted extensive evaluation experiments using real-world data collected from ByteDance, a top-tier content service provider providing services for one billion+ monthly active users (MAU). *OmniCluster* achieves an NMI of 0.9160, significantly outperforming baseline methods (§ 4.4.2). It reduces the training time of five anomaly detection models by 95.01% on average without significantly degrading F_1-score (§ 5.3).
(4) We have published a labeling tool for MTS clustering and a labeled dataset for further studies (Appendix A).

2 BACKGROUND AND CHALLENGES

2.1 Background

MTS anomalies of system instances (*e.g.*, fluctuations or rapid changes that deviate from normal patterns) [29, 34] often indicate potential faults, such as hardware crash, service collapse, software bugs, *etc.* These faults usually negatively impact service availability and user experience. Therefore, it is necessary to proactively detect anomalies to mitigate possible failures timely.

The offline training time of an anomaly detection approach is the time between when the training begins and when the anomaly detection system becomes effective. A collection of MTS anomaly detection algorithms, *e.g.*, [6, 13, 26, 40, 49], require long training time to reach their optimal performance. In Table 1 we list the empirical offline training time of five state-of-the-art MTS anomaly detection approaches (see Appendix D.1) using the same dataset (see § 4.1.1) on a high-performance server. From the second column of Table 1, we can see that the average offline training time of these approaches for training one model ranges from about 8 seconds to about 8 minutes. With the number of system instances becoming one million, the estimated accumulated training time is at least 99.87 days. Due to their considerable training cost, it is quite difficult, if not infeasible, to deploy them for large-scale Web services.

2.2 Challenges

In this work, we aim to accurately cluster system instances according to their normal metric patterns, which can be transformed into an MTS clustering problem. It faces the following three challenges:

A vast number of system instances with massive time points containing noise and anomalies. The MTS of large-scale Web services have a considerable space because: First, (instance dimension) the scale of Web services has increased fast in the past few

Table 1: The offline training time of five deep-learning-based anomaly detection approaches on a high-performance server.

Method	1 Instance	1 M Instances
USAD [6]	8.63 s	99.87 day
OmniAnomaly [40]	2.99 min	5.70 year
SDFVAE [13]	4.08 min	7.70 year
InterFusion [26]	8.09 min	15.40 year
DAGMM [49]	17.81 s	206.16 day

years, and it can include millions of system instances now. Second, (temporal dimension) each system instance is monitored in a fine-grained frequency, *e.g.*, with a five-minute monitoring interval, there will be 2016 time points for a system instance each week. However, a clustering algorithm usually needs to compare the pairwise distance of all data, which will consume too many computational resources for such large-scale MTS data. Additionally, we should cluster MTS based on their *normal* patterns, whereas there are lots of noise and anomalies in the MTS, which can degrade the performance of MTS clustering. We apply 1D-CAE to extract the main features and embed high-dimensional data into low-dimensional spaces, which significantly improves the computational efficiency of *OmniCluster* and avoids the interference of noise and anomalies.

Some metrics may degrade the performance of MTS clustering. Operators usually configure tens to one hundred+ metrics for a system instance. Some metrics can be highly interdependent, *e.g.*, the CPU-related metrics typically manifest very similar patterns [33]. The abundant metrics (features) can degrade the performance of MTS clustering [39]. Moreover, operators usually believe that a non-periodic metric is uninformative for MTS clustering, and thus it should be removed before MTS clustering. We select periodic and representative metrics to eliminate the impact of redundant and non-periodic features through feature selection.

Lack of labeling tool. Although there is no need to obtain the labels of MTS clusters for training an unsupervised MTS clustering model, we still need labeled data to evaluate and improve the model's performance. It is challenging, if not impossible, to label such large-scale data (in terms of instance dimension and temporal dimension) without the help of user-friendly tools. Therefore, we implement a labeling tool for MTS clustering with user-friendly interfaces.

3 APPROACH

We denote the value of the s-th system's m-th metric at time t by x_{smt}. The MTS of the s-th system $\mathbf{x}_s$ is a $M \times T$ matrix, *i.e.*, $\mathbf{x}_s \in \mathbb{R}^{M \times T}$, where M is the number of metrics per system instance and T is the number of data points in each metric.

3.1 Overview

The overall architecture of *OmniCluster* is illustrated in Figure 2. *OmniCluster* consists of two major components: offline clustering and online classification. Offline clustering has four stages. The first stage is preprocessing. In this stage, we smooth MTS data and do normalization. After that, *OmniCluster* uses 1D-CAE to reduce the number of time points in each metric (temporal dimension). Their hidden representations $\mathbf{z}$ can be obtained. Then, feature selection is performed on $\mathbf{z}$ to get $\mathbf{z}''$, reducing the number of metrics in

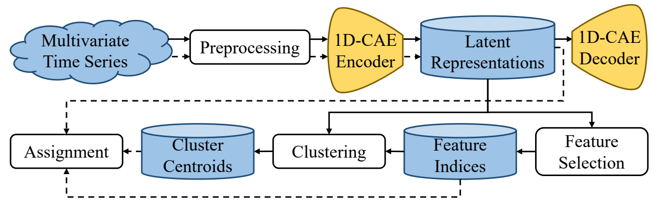


Figure 2: The overall process of *OmniCluster*. Solid lines denote offline clustering, dash lines denote online classification.

each MTS (metric dimension). High-dimensional data is converted into low-dimensional features through these two steps. Thus, *OmniCluster* can deal with a vast number of system instances with massive time points and multiple metrics. Finally, we use $\mathbf{z}''$ in hierarchical agglomerative clustering. In online classification, *OmniCluster* feeds preprocessed data into the encoder and extracts feature subsets according to saved feature indices. Then, we assign class labels to system instances and identify outliers.

3.2 Preprocessing

MTS usually have anomalies, noise, and missing values that can significantly affect their shapes. It is necessary to minimize the negative impact brought by them. Extreme values usually have a better chance of being anomalies. We remove the top 5% data deviating from the mean value to handle these extreme values [27]. There may also be some missing values due to errors in the data collection process. We use linear interpolation to fill the removed or missing values. As a result, extreme values and missing values are replaced by normal observations around them. After that, we smooth MTS curves by extracting their baselines. To deal with noise, we apply the moving average algorithm with a suitable sliding window. Each data point is replaced by the average value of w points around it, where w is the size of the sliding window. To deal with the difference of amplitudes, we adopt normalization in *OmniCluster*, scaling each data point to be in the range of [0, 1]:

$$\mathbf{x}'_{sm} = \frac{\mathbf{x}_{sm} - \min_{t \in T} x_{smt}}{\max_{t \in T} x_{smt} - \min_{t \in T} x_{smt}} \tag{1}$$

3.3 Reduction of the Temporal Dimension

To tackle the challenge of dimensionality [8], *OmniCluster* uses deep learning methods to reduce the temporal dimension and capture the non-linear relationship between the input. We employ 1D-CAE and use the reconstruction loss function for model training (Figure 3). More details can be seen in Appendix D.2. The convolutional encoder can be used for feature extraction and dimensionality reduction. It tries to learn the normal patterns of the input data and ignores noise and anomalies. 1D-CNN will extract the local features of MTS without making any assumptions about their distribution and use the extracted low-dimensional representations to reconstruct the input data. The encoder in *OmniCluster* is composed of M 1D-CNNs with independent parameters. Each metric in MTS will be input into different CNNs (composed of several 1D convolutional layers) in the encoder to obtain M corresponding features. The decoder structure is similar to that of the encoder, which is

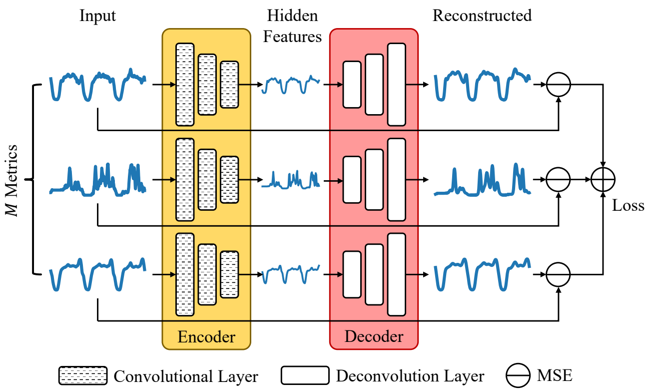


Figure 3: A 1D-CAE model with three convolutional layers on each side. The input MTS has three metrics.

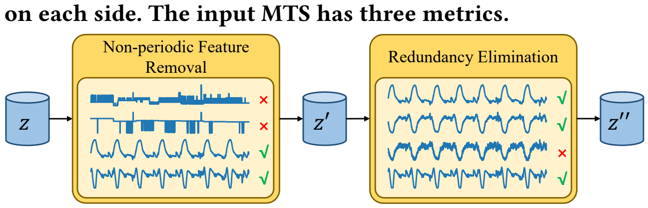


Figure 4: The process of feature selection.

composed of M different deconvolutional networks. The output of the decoder is the reconstruction $\hat{\mathbf{x}}$ of the original data.

We use different 1D-CNN on each metric instead of 2D-CNN on the whole MTS because the information loss of 2D-CNN will cause some metrics to be lost. Furthermore, each metric has a corresponding physical meaning, so we use different CNNs for different metrics to avoid interference between them. The empirical experiments in § 4.3 show that this network structure achieves good performance. In addition, *OmniCluster* uses strided CNN instead of spatial pooling layers and has better generalization capabilities [7]. The extracted feature set is the output of the trained encoder, denoted by $\mathbf{z} \in \mathbb{R}^{M \times T'}$, where T' is the number of data points in a single feature.

OmniCluster continuously updates the model by minimizing the loss between the input data $\mathbf{x}$ and the output data $\hat{\mathbf{x}}$. We use mean squared error as the loss function.

3.4 Feature Selection

In this paper, a robust and generic feature selection method for MTS is implemented to reduce the number of features in the metric dimension and improve the performance of clustering. Our approach relies on two key observations: 1) MTS are usually periodic. MTS without seasonal patterns are poorly informative and can harm the clustering results. 2) Keeping useful information as much as possible in the low-dimensional space can improve clustering performance. We perform feature selection on the low-dimensional feature $\mathbf{z}$ output by 1D-CAE in the previous stage. The periodicity and diversity of the data itself determine the number of selected features, without the need for expert intervention.

The feature selection process includes three steps (Figure 4). The first step is to remove non-periodic features, the second and the third step is to remove redundant features.

3.4.1 Step 1: Non-periodic Feature Removal. In this step, we use YIN [12] to extract periodic information from features. YIN is designed to extract the frequency of sound data. $YIN(\mathbf{z}_{sm}) > 0$ means $\mathbf{z}_{sm}$ is periodic, while $YIN(\mathbf{z}_{sm}) = 0$ means the feature does not have an obvious periodic pattern. Features that are non-periodic in most system instances will be removed (see Appendix C Algorithm 1 for more details). we remove non-periodic features using and get the preserved features $\mathbf{z}'$.

3.4.2 Step 2: Redundancy Matrix Construction. The second feature selection step builds a matrix of feature redundancy $\mathbf{R} \in [0,1]^{M' \times M'}$, such that $R_{ij} > 0$ if features i and j are pairwise redundant and $R_{ij} = 0$ otherwise, where M' is the number of remaining features after step 1. Given a feature matrix $\mathbf{z}' \in \mathbb{R}^{M' \times T'}$ output by step 1, the redundancy matrix is computed as Appendix C Algorithm 2, where NCC denotes the normalized cross-correlation function [27]. For each sample, *OmniCluster* computes NCC between all univariate sequences in $\mathbf{z}'_s$. If $NCC(\mathbf{z}'_{si}, \mathbf{z}'_{sj}) > \theta_s$ for the pair i, j, we can determine that features i and j are positively correlated on the s-th sample. The redundancy matrix $\mathbf{R}$ provides a unified picture of which features positively correlate with a sufficiently large share of input samples.

3.4.3 Step 3: Redundancy Elimination. The third step applies the feature selection/elimination rules to exploit the information in the redundancy matrix $\mathbf{R}$. *OmniCluster* defines a set of unassigned features $\mathcal{F}$, which initially contains the indices of all the M' features. The rules are applied iteratively to $\mathcal{F}$ following a priority order until all features are assigned to either the set of selected features $\mathcal{SF}$ or to the set of the deleted ones $\mathcal{DF}$. The details of the feature selection rules and their priority pattern are described by the following procedure:

RULE 1: If a row $\mathbf{R}_i$ is completely uncorrelated with the others in $\mathbf{R}$ (*i.e.*, $\mathbf{R}_i$ contains only zeros),
- (a) Add i to the selected subset: $\mathcal{SF} = \mathcal{SF} \cup \{i\}$;
- (b) Remove i from $\mathcal{F}$ and remove the corresponding entries in $\mathbf{R}$;

RULE 2: If a row $\mathbf{R}_i$ is correlated with all the others and there is at least one noncompletely correlated feature (*i.e.*, $\mathbf{R}$ does not contain only non-zero off-diagonal values),
- (a) Add i to the deleted subset: $\mathcal{DF} = \mathcal{DF} \cup \{i\}$;
- (b) Remove i from $\mathcal{F}$ and remove the corresponding entries in $\mathbf{R}$;

RULE 3: If all features in $\mathcal{F}$ are correlated with each other (*i.e.*, $\mathbf{R}$ contains only non-zero off-diagonal values),
- (a) Select the feature i that is minimally correlated with those currently in $\mathcal{SF}$;
- (b) Add i to the selected subset: $\mathcal{SF} = \mathcal{SF} \cup \{i\}$;
- (c) Remove i from $\mathcal{F}$;
- (d) Move the remaining features $\mathcal{F}$ to the deleted subset ($\mathcal{DF} = \mathcal{DF} \cup \mathcal{F}$) and terminate.

RULE 4: If neither 3.4.3 nor 3.4.3 apply,
- (a) Extract feature $i \in \mathcal{F}$ that is minimally correlated with the features still in $\mathcal{F}$;
- (b) Define $\mathcal{S}(i) \subset \mathcal{F}$ as the subset of features correlated with i and select $j \in \mathcal{S}(i)$ as the maximally correlated feature with those currently in $\mathcal{SF}$;

(c) Add i to $\mathcal{SF}$ and j to $\mathcal{DF}$;
(d) Remove i, j from $\mathcal{F}$ and remove the corresponding entries in $\mathbf{R}$.

The elimination/selection rules are tested sequentially, from **RULE 1** to **RULE 4**, and their test conditions are such that at least one of them fires at each iteration of the algorithm; hence, the cost of computing the third step is at most linear to the number of the original features.

Note that, in rule **RULE 3**(a), we determine the feature i that is minimally correlated with those in $\mathcal{SF}$ by

$$i = \arg\min_{i' \in \mathcal{F}} \sum\nolimits_{j \in \mathcal{SF}} R_{i'j} \tag{2}$$

where $\mathbf{R}$ is the complete redundancy matrix output by step 2. **RULE 4**(a) and **RULE 4**(b) uses a similar strategy to determine which feature i is extracted or which feature j has to be deleted. Finally, *OmniCluster* concatenates all the selected features in $\mathcal{SF}$ as $\mathbf{z}''$, the input of clustering or assignment.

3.5 Clustering and Assignment

In *OmniCluster*, hierarchical agglomerative clustering (HAC) with average linkage is adopted because of the following reasons: 1) Due to the diversity of MTS patterns, it is difficult to specify the number of clusters in advance. HAC is a "bottom-up" approach and can use the threshold of the distance between clusters as a hyperparameter τ_d instead of specifying the number of clusters. 2) HAC using average linkage lets each data in the cluster have an equal effect on the distance between clusters, making the distance measurement transitive. 3) HAC can determine whether the data is an outlier by its distance from other data.

OmniCluster uses Euclidean distance, which is competitive in time series classification or clustering [22]. Only when the distance is less than the threshold, two clusters will be grouped. The distance between two clusters $\mathcal{A}$, $\mathcal{B}$ is defined as

$$D_{cluster}(\mathcal{A}, \mathcal{B}) = \frac{1}{|\mathcal{A}| \cdot |\mathcal{B}|} \sum_{\mathbf{a} \in \mathcal{A}} \sum_{\mathbf{b} \in \mathcal{B}} D_{MTS}(\mathbf{a}, \mathbf{b}) \tag{3}$$

where

$$D_{MTS}(\mathbf{a}, \mathbf{b}) = \sum_{m=1}^{M''} \|\mathbf{a}_m - \mathbf{b}_m\|_2 \tag{4}$$

$|*|$ denotes the cardinality of a set, and M'' is the number of features selected by § 3.4.

After offline clustering is completed, *OmniCluster* saves the encoder part of the 1D-CAE and the feature subset indices $\mathcal{SF}$ obtained by feature selection. The centroid of each cluster can represent the general characteristics of the cluster. *OmniCluster* computes the centroid $\mathbf{c}$ of each cluster C by

$$\mathbf{c} = \arg\min_{\mathbf{a} \in C} \sum_{\mathbf{b} \in C} D_{MTS}(\mathbf{a}, \mathbf{b}) \tag{5}$$

The saved encoder and subset indices are used to extract useful features from the preprocessed input, and then the extracted features are used to calculate their distance to each cluster's centroid. The closest centroid is selected, and *OmniCluster* checks whether the distance exceeds τ_d. If the distance is smaller than τ_d, the newly coming data gets the same class label as the selected centroid; otherwise, the data is considered as an outlier and reported to operators.

4 EVALUATION

4.1 Experimental Setup

4.1.1 Dataset and Environment. There are many public datasets (*e.g.*, Robot Failure [11], LIBRAS [14], Pendigits [5]) for MTS clustering, but none of these has a large enough scale to thoroughly test the performance of *OmniCluster*. In this work, we conduct evaluation experiments on a system-related dataset collected from ByteDance, a top-tier global content service provider providing services for one billion+ monthly active users (MAU). The source code of *OmniCluster* and the dataset is available at [1, 2]. Specifically, the original data is 7-day-long MTS segments collected from 3175 system instances sampled at an interval of five minutes. The number of metrics of each instance is 19. All the data has been manually labeled by experienced operation engineers. There are a total of 29 classes and 28 outlier instances in the dataset. We have developed a dedicated GUI tool to help operators effectively label the classes of instances. It is open-source and publicly available [3]. A more detailed description of the labeling tool can be found in Appendix A. All the experiments are run on a server with two 16C32T Intel(R) Xeon(R) Gold 5218 CPU @ 2.30 GHz, one NVIDIA(R) Tesla(R) V100S, and 192 GB RAM.

4.1.2 Evaluation Metrics. Inspired by previous works [15, 27], we adopt two well-accepted metrics, namely normalized mutual information (NMI) and accuracy (ACC), to measure the performance of *OmniCluster*. NMI measures the mutual dependence between the clustering results and the ground truth, considering both homogeneity and completeness. Let S be the total number of system instances, $\mathcal{T}_p$ be the p-th class of the ground truth, and C_q be the q-th cluster generated by a clustering algorithm. NMI is defined as

$$NMI = -\frac{2 \times \sum_p \sum_q \left(|\mathcal{T}_p \cap C_q| \times \log\left(\frac{S \times |\mathcal{T}_p \cap C_q|}{|\mathcal{T}_p| \times |C_q|}\right)\right)}{\sum_p \left(|\mathcal{T}_p| \times \log\left(\frac{|\mathcal{T}_p|}{S}\right)\right) + \sum_q \left(|C_q| \times \log\left(\frac{|C_q|}{S}\right)\right)} \tag{6}$$

NMI ranges from 0 to 1, where 1 means the results perfectly match the ground truth and 0 indicates they are completely irrelevant. ACC is measured by

$$ACC(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{S} \sum_{s=1}^{S} \delta(y_s = \hat{y}_s) \tag{7}$$

where $\mathbf{y}$ is the actual class labels of the results and $\hat{\mathbf{y}}$ is the labels that match the ground truth best. δ denotes the Kronecker delta.

In addition, real-world datasets usually have outliers, where the instances cannot be grouped into any cluster. We use F_1-score (F_1 for short) to test *OmniCluster*'s ability to detect outliers, with True Positives (TP), False Positives (FP), and False Negatives (FN). The calculation is given by $F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$, where precision $= \frac{\text{TP}}{\text{TP+FP}}$ and recall $= \frac{\text{TP}}{\text{TP+FN}}$.

4.1.3 Hyperparameters. *OmniCluster* has several hyperparameters. Some are robust and insensitive to the algorithm, *e.g.*, θ_y should be less than 50%, θ_s can take a value from 0.89 to 0.97, and θ_p should be greater than 40% (see more details in Appendix B). Some parameters can be automatically determined, *e.g.*, τ_d is automatically calculated to be 7 (see it in Appendix B). Some other parameters are determined empirically or from widely accepted references. For example, the

Table 2: The overall performance of MTS clustering algorithms.

Method	NMI	ACC	F_1	#C	Avg. Time
OmniCluster	**0.9160**	**0.7990**	**0.9057**	19	**11.69 min**
TICC	0.4826	0.3798	–	40	104.17 h
Mc2PCA	0.2703	0.2306	–	10	22.03 min
FCFW	0.6236	0.4117	–	10	195.86 h
SPCA+AED	0.4084	0.2746	–	40	4.91 h

length of the sliding window w in the preprocessing stage is set to 12, both the encoder and the decoder of 1D-CAE have 3 convolutional layers, the stride of each convolutional layer is 2 with the kernel size of 7, and the number of channels in the encoder and decoder is (16, 32, 1) and (32, 16, 1), respectively.

4.2 Overall Performance

To demonstrate the effectiveness and efficiency of *OmniCluster*, we compare it with four advanced MTS clusterig methods: TICC [20], Mc2PCA [23], FCFW [24], SPCA+AED [17] (see § 6). All of the baseline methods need to preset the number of clusters, which is contrary to unsupervised learning. We set it for each baseline that can lead to the best performance through many experiments. Table 2 shows the NMI, ACC, F_1 of outlier detection, number of clusters (#C), and average time cost of all algorithms. *OmniCluster* outperforms all baselines in terms of effectiveness and efficiency.

OmniCluster's NMI and ACC are 0.9160 and 0.7990 respectively, while the best NMI and ACC of baselines is 0.6236 and 0.4117, which is far lower than *OmniCluster*. As for outlier detection, only *OmniCluster* can identify outliers in the data with a F_1 of 0.9057. Other baselines are unable to distinguish outliers. In addition, *OmniCluster* takes 11.69 minutes to cluster all the system instances. Only Mc2PCA can achieve similar time consumption (22.03 minutes), while the other three baselines (TICC, FCFW, and SPCA+AED) cost far more time than *OmniCluster*, which are not suitable for the scenario where there are massive MTS.

We try to analyze the reasons for this result next. TICC is suitable for short time series with consistent patterns and is ineffective for long MTS whose patterns change over time. Therefore, *OmniCluster* outperforms it in our scenario where every MTS is long. In addition, TICC constructs a large Toeplitz Matrix for each cluster and uses the ADMM algorithm to update the clustering result iteratively, both of which are computationally inefficient. Mc2PCA only optimizes the projection of data points for each cluster but does not pay attention to the difference between the projections. It may be lead to a large cluster number, and some clusters have a very close distance. In FCFW, cluster centers are artificially selected and will not be updated later, which may affect its performance. Moreover, FCFW calculates the SBD distance independently for each metric and the DTW distance for each pair of MTS, both of which are very time-consuming. SPCA+AED's result heavily relies on the correctness of randomly initialized cluster centers. In addition, another reason for the poor performance of competing methods is that they mainly focus on ideal smooth data and do not consider the anomalies and noise in the real-world MTS. *OmniCluster* handles anomalies and noise by preprocessing and feature extraction, making it robust against non-smooth MTS.

Table 3: The performance of *OmniCluster* and its variants.

Method	NMI	ACC	F_1	#C	Avg. Time
OmniCluster	**0.9160**	0.7990	0.9057	19	11.69 min
C1	0.8511	0.6406	0.5243	46	**6.65 min**
C2	0.9102	**0.8009**	0.9057	23	135.60 min
C3	0.7602	0.4387	0.3022	117	9.09 min
C4	0.8724	0.6548	**0.9455**	35	11.48 min

4.3 Contributions of Components

To show the effects of three key techniques in *OmniCluster*: 1) 1D–CAE; 2) non-periodic feature removal; 3) redundancy elimination, we reconfigure *OmniCluster* to create four variants **C1-C4**, described as follows: **C1:** The 1D-CAE is replaced by 2D-CAE, denoted by "w/ 2D-CAE". **C2:** The CAE part is removed, denoted by "w/o CAE". **C3:** The non-periodic feature removal is omited, denoted by "w/o Non-periodic Feature Removal". **C4:** The redundancy elimination step is omitted, denoted by "w/o Redundancy Elimination". Table 3 shows the performance of *OmniCluster* and its variants.

Effect of 1D-CAE. "w/o CAE" can achieve performance similar to *OmniCluster*, even a little better ACC, but take a lot more time to consume (more than ten times). This indicates 1D-CAE or 2D-CAE can effectively reduce the dimensions of the original metrics, then improve the efficiency clustering. When using 2D-CAE instead of 1D-CAE, the average clustering time is further reduced. However, 2D-CAE will make metrics in an MTS interfere with each other, losing the original shape information, which leads to a relatively worse clustering result: more worse NMI, ACC, and F_1, and a more inaccurate number of clusters.

Effect of non-periodic feature removal. With "w/o Non-periodic Feature Removal", the dataset is grouped into 117 clusters, four times more than the ground truth. Both the NMI, ACC, and F_1 are relatively poor. The reason is that non-periodic features are irregular, having no contribution to clustering. On the contrary, they increase the noise of clustering. Generally, between two instances, the distance of non-periodic features can be considerable even though their periodic features are relatively similar. Therefore, it is indispensable to remove these non-periodic features.

Effect of redundancy elimination. "w/o Redundancy Elimination" has close time consumption and F_1, but lower NMI and ACC than *OmniCluster*. By using redundancy elimination, the time for calculating distance matrices is reduced, but it takes some more time to do redundancy elimination, so its time cost is close to "w/o Redundancy Elimination". If the instances contain more similar/redundancy features, their weight will be too large when calculating the distance and overshadow the role of different features, resulting in poor clustering results.

4.4 Validation of Design Choices

4.4.1 Choice of the Distance Measure. In this experiment, we replace our distance measurement method D_{MTS} with other popular ones: Manhattan distance [35], shape-based distance (SBD) [38], mean squared error (MSE) [4], normalized Euclidean distance (NED) [9], Wasserstein distance [42], Jensen-Shannon divergence (JSD) [21] and Pearson correlation coefficient (PCC) [4].

From Figure 5 we can see that *OmniCluster* with D_{MTS} outperforms the other options. Manhattan distance, MSE, and NED are not processed separately for different features. They just calculate

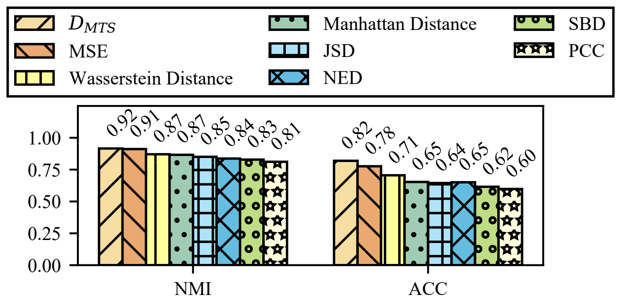


Figure 5: The NMI and ACC for different distance measurement methods.

Table 4: The performance of different clustering algorithms.

Method	NMI	ACC	F_1	#C	Avg. Time
HAC	**0.9160**	**0.7990**	0.9057	19	11.69 min
k-means	0.8798	0.6998	0.1333	28	**11.68 min**
DBSCAN	0.8713	0.6900	**0.9455**	26	11.69 min

the point-wise distance of all the features. SBD considers the data has phase shifts. However, our algorithm is based on the shape of features, and the phase shifts between features are also a type of shape change in our algorithm. As for Wasserstein distance and JSD, they focus on the distribution of data and are not fit for shape-based MTS clustering tasks. PCC is related to the length of its input. When the length of features is big, the absolute value of PCC will be scaled down, which can not accurately reflect the distance between features. D_{MTS} used by *OmniCluster* calculates the distance of different features separately and does not consider the influence of any additional factors, which is more suitable for MTS clustering.

4.4.2 Choice of the Clustering Algorithm. We replace HAC with DBSCAN [27] or k-means [43] to cluster on feature $\mathbf{z}''$. Table 4 shows the performance of these variants. *OmniCluster* with HAC achieves the best NMI and ACC. DBSCAN and k-means are more sensitive to extreme values. Many outliers are not successfully detected by k-means and are all grouped into normal clusters, which will greatly affect the performance of clustering. DBSCAN has a slightly better performance in outlier detection, but its NMI and ACC are lower than *OmniCluster* with HAC. In *OmniCluster*, HAC provides more stable results and is not affected by random initial values. This experiment proves that it has the best performance, so *OmniCluster* chooses HAC as the clustering algorithm.

5 ANOMALY DETECTION

To prove *OmniCluster* is universal to anomaly detection algorithms, we select five detection algorithms in this section: OmniAnomaly [40], USAD [6], SDFVAE [13], InterFusion [26], and DAGMM [49]. The architectures of these approaches are various.

We also compare "*OmniCluster* + OmniAnomaly" with CTF [42], a framework designed for OmniAnomaly to improve training efficiency, to test the performance of *OmniCluster* with some anomaly detection approaches against specifically designed frameworks.

Table 5: The performance of different anomaly detection setups.

Method	E1		E2		E4	
	F_1^*	Time (s)	F_1^*	Time (s)	F_1^*	Time (s)
OmniAnomaly	0.842	56773.77	0.833	219.36	**0.845**	2748.88
USAD	**0.926**	2726.80	0.841	8.99	0.923	133.88
SDFVAE	**0.893**	76740.62	0.831	242.86	0.886	3511.04
InterFusion	**0.836**	153378.84	0.680	295.35	0.827	9370.28
DAGMM	0.872	5628.57	0.826	18.50	**0.873**	254.89

5.1 Experimental Setup

We conduct four groups of experiments:

E1: Sharing No Model. Each system instance will have an anomaly detection model dedicatedly trained for it.

E2: Sharing One Model. In this setup, one model is used for all system instances. We randomly selected one system instance and use it to train a model.

E3: CTF. CTF cannot be used together with other anomaly detection algorithms, so we only compare "*OmniCluster* + OmniAnomaly" with CTF.

E4: *OmniCluster*. We first use *OmniCluster* to cluster system instances. Then we use the centroid of each cluster to train the anomaly detection model for that cluster. The instances in the same cluster share similar patterns so they can share the same model.

5.2 Dataset and Evaluation Metrics

We randomly selected 10% data (316 instances) from the dataset in § 4.1.1 for anomaly detection because of the limitations of manual labeling cost and training time. *OmniCluster* clusters these data into 19 classes. We divide each labeled 7-day-long data into two parts, the first 1440 points (5 days) as the training input and the other 576 points (2 days) as the test data. All selected data are labeled by experienced engineers according to actual system failures using the tool provided by CTF. The point-wise anomaly rate, *i.e.*, $\frac{\#\ anomaly\ data\ points}{\#\ total\ data\ points}$, is 9.04%.

We use F_1 to evaluate the performance of anomaly detection. In this paper, the final F_1 is obtained by micro-averaging, *i.e.*, the detection results of all system instances are aggregated to compute the precision and recall. By enumerating all possible thresholds, we calculate the best F_1 of each model, denoted by F_1^*, as the algorithm's theoretical best possible performance on our dataset. We also take the time required for model training into account to evaluate the training efficiency.

5.3 Results and Analysis

The experimental results of E1, E2, and E4 are displayed in Table 5.

Comparisons with E1. The training time when combining *OmniCluster* with anomaly detection algorithms (E4) is only about 5% of E1. The total model training time of E1 will become exceedingly large when dealing with a larger number of MTS. With *OmniCluster*, we just need to train one model for each category. Even if the scale of system instances expands further, the number of clusters can still be a small one, so the training time will not largely increase. Figure 6 shows that the training time of E4 will not significantly increase when the number of system instances in the dataset becomes

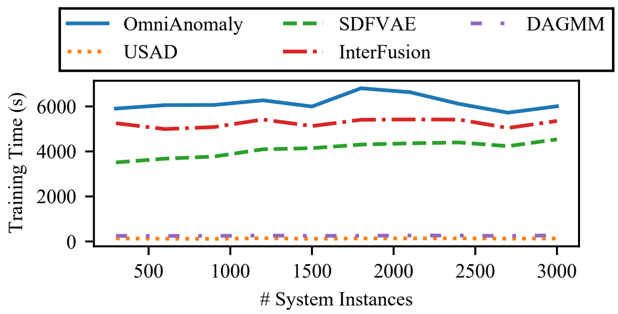


Figure 6: The offline training time of different anomaly detection methods with different numbers of system instances in the dataset.

larger. For larger datasets where each cluster contains more data, the advantage of reducing time becomes even more significant.

Among five anomaly detection approaches, there is no apparent difference in terms of F_1^* between E1 and E4. In E1, dedicated models for each system instance can perform very well, which is in line with our expectations. In E4, although only one model is trained for each cluster, MTS in the same cluster are very similar and the centroid that is used to train the model is representative.

Compared with training one specific model for each instance, training anomaly detection models with E4 could largely reduce training time by about 95.01% without impacting the performance.

Comparisons with E2. Training only one model for the whole dataset costs the least time of all four setups. Since E4 needs to train different models for 19 clusters, the training time of each model is about 18 times of E2. However, the F_1^* decreases by an average of 7.93%. This is because MTS can have various patterns and the selected MTS can not represent all the patterns.

In short, although E2 greatly reduces the training time, it sacrifices more performance, which is unacceptable in the production environment. Compared with E2, E4 can achieve satisfactory F_1^* within an acceptable training time.

Comparisons with CTF. The running time of CTF (E3) is 5601.48 s, about two times of E4. This is because the model structure of CTF is more complex and the amount of data it requires in fine-tuning is large. CTF's F_1^* is 0.8323, slightly lower than OmniAnomaly combined with *OmniCluster*. It shows that the performance of *OmniCluster* with some anomaly detection approaches is also comparable to frameworks that are specifically designed for a certain anomaly detection method.

Task Agnostic Clustering for Different Anomaly Detection Algorithms. It can be seen that the F_1^* of the combinations of *OmniCluster* and all algorithms are above 0.8. No matter which anomaly detection method is used together with *OmniCluster*, *OmniCluster* will not have a great impact on the effectiveness of the algorithm. Therefore, taking the comparisons with CTF into account, we believe that *OmniCluster* is task-agnostic.

6 RELATED WORK

Traditional clustering methods directly apply clustering algorithms, *e.g.*, k-means, DBSCAN, HAC, to the original data. Apart from naive clustering methods, there are algorithms proposed for the clustering of univariate time series, *e.g.*, SPF [25], ROCKA [27]. However, these methods can not be applied to MTS directly, which are either time-consuming or will lose important information when compressing the data.

There have been many studies on clustering MTS data. Copulas [36] compared intra-dependence between two MTS and the inter-dependence between two metrics in the same instance. However, non-parametric estimations of density suffer from the explosion of dimensionality and are costly to compute. Mc2PCA [23] is based on CPCA. It constructed common projection axes as the prototype of each cluster. The reconstruction error of each MTS projected on the corresponding common projection axes was used to reassign the member of the cluster. However, this algorithm only considered the similarity within the cluster instead of the similarity between clusters. SPCA+AED [17] consists of the PCA similarity factor (SPCA) and the average-based Euclidean distance (AED). It used fuzzy clustering. They stated that neither SPCA nor AED alone can effectively separate distinguishable instances. FCFW [24] is also based on two distance measurement methods —DTW and SBD. It utilized fuzzy c-means to calculate the fuzzy membership matrices and generate clustering results. The time complexity of DTW is $O\left(N^2\right)$, which is unacceptable for large-scale datasets. Toeplitz Inverse Covariance-Based Clustering (TICC) [20] focuses on subsequences in MTS. They proposed a model-based clustering method, with every cluster in the TICC algorithm defined by a correlation network characterizing the interdependencies between different observations in a typical subsequence of that cluster. Based on this graphical representation, TICC simultaneously segments and clusters MTS data. This method is also very time- and space-consuming.

7 CONCLUSION

This paper proposes *OmniCluster*, an efficient and robust algorithm for clustering high-dimensional MTS with noise, anomalies, and redundant features. The 1D-CAE of *OmniCluster* performs dimensionality reduction on the temporal dimension to improve efficiency and avoid the interference of noise and anomalies. Additionally, the novel three-step feature selection strategy prevents redundant and non-periodic features from degrading *OmniCluster*'s performance. Extensive experiments using large-scale real-world data demonstrate that *OmniCluster* greatly reduces the training overhead of anomaly detection methods. We have learned several lessons from designing *OmniCluster* (see Appendix E).

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. The work was supported by the National Key R&D Program of China (Grant 2019YFB1802504), National Natural Science Foundation of China (Grant No. 61902200 and 62072264), The China Postdoctoral Science Foundation (2019M651015), and Beijing National Research Center for Information Science and Technology (BNRist).

REFERENCES

[1] 2021. https://github.com/omni-cluster/OmniCluster-code/
[2] 2021. https://github.com/omni-cluster/OmniCluster-dataset/
[3] 2021. https://github.com/omni-cluster/OmniClutser-labeling-tool/
[4] Saeed Reza Aghabozorgi, Ali Seyed Shirkhorshidi, and Ying Wah Teh. 2015. Time-series clustering - A decade review. *Inf. Syst.* 53 (2015), 16–38. https://doi.org/10.1016/j.is.2015.04.007
[5] Fevzi Alimoglu and Ethem Alpaydin. 1997. Combining Multiple Representations and Classifiers for Pen-based Handwritten Digit Recognitio. In *4th International Conference Document Analysis and Recognition (ICDAR '97), 2-Volume Set, August 18-20, 1997, Ulm, Germany, Proceedings*. IEEE Computer Society, 637–640. https://doi.org/10.1109/ICDAR.1997.620583
[6] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. 2020. USAD: UnSupervised Anomaly Detection on Multivariate Time Series. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 3395–3404. https://doi.org/10.1145/3394486.3403392
[7] Riadh Ayachi, Mouna Afif, Yahia Said, and Mohamed Atri. 2020. Strided Convolution Instead of Max Pooling for Memory Efficiency of Convolutional Neural Networks. In *Proceedings of the 8th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT'18), Vol.1*. Springer International Publishing, 234–243. https://doi.org/10.1007/978-3-030-21005-2_23
[8] Richard Bellman. 2015. *Adaptive Control Processes - A Guided Tour (Reprint from 1961)*. Princeton Legacy Library, Vol. 2045. Princeton University Press. https://doi.org/10.1515/9781400874668
[9] Antonin Bernardin, Ludovic Hoyet, Antonio Mucherino, Douglas Soares Gonçalves, and Franck Multon. 2017. Normalized Euclidean distance matrices for human motion retargeting. In *Proceedings of the Tenth International Conference on Motion in Games, MIG 2017, Barcelona, Spain, November 08 - 10, 2017*, Nuria Pelechano, Stephen N. Spencer, Carol O'Sullivan, and Julien Pettré (Eds.). ACM, 15:1–15:6. https://doi.org/10.1145/3136457.3136466
[10] Purnima Bholowalia and Arvind Kumar. 2014. EBK-Means: A Clustering Technique based on Elbow Method and K-Means in WSN. *International Journal of Computer Applications* 105, 9 (November 2014), 17–24.
[11] Luis M. Camarinha-Matos, Luís Seabra Lopes, and José Barata. 1996. Integration and learning in supervision of flexible assembly systems. *IEEE Trans. Robotics Autom.* 12, 2 (1996), 202–219. https://doi.org/10.1109/70.488941
[12] De Cheveigné, Alain, Kawahara, and Hideki. 2002. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America* 111, 4 (2002), 1917–1930. https://doi.org/10.1121/1.1458024
[13] Liang Dai, Tao Lin, Chang Liu, Bo Jiang, Yanwei Liu, Zhen Xu, and Zhi-Li Zhang. 2021. SDFVAE: Static and Dynamic Factorized VAE for Anomaly Detection of Multivariate CDN KPIs. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 3076–3086. https://doi.org/10.1145/3442381.3450013
[14] Daniel B. Dias, Renata C. B. Madeo, Thiago Rocha, Helton Hideraldo Bíscaro, and Sarajane Marques Peres. 2009. Hand movement recognition for Brazilian Sign Language: A study using distance-based neural networks. In *International Joint Conference on Neural Networks, IJCNN 2009, Atlanta, Georgia, USA, 14-19 June 2009*. IEEE Computer Society, 697–704. https://doi.org/10.1109/IJCNN.2009.5178917
[15] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. 2017. Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 5747–5756. https://doi.org/10.1109/ICCV.2017.612
[16] Bo Du, Wei Xiong, Jia Wu, Lefei Zhang, Liangpei Zhang, and Dacheng Tao. 2017. Stacked Convolutional Denoising Auto-Encoders for Feature Representation. *IEEE Trans. Cybern.* 47, 4 (2017), 1017–1027. https://doi.org/10.1109/TCYB.2016.2536638
[17] Cristiano Hora Fontes and Hector Budman. 2017. A hybrid clustering approach for multivariate time series – A case study applied to failure analysis in a gas turbine. *ISA Transactions* 71 (Nov 2017), 513–529. https://doi.org/10.1016/j.isatra.2017.09.004
[18] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 3–18. https://doi.org/10.1145/3297858.3304013
[19] Gartner. 2021. *Top Strategic Technology Trends for 2022*. https://www.gartner.com/en/information-technology/insights/top-technology-trends
[20] David Hallac, Sagar Vare, Stephen P. Boyd, and Jure Leskovec. 2018. Toeplitz Inverse Covariance-based Clustering of Multivariate Time Series Data. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, Jérôme Lang (Ed.). ijcai.org, 5254–5258. https://doi.org/10.24963/ijcai.2018/732
[21] Yuto Kingetsu and Yukihiro Hamasuna. 2021. Jensen-Shannon Divergence-Based k-Medoids Clustering. *J. Adv. Comput. Intell. Intell. Informatics* 25, 2 (2021), 226–233. https://doi.org/10.20965/jaciii.2021.p0226
[22] Thomas Andrew Lampert, Baptiste Lafabregue, Thi-Bich-Hanh Dao, Nicolas Serrette, Christel Vrain, and Pierre Gançarski. 2019. Constrained Distance-Based Clustering for Satellite Image Time-Series. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.* 12, 11 (2019), 4606–4621. https://doi.org/10.1109/JSTARS.2019.2950406
[23] Hailin Li. 2019. Multivariate time series clustering based on common principal component analysis. *Neurocomputing* 349 (2019), 239–247. https://doi.org/10.1016/j.neucom.2019.03.060
[24] Hailin Li and Miao Wei. 2020. Fuzzy clustering based on feature weights for multivariate time series. *Knowl. Based Syst.* 197 (2020), 105907. https://doi.org/10.1016/j.knosys.2020.105907
[25] Xiaosheng Li, Jessica Lin, and Liang Zhao. 2019. Linear Time Complexity Time Series Clustering with Symbolic Pattern Forest. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 2930–2936. https://doi.org/10.24963/ijcai.2019/406
[26] Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. 2021. Multivariate Time Series Anomaly Detection and Interpretation using Hierarchical Inter-Metric and Temporal Embedding. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 3220–3230. https://doi.org/10.1145/3447548.3467075
[27] Zhihan Li, Youjian Zhao, Rong Liu, and Dan Pei. 2018. Robust and Rapid Clustering of KPIs for Large-Scale Anomaly Detection. In *26th IEEE/ACM International Symposium on Quality of Service, IWQoS 2018, Banff, AB, Canada, June 4-6, 2018*. IEEE, 1–10. https://doi.org/10.1109/IWQoS.2018.8624168
[28] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 338–347. https://doi.org/10.1109/ICSE-SEIP52600.2021.00043
[29] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: Towards Practical and Automatic Anomaly Detection Through Machine Learning. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring (Eds.). ACM, 211–224. https://doi.org/10.1145/2815675.2815679
[30] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. 2017. A survey of deep neural network architectures and their applications. *Neurocomputing* 234 (2017), 11–26. https://doi.org/10.1016/j.neucom.2016.12.038
[31] Ezequiel López-Rubio, Esteban J. Palomo, and Francisco Ortega-Zamorano. 2018. Unsupervised learning by cluster quality optimization. *Inf. Sci.* 436-437 (2018), 31–55. https://doi.org/10.1016/j.ins.2018.01.007
[32] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. 2020. AutoMAP: Diagnose Your Microservice-based Web Applications Automatically. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 246–258. https://doi.org/10.1145/3366423.3380111
[33] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, Feifei Li, Changcheng Chen, and Dan Pei. 2020. Diagnosing Root Causes of Intermittent Slow Queries in Large-Scale Cloud Databases. *Proc. VLDB Endow.* 13, 8 (2020), 1176–1189. https://doi.org/10.14778/3389133.3389136
[34] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. 2021. Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 413–426.
[35] Mouâd Mansouri and Leghris Cherkaoui. 2019. New Manhattan distance-based fuzzy MADM method for the network selection. *IET Commun.* 13, 13 (2019), 1980–1987. https://doi.org/10.1049/iet-com.2018.5454
[36] Gautier Marti, Frank Nielsen, and Philippe Donnat. 2016. Optimal copula transport for clustering multivariate time series. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*. IEEE, 2379–2383. https://doi.org/10.1109/ICASSP.2016.7472103
[37] Hrushikesh Mhaskar, Qianli Liao, and Tomaso A. Poggio. 2017. When and Why Are Deep Networks Better Than Shallow Ones?. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 2343–2349.

[38] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1855–1870. https://doi.org/10.1145/2723372.2737793

[39] Ani Dijah Rahajoe, Edi Winarko, and Suryo Guritno. 2017. A Hybrid Method for Multivariate Time Series Feature Selection. *International Journal of Computer Science and Network Security (IJCSNS)* 17, 3 (2017), 103.

[40] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 2828–2837. https://doi.org/10.1145/3292500.3330672

[41] Ya Su, Youjian Zhao, Ming Sun, Shenglin Zhang, Xidao Wen, Yongsu Zhang, Xian Liu, Xiaozhou Liu, Junliang Tang, Wenfei Wu, and Dan Pei. 2021. Detecting Outlier Machine Instances through Gaussian Mixture Variational Autoencoder with One Dimensional CNN. *IEEE Trans. Comput.*, 1–1. https://doi.org/10.1109/TC.2021.3065073

[42] Ming Sun, Ya Su, Shenglin Zhang, Yuanpu Cao, Yuqing Liu, Dan Pei, Wenfei Wu, Yongsu Zhang, Xiaozhou Liu, and Junliang Tang. 2021. CTF: Anomaly Detection in High-Dimensional Time Series with Coarse-to-Fine Model Transfer. In *40th IEEE Conference on Computer Communications, INFOCOM 2021, Vancouver, BC, Canada, May 10-13, 2021*. IEEE, 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488755

[43] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 3861–3870.

[44] Fanghua Ye, Zhiwei Lin, Chuan Chen, Zibin Zheng, and Hong Huang. 2021. Outlier-Resilient Web Service QoS Prediction. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 3099–3110. https://doi.org/10.1145/3442381.3449938

[45] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 3087–3098. https://doi.org/10.1145/3442381.3449905

[46] Guangba Yu, Pengfei Chen, and Zibin Zheng. 2019. Microscaler: Automatic Scaling for Microservices with an Online Learning Approach. In *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019*, Elisa Bertino, Carl K. Chang, Peter Chen, Ernesto Damiani, Michael Goul, and Katsunori Oyama (Eds.). IEEE, 68–75. https://doi.org/10.1109/ICWS.2019.00023

[47] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. 2019. A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 1409–1416. https://doi.org/10.1609/aaai.v33i01.33011409

[48] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, Youjiang Wu, Ken Hsieh, Kaixin Sui, Xin Meng, Yaohai Xu, Wenchi Zhang, Furao Shen, and Dongmei Zhang. 2019. Cross-dataset Time Series Anomaly Detection for Cloud Systems. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, Dahlia Malkhi and Dan Tsafrir (Eds.). USENIX Association, 1063–1076.

[49] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Dae-ki Cho, and Haifeng Chen. 2018. Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 202–219.

A LABELING TOOL

Figure 7 shows the user interface of the tool. On the left side is the widget displaying MTS, while buttons to assign class labels and other actions are sitting on the right side.

It loads MTS and displays them as a group of line graphs in the left panel. Operators can use arrow buttons and number buttons on the right side to navigate through the data. The class label of the current data is presented under the navigation area. When new data comes, operators can click the checkbox before each class label to compare new data with the manually picked centroid of each class and determine which class it should be. However, with manually labeling, it is impossible to set a fixed number of classes and pick the centroid of each class at the beginning. We design our tool to be able to create new classes and update centroids easily. If the current data is not similar to any class centroids, operators can create a new class with the current data by clicking the "new" button. Once operators found a better centroid for one class, they can replace the old one.

One possible issue of labeling is that mistakes can happen, especially when operators work with ambiguous data patterns. This kind of data can be classified into incorrect classes. Our tool also provides a module for visualizing the class labeling result. In this module, MTS belonging to the same class will be displayed in the left panel at the same time, so it is easy for operators to check if any data should not be tagged as this class and reassign the label.

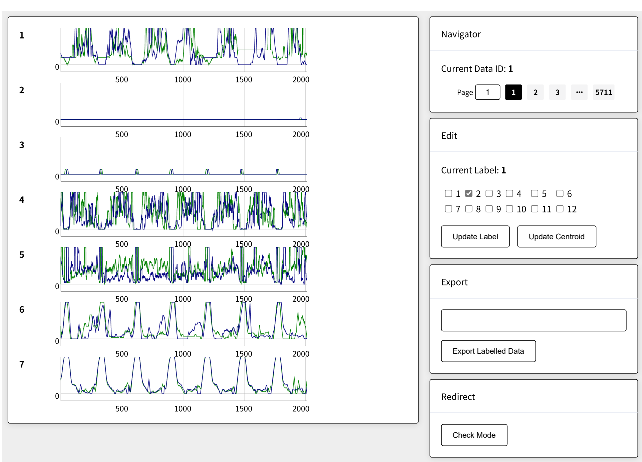


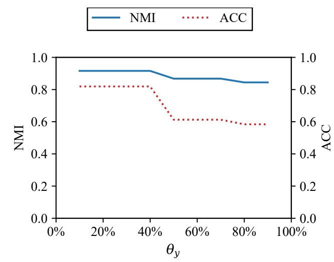
Figure 7: The GUI of the labeling tool.

B EFFECT OF HYPERPARAMETERS

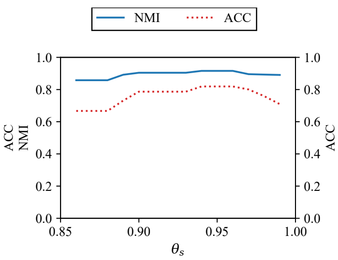
We mainly conducted experiments on four hyperparameters that have a significant impact on the results: θ_y, θ_s, θ_p, and τ_d.

The larger the θ_y, the stronger the requirement for the periodicity of features. Empirically, we set this value to 30% in *OmniCluster*. A too-large θ_y may cause unnecessary feature loss. This will adversely affect clustering. Figure 8a shows when θ_y is greater than 50%, NMI and ACC have an obvious drop. There are too many features removed, some of which are helpful for clustering. The value of θ_y will not affect NMI and ACC within a certain range.

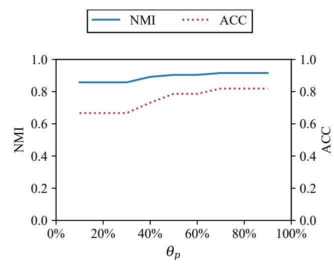
The larger the θ_s, the stricter the decision of whether two features are considered positively correlated with each other. θ_s is set



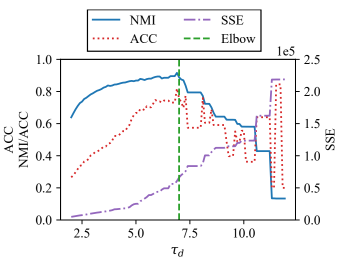
(a) The performance of different θ_y.



(b) The performance of different θ_s.



(c) The performance of different θ_p.



(d) Use SSE to select the optimal τ_d.

Figure 8: The effect of varying different hyperparameters.

to 0.95 in *OmniCluster*. A too-large θ_s may cause two features that are very similar in shape are not seen as correlated. A too-small θ_s will consider too many informative features redundant, which leads to feature loss. Figure 8b shows the NMI and ACC with different θ_s. When θ_s is greater than 0.97 or smaller than 0.89, the ACC drops significantly. When θ_s is between 0.9 and 0.97, the NMI and ACC stay at a high level. Therefore, setting θ_s in this range will not affect the results too much.

The larger the θ_p, the stricter the decision of whether two features can be represented by each other in the dataset. It is set to be 80% in *OmniCluster*. A large θ_p means only those features that are correlated in most samples are considered redundant. A too-small θ_p will think most of the features redundant, despite that they may only be correlated in less than half of the samples. Figure 8c shows when θ_p is smaller than 40%, there are no sufficient remaining features to distinguish different categories.

From the experiments above, we can determine that we have a large room to choose a good θ_y, θ_s, or θ_p. We claim *OmniCluster* is robust because it is insensitive to hyperparameters. When working with other datasets, the value of θ_y, θ_s, and θ_p used in our experiments may also fit.

τ_d is a very important and sensitive parameter in *OmniCluster*, which will have a great influence on the clustering results. However, supervised metrics like NMI and ACC cannot help to choose τ_d. *OmniCluster* uses the sum of squared error (SSE) to select an appropriate τ_d. SSE together with the elbow method is usually used for the selection of hyperparameters in the clustering algorithm [10, 31]. Figure 8d shows the relationship between τ_d and NMI, ACC, and SSE. The larger τ_d is, the fewer clusters *OmniCluster* will produce. So we enumerate τ_d from big to small to find the appropriate elbow of SSE. The automatically selected τ_d value is 7, while the optimal τ_d is 6.9. The NMI and ACC at $\tau_d = 7$ are very close to those at $\tau_d = 6.9$.

Table 6: The Overview of Anomaly Detection Algorithms. W: With inter-metric correlation. E: Efficient. P: Good interpretability. N: Noise-resilient.

Method	Structure	W	E	P	N
OmniAnomaly	RNN+VAE	✓	×	✓	×
USAD	GAN+AE	×	✓	×	✓
SDFVAE	CNN+LSTM+VAE	×	×	×	✓
InterFusion	CNN+RNN+VAE	✓	×	✓	×
DAGMM	AE+GMM	×	✓	×	×

C ALGORITHMS

Algorithm 1 Non-periodic Feature Removal

Require: $\mathbf{z}$ composed of S data instances, each of which has M features with 7-day-long period. Threshold θ_y.
$\mathbf{P} \leftarrow \mathbf{0}$
for $s \in \{1, \ldots, S\}$ and $m \in \{1, \ldots, M\}$ **do**
  if $YIN(\mathbf{z}_{sm}) > 0$ **then**
    $P_m \leftarrow P_m + 1$
  end if
end for
$\mathbf{P} \leftarrow \mathbf{P}/S$
$\mathcal{S} = \{m | P_m \geq \theta_y, 1 \leq m \leq M\}$
$\mathbf{z}' \leftarrow$ Select all the features in $\mathcal{S}$ from $\mathbf{z}$
return $\mathbf{z}'$

Algorithm 2 Redundancy Matrix Construction

Require: $\mathbf{z}'$ composed of S data instances, each of which has M' features with 7-day-long period. Thresholds θ_s, θ_p.
$\mathbf{R} \leftarrow \mathbf{0}$
for $s \in \{1, \ldots, S\}$ and $i, j \in \{1, \ldots, M'\}$ **do**
  if $NCC(\mathbf{z}'_{si}, \mathbf{z}'_{sj}) > \theta_s$ **then**
    $R_{ij} \leftarrow R_{ij} + 1$
  end if
end for
$\mathbf{R} \leftarrow \mathbf{R}/S$
for $i, j \in \{1, \ldots, M'\}$ **do**
  if $R_{ij} \leq \theta_p \vee i = j$ **then**
    $R_{ij} = 0$
  end if
end for
return $\mathbf{R}$

D RELATED ALGORITHMS

D.1 Anomaly Detection

The structure and properties of the five selected anomaly detection algorithms are displayed in Table 6.

OmniAnomaly used techniques such as stochastic variable connection and planar normalizing flow with VAE to learn the robust representations of normal patterns. USAD adversely trained AE to take advantage of the stability of AE and the ability of GAN to isolate anomalies. It is very efficient and robust to the noise in data. Unfortunately, USAD cannot interpret the detected anomalies. SDFVAE learned the representations of normal patterns by factorizing the latent variables into dynamic and static parts to explicitly model invariance to help resist noise in data. InterFusion modeled the normal patterns inside MTS through hierarchical VAE with two stochastic latent variables. It embedded the inter-metric and temporal information into low dimensions. DAGMM utilized an AE to generate the low-dimensional representation and the reconstruction error and fed them into a Gaussian Mixture Model (GMM) to jointly optimize the parameters of AE and GMM simultaneously in an end-to-end fashion. DAGMM cannot interpret anomalies either.

D.2 Convolutional Autoencoder

Autoencoder (AE) comprises two basic units: an encoder and a decoder. The encoder compresses the input into a latent-space representation, which is used by the decoder to reconstruct the input data. AE can be optimized by minimizing the difference between the input and the output. Convolutional autoencoder (CAE) uses convolutional neural networks (CNN) [16] as its encoder and decoder. CNN with 1D convolution kernels is often used for time series analysis [47]. 1D-CAE employs multiple 1D convolution kernels, each sliding along the input with a fixed stride. With the shared-weight architecture, convolution kernels have fewer parameters than dense layers, so deeper network structures and more kernels can be used, which can have better performance than shallow networks [37]. Furthermore, the local connectivity of CNN can preserve the relative spatial information of its input [30].

E DISCUSSION

In developing *OmniCluster*, we have learned the following lessons:

(1) Due to the curse of dimensionality, it is necessary to reduce the dimensionality of high dimensional data for clustering.
(2) 1D-CNN is more effective than 2D-CNN in capturing the shape features of MTS and saving more useful information. It has less computational cost than dense networks and makes the model easier to be trained.
(3) Periodicity is a very important characteristic of MTS. For subsequent applications such as anomaly detection and outlier detection, it is difficult for experts to obtain sufficient information from non-periodic data.
(4) Proper clustering methods and distance measures are essential for the clustering of MTS. We need to choose a clustering algorithm that can effectively identify outliers.
(5) We use 7-day-long MTS (which is not a very long period for a system instance) for experiments, because the patterns of a system change frequently. It is not easy to use a consistent pattern to represent one system. A pattern change is less significant in a long period of seasonal MTS than a short one, making sense to cluster data with a short period.
(6) *OmniCluster* requires that the system instances to be clustered have the same number of metrics and the same number of data points in each metric. Thus, different types of system instances (containers, service instances) that have a different number of metrics or a different number of data points cannot be trained together in *OmniCluster*.
(7) In our scenario, a system instance represents a physical machine. However, *OmniCluster* can be applied to cluster various types of instances, including virtual machines, dockers, containers, *etc.*