Information Sciences 690 (2025) 121590

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Privacy-preserving MTS anomaly detection for network devices through federated learning

Shenglin Zhang [a,c], Ting Xu [a], Jun Zhu [a], Yongqian Sun [a,d,*], Pengxiang Jin [a], Binpeng Shi [a], Dan Pei [b]

[a] *College of Software, Nankai University, Tianjin, China*
[b] *Department of Computer Science and Technology, Tsinghua University, Beijing, China*
[c] *Haihe Laboratory of Information Technology Application Innovation, Tianjin, China*
[d] *Tianjin Key Laboratory of Software Experience and Human Computer Interaction, Tianjin, China*

ARTICLE INFO

Keywords:
Privacy-preserving
Anomaly detection
Federated learning

ABSTRACT

In the context of Maintenance-as-a-Service (MaaS), it is important for device vendors to develop multivariate time series (MTS) anomaly detection models that can accurately identify anomalies without compromising the privacy of customer enterprises' data. In this paper, we investigate the relationship between MTS data patterns and the parameters of unsupervised autoencoder (AE) models and show that they are highly consistent. Building on this insight, we propose a novel unsupervised federated learning (FL)-based framework called *OmniFed*, which cannot only address the heterogeneity of non-independent identically (non-iid) distributed data on different devices, but also achieve high-precision detection of device MTS anomalies while ensuring privacy. Specifically, *OmniFed* is initialized with an AE model and then trains local AE models on individual devices via federated learning. Finally, *OmniFed* clusters devices based on the parameters of the AE models and trains a cluster-specific MTS anomaly detection model using FL. Our experiments on two real-world datasets demonstrate that *OmniFed* achieves an F1-Score of 0.921, significantly higher than the best baseline method.

1. Introduction

Maintenance-as-a-Service (MaaS) has gained much attention recently [1]. In this paradigm, device (*i.e.*, routers, switches, servers, firewalls) vendors provide online tools (say subscription-based tools) to monitor the status of their devices, as shown in Fig. 1. These tools allow customer enterprises to obtain predictive maintenance tasks, which are prohibitive costs for small and medium-sized enterprises. Among the tasks, anomaly detection, which aims at proactively detecting anomalous device behaviors, preventing anomalies from deteriorating into severe failures, and timely mitigating device failures, is vitally essential for customer enterprises [2]. To perform anomaly detection, operators configure diverse types of system-level metrics (*e.g.*, memory utilization, disk I/O, network throughput, CPU utilization) and continuously collect their monitoring data at predefined time intervals (say one or five minutes) [3]. The monitoring metrics of a device thus form a multivariate time series (MTS), as shown in Fig. 2.

* Corresponding author.
E-mail addresses: zhangsl@nankai.edu.cn (S. Zhang), xuting@mail.nankai.edu.cn (T. Xu), zhujuncs@mail.nankai.edu.cn (J. Zhu), sunyongqian@nankai.edu.cn (Y. Sun), jinpengxiang@mail.nankai.edu.cn (P. Jin), shibinpeng@mail.nankai.edu.cn (B. Shi), peidan@tsinghua.edu.cn (D. Pei).

https://doi.org/10.1016/j.ins.2024.121590
Received 24 July 2024; Received in revised form 30 September 2024; Accepted 22 October 2024

Available online 28 October 2024

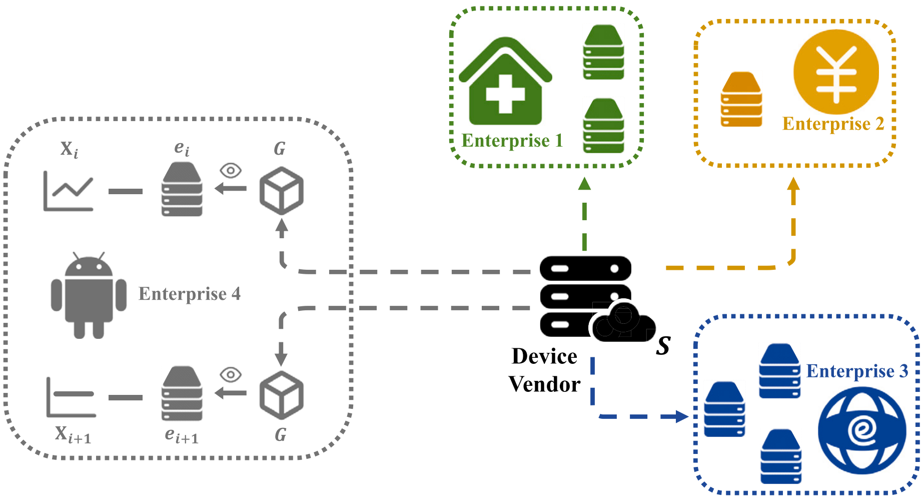


Fig. 1. With MaaS, the device vendor provides an online maintenance tool G for customer enterprises. G conducts anomaly detection based on the MTS data $\mathbf{X}_i$ collected from e_i. In addition, G cannot transfer any data to the device vendor.

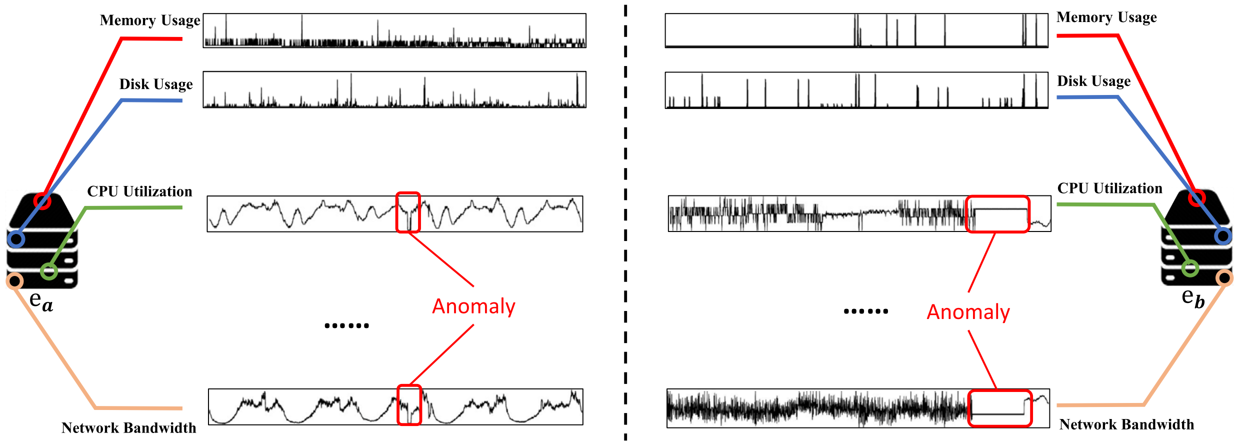


Fig. 2. Different MTS patterns of different devices, with anomalous data labeled in the red box. **Left:** device e_a is deployed in a content service provider, so its MTS data has a strong periodicity, and the unusual dip in the red box means there is an anomaly caused by a network interruption. **Right:** device e_b is deployed in a manufacturing enterprise, and the MTS data pattern is strongly related to manufacturing activities. The unusual stable metric in the red box refers to the disconnection of some key components.

Since deep learning methods can accurately capture the representations of complex MTS data, numerous deep learning-based MTS anomaly detection methods have been proposed for detecting anomalous devices [4–8]. Typically, an MTS anomaly detection model learns the normal patterns of a device and determines that the device becomes anomalous when its MTS behavior deviates from the learned normal patterns. In Maintenance-as-a-Service (MaaS) context, small and medium-sized customer enterprises often lack access to a centralized server/cluster or machine learning experts. As a result, device vendors may deploy a centralized server/cluster to train a deep learning-based MTS anomaly detection model for each device of each customer enterprise. However, customer enterprises may hesitate to share their MTS data with the device vendor due to commercial considerations. Moreover, MTS anomaly detection models typically require a large volume of high-quality training data to accurately learn the normal patterns of the MTS data, as previous research has shown. Unfortunately, it is almost impossible for device vendors to simulate the real-world MTS data of each customer enterprise for model training, as the patterns of MTS data can vary significantly across enterprises that provide different services. This creates a dilemma: on the one hand, it is impractical for device vendors to train an accurate anomaly detection model without sufficient MTS data from customer enterprises. On the other hand, it is risky for customer enterprises to use a device without a precise anomaly detection model. As a result, neither customer enterprises nor device vendors can achieve satisfactory results.

Over the years, federated learning (FL), a distributed learning paradigm training a global model in a central server by aggregating the models from the edge nodes, has been proposed and widely used in privacy-preserving computing [9]. It shares model parameters instead of raw data. In this way, the central server can effectively use the data of edge nodes, and the privacy information will not be leaked [10–14]. Therefore, FL is a promising direction in which to solve the above problem.

However, applying an FL-based MTS anomaly detection model in the MaaS scenario faces the following challenges.

(1) Training a separate detection model for each device may lead to poor performance because of insufficient training data. Therefore, FL usually aggregates the information of edge nodes (customer enterprises' devices in our scenario) to train a global model. Different devices house different applications with varying patterns of traffic [3]. Therefore, different devices typically have different patterns in the MTS data [7]. In other words, the MTS data from different devices is usually non-independent and identically distributed (non-iid). Consequently, training a global FL-based anomaly detection model, which requires that the MTS data have similar normal patterns for *all* devices, can lead to poor performance [7] (see §4.3 for more details).
(2) To address the above challenge, it is intuitive to train a "global model" for a cluster of devices with similar MTS data characteristics [11–15]. Thus, a suitable device clustering algorithm is required instead of metaheuristic optimization algorithms [16]. It is conventional to cluster devices based on their MTS data [6,7,17]. However, the global clustering should be performed on the device vendor. Due to the privacy-preserving requirement, any form of customer enterprises' raw MTS data cannot be shared.
(3) The existing non-iid data-based FL models mainly work on overcoming the imbalance of data labels (*e.g.*, labels of pictures) [15,18–21]. Different from these works, the lack of labeled data exacerbates the difficulty of training an FL-based anomaly detection model.

In this paper, we propose an unsupervised FL-based MTS anomaly detection framework, *OmniFed*, to accurately and privacy-preserving detect device MTS anomalies for customer enterprises in the MaaS scenario. For privacy-preserving purposes, *OmniFed* clusters devices through the parameters of each device's autoencoder (AE) model instead of each device's raw MTS data. For a given cluster, it then trains an FL-based cluster-specific MTS anomaly detection model, which can accurately detect MTS anomalies for each device of this cluster (see § 3).

The contributions of this paper are summarized as follows:

(1) To the best of our knowledge, we performed the first study on the relationship between the MTS data pattern and the parameters of the AE models trained based on the MTS data and found that they are highly consistent.
(2) Based on the above observation, we propose a novel privacy-preserving device clustering method based on the parameters of the AE models trained based on each device's MTS data. In this way, the device vendor indirectly obtains *sufficient* MTS data for training *accurate* anomaly detection models (addressing the first challenge), and customer enterprises' data privacy is preserved simultaneously (addressing the second challenge).
(3) We design an unsupervised FL-based MTS anomaly detection framework *OmniFed*. This makes the model practical in real-world scenarios where anomaly labels are difficult to obtain (addressing the third challenge).
(4) We evaluate *OmniFed* using two real-world datasets, including a real-world dataset collected from a top-tier global content service provider, proving it achieves a 0.921 F1-score, significantly higher than the best baseline method (see § 4). We have released the labeled dataset[1] and source code[2].

2. Preliminary

2.1. Background

As shown in Fig. 1, in MaaS, n devices ($e_1, e_2, ..., e_n$) are deployed across different customer enterprises. To provide maintenance as a service, the device vendor distributes a global anomaly detection model G as a maintenance tool to these devices. For a specific device, G continuously monitors its MTS data and determines whether there is an anomaly (*e.g.*, the red box in Fig. 2). If G detects an anomaly, the operator(s) of the device will receive an alarm and take measures to mitigate the anomaly.

MTS. At time t, the metrics' value of a device forms a vector X^t. Specifically, $X^t = \left\{x_1^t, x_2^t, ..., x_D^t\right\}$ is a D-dimensional vector, where D is the number of metrics. Continuous monitoring of these metrics over some time forms an MTS, denoted by $\mathbf{X}$. $\mathbf{X}$ is a $T \times D$ matrix $\left\{X^1, X^2, ..., X^T\right\}$, where T is the monitoring duration. The sliding window technique is usually applied to divide the raw data $\mathbf{X}$ into sets $\left\{\mathbf{W}^0, \mathbf{W}^1, ..., \mathbf{W}^{T-L}\right\}$, where $\mathbf{W}^t = \left\{X^{t-L+1}, ..., X^{t-1}, X^t\right\}$, L is the length of the sliding window, and $\mathbf{W}^t$ will be the input of the following steps.

Anomaly. An anomaly in the MTS refers to a period when the data deviates from the regular pattern (as shown in the red box of Fig. 2). Typically, an anomaly in the MTS indicates an unexpected situation, *e.g.*, software bugs, hardware failure, or malicious attack, which may degrade the user experience and bring revenue loss to services. Therefore, device operators require prompt and accurate anomaly detection models.

Device MTS anomalies. During the system operation, the MTS data on different devices show anomalies, i.e., periods in which the data deviate from the normal pattern (as shown by the red boxes in Fig. 2). Such anomalies may be triggered by hardware failures, communication problems, or device data anomalies, leading to the inability of the devices to participate in the task normally, which in turn affects the training process and model performance of the entire federated learning system.

[1] https://github.com/OmniFedCD/OmniFed-dataset.
[2] https://github.com/OmniFedCD/OmniFed-code.

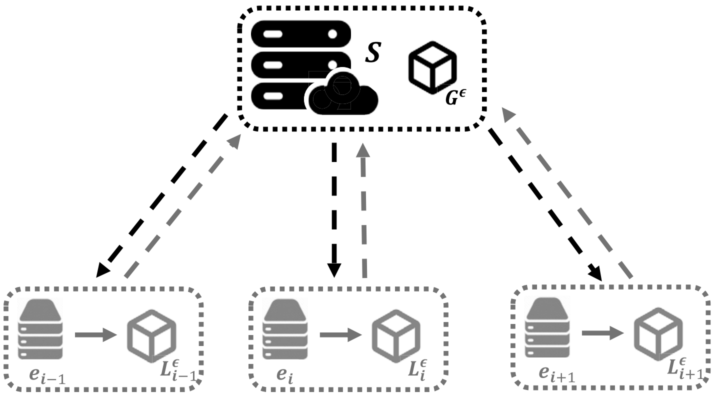


Fig. 3. In FL, the device vendor S distributes global model G^{ϵ} to customer device e_i at epoch ϵ. e_i trains the model with its own data and gets a new local model L_i^{ϵ}. Finally, S aggregates the local models to generate a new global model $G^{\epsilon+1}$.

2.2. Federated learning

FL can train models over device vendors while keeping data localized. It avoids uploading the sensitive raw data of devices and provides MaaS an alternative to train an accurate model by taking advantage of each device's data [9,10,22].

In FL, as shown in Fig. 3, the central server S distributes the global model G^{ϵ} (where ϵ represents the global epochs to each device). To obtain a new global model $G^{\epsilon+1}$ in MaaS, device e_i uses its local dataset $\mathbf{X}_i$ to update the received global model G^{ϵ} and obtain a local model L_i^{ϵ}, which is then uploaded to S. After receiving all local models, S aggregates them and calculates the new global model $G^{\epsilon+1}$:

$$G^{\epsilon+1} = \sum_{i=1}^{m} \frac{1}{m} L_i^{\epsilon}, \tag{1}$$

where m is the number of customer devices in the MaaS. These steps iterate until the global model G^{ϵ} converges. Please note that the combination of FL and anomaly detection is not our main contribution.

2.3. Motivation

To investigate the feasibility of applying FL to detect anomalies on devices from different business scenarios, we conduct an empirical study on a real-world dataset collected from the Web servers of a top global online video service provider with 10+ subsidiary corporations (from now on, we use $\mathbf{D_1}$ to denote this dataset). It consists of the MTS data collected every 30 seconds over seven days from 303 network devices, each deployed with one or more services of a specific subsidiary corporation and monitored with 19 metrics. First, we request three experienced operators to manually cluster the 303 devices into different clusters, each with the same MTS data pattern. Second, we divide the data of the first five days of each device as the training set and the last two days as the test set (see §4.1 for more details).

As aforementioned, different devices can have different MTS data patterns, impacting the performance of MTS anomaly detection methods [6,7]. Thus, we are interested in the relationship between MTS data patterns and the corresponding MTS anomaly detection models. In order to process MTS data effectively, the parameter settings of the anomaly detection model need to be adjusted according to the characteristics of the data to be modeled effectively. We can establish a highly consistent relationship with anomaly detection through preliminary experiments on real case studies, data preprocessing (sliding window technique), model structure settings (encoder and decoder settings), model parameter optimization (sliding window size, Dimension of latent space, Local training epoch, number of encoder layers, etc.), device clustering, reconstruction error (minimization of loss function), and so on, and anomaly detection, etc. Establishing a high degree of consistency in this relationship.

(1) **Data Preprocessing (sliding window technique).** Since the time series length affects the model's input dimension. We split the long time series into multiple fixed-length windows to handle long time series. The MTS data are preprocessed through sliding windows. (specific processing can be found in §2.1)The length of the window is an important parameter of the AE model, which determines whether the model can capture global information (see§4.3 for specific experiments).
(2) **Case Results of Preliminary Experiments.** Through the relationship that data distance is positively correlated with model distance, we can see that the data distance between corresponding layers in two models can measure the distance between two models. Therefore, we need to cluster the devices to mitigate the different MTS data patterns (see§2.3 for the specific reasoning process) and, at the same time, train the data patterns (Non-iid) without revealing private information (see §3.3 for the privacy training process).

(3) **Device Representation of Models (see §3.3.1)**. We train a model for each device and upload each model to the device provider, but the raw data is not shared. To further ensure the privacy and security of the customer's enterprise data, we choose the AE model because of its advantages of downscaling and extracting potential feature representations from the data.
(4) **Structural design and parameter optimisation of the AE model (see§3.3.2).** We upload only a part of the model parameters of the encoder and optimize these parameters, such as the hidden layer dimension compression of the recursive layer and the number of encoder layers (see §4.3 for specific experiments). The encoder maps the data patterns of the MTS into the lower dimensional space of the model, which is then reconstructed to the original dimensions using the decoder. The choice of appropriate parameters for the model allows it to capture better the correlations and dynamics in the multivariate time series.
(5) **Device clustering (see §3.3.2).** According to upload a part of the model parameters (decoder or encoder parameters) to the server. The model distance between each model pair Performs bottom-up group clustering of devices using the (Hierarchical Coalescent Clustering (HAC) and model distance. In this way, device vendors can train MTS anomaly detection models for each device through clustering.
(6) **Reconstruction Error (see §3.4) and Anomaly Detection (see §3.5).** Due to the large number of devices and the large amount of MTS data per device, an unsupervised learning method is needed to capture the MTS data patterns of each device. The reconstruction error can be used for anomaly detection for the unsupervised learning task. The reconstruction error of the AE model reflects the data pattern of MTS. The AE model optimizes the parameters in the dense layers of the encoder and decoder to minimize the loss function, which allows for better adaptation to the heterogeneity of the data (non-iid). An anomaly score threshold is set for each device based on the reconstruction error distribution. Using the Peak Over Threshold (POT) filter, a device is considered abnormal when the anomaly score exceeds the threshold.

By properly optimizing the parameters of the unsupervised AE model, it can capture the complex relationship of MTS data patterns between time dependence and variables. In this way, the relationship between MTS and AE is constructed.

To reflect this relationship between the MTS data patterns and the corresponding MTS anomaly detection models, we investigated the correlation between *data distance* and *model distance*.

As shown in Fig. 4, *Data distance* measures the difference between two MTS data patterns. Because the pattern difference of MTS data can be calculated by Euclidean distance [7], we use the Euclidean distance as the *data distance* between two MTSes,

$$D\left(\mathbf{X}_\alpha, \mathbf{X}_\beta\right) = \|\mathbf{X}_\alpha - \mathbf{X}_\beta\|_2 \tag{2}$$

where $\mathbf{X}_\alpha$ and $\mathbf{X}_\beta$ are the MTSes of device α and β, respectively. A larger $D(\mathbf{X}_\alpha, \mathbf{X}_\beta)$ denotes a larger pattern difference between $\mathbf{X}_\alpha$ and $\mathbf{X}_\beta$.

Model distance measures the difference between two models. We choose the AE model as the basic anomaly detection model, whose superior ability has been extensively proved in previous works [7,8,23,24]. In addition, the AE model structure can reduce dimensionality and extract potential feature representations in the data. The AE model can effectively reduce the dimensionality of MTS data by learning the implicit patterns or features in the time series through its encoder part. In this way, the AE model can reconstruct the time series data through the decoder and detect anomalies through the reconstruction error in time series anomaly detection. And to calculate the distance between two AE models, motivated by [14], we introduce the model distance

$$D\left(L_\alpha, L_\beta\right) = \sum_{j=1}^{k} \|l_{\alpha,j} - l_{\beta,j}\|_2 \tag{3}$$

where L_α and L_β represent the two AE models trained based on the MTS data of device α and device β, respectively. k is the number of layers in the AE model, and $l_{\alpha,j}$ and $l_{\beta,j}$ are the parameter matrices of the j-th layer in L_α and L_β, respectively. This metric treats each layer of the AE model as a matrix and computes the Euclidean distance between corresponding layers in the two AE models.

We train an AE model with the same initial status using the MTS data of each device. For each pair of devices α and β, we calculate their *data distance* $D(X_\alpha, X_\beta)$ and *model distance* $D(L_\alpha, L_\beta)$. As shown in Fig. 4, with the *data distance* rising, the *model distance* increases. To quantify the correlation between *Data Distance* and *Model Distance*, we also calculate the covariance of X (*Data Distance*) and Y (*Model Distance*), which is 6.56, and this result shows that the *Model Distance* are positively correlated.

Observation 1: Different devices house different business applications, which lead to their different MTS data patterns (**non-iid**), reflected by the *model distances*.

The performance degradation of MTS anomaly detection introduced by the different MTS data patterns (i.e., non-iid) can be alleviated by clustering devices through their MTS data patterns [11–15]. However, clustering through the raw MTS data of devices is unrealistic in MaaS because the customer enterprise refuses to share its raw data directly with the vendor for commercial considerations.

Motivated by *Observation 1*, it is intuitive to **cluster devices based on the AE models trained according to their MTS data** (after this, we call it AE model-based clustering) instead of their raw MTS data. We can learn the training data pattern without leaking private information.

We conduct experiments to verify the practicability of clustering devices based on AE models. More specifically, we compare AE model-based clustering with three advanced raw MTS data-based clustering methods: OmniCluster [7], Mc2PCA [25], and SPCA+AED

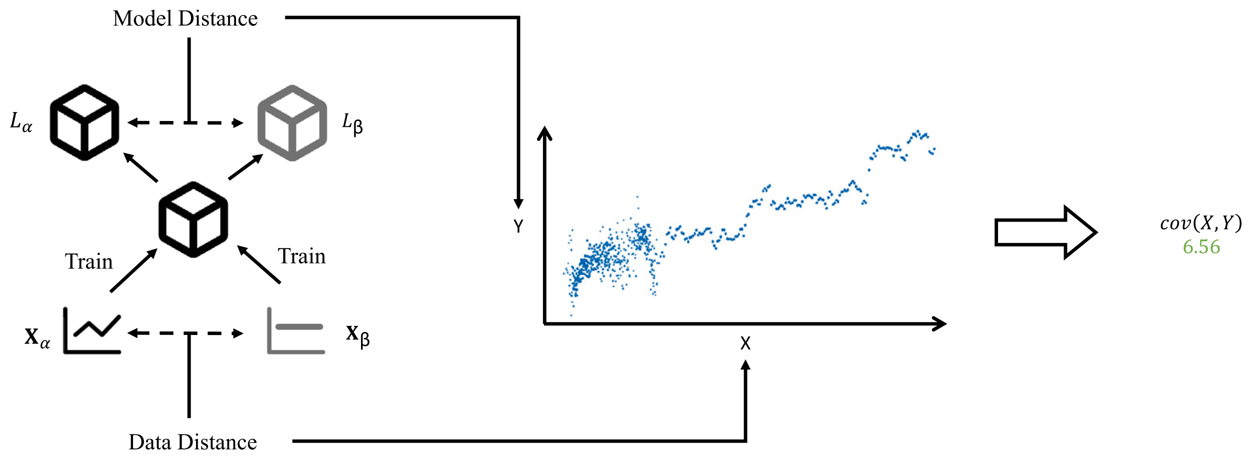


Fig. 4. If two devices have similar MTS data patterns, the two AE models trained based on them will likely have small *model distance*. On the contrary, if two devices have significantly different MTS data patterns, the two AE models trained based on them will likely have a large model distance. In other words, as the *data distance* increases, the *model distance* also rises.

Table 1
AE-model-based clustering (preserving privacy) achieves competitive performance compared to raw MTS data-based clustering (violating privacy)

Method	Privacy	NMI	ARI
AE-based clustering	✓	0.834	0.635
OmniCluster	×	**0.879**	**0.662**
SPCA+AED	×	0.570	0.185
Mc2PCA	×	0.308	0.113

[26]. We run the four clustering methods on $\mathbf{D_1}$ and use two well-adopted metrics: Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI), which is used in previous work [7,17,27], to evaluate the clustering performance. The results are listed in Table 1. Both AE model-based clustering and OmniCluster significantly outperform SPCA+AED and Mc2PCA. While OmniCluster achieves the best performance, the performance of AE model-based clustering is also comparable. Specifically, OmniCluster is only slightly better (0.065 higher in NMI and 0.027 higher in ARI) than AE model-based clustering. The performance gap is *negligible*. However, OmniCluster clusters devices based on their raw MTS data, which cannot be obtained in the MaaS scenario. On the contrary, AE model-based clustering does not require access to raw data.

Observation 2: Compared to clustering methods that rely on raw MTS data, AE model-based clustering can achieve comparable performance while preserving customer enterprises' privacy.

3. Design

3.1. Overview

We propose *OmniFed*, a privacy-preserving FL-based MTS anomaly detection framework, which consists of three stages: device clustering, FL training, and device anomaly detection. Fig. 5 shows the overview of *OmniFed*. To ensure the proper functioning and training of the model, the input data is preprocessed before each stage. In the device clustering stage, we train an AE model for each device and upload the parameters of each model to the device vendor to cluster the devices in a privacy-preserving manner. Then, in the FL-based training stage, we use an unsupervised MTS anomaly detection model in an FL framework to train the local and cluster-specific anomaly detection models iteratively. Finally, in the anomaly detection stage, we deploy these cluster-specific MTS anomaly detection models to conduct anomaly detection for customer enterprise devices. To adapt to the frequent hardware/software changes in network devices and their hosted services, the workflow of *OmniFed* is rerun every day, which is a widely adopted updating strategy for anomaly detection models [4,5,8].

In *OmniFed*, the raw MTS data of the customer enterprise's devices is used only to train the local model of each device in the customer enterprises and is not shared with the device vendor. We only upload the local model parameters, which cannot be restored to the raw MTS data, to the device vendor for clustering or model training. We use ϕ and ι to denote the global and local AE models in §3.3, and G and L to denote the global and local anomaly detection models in §3.4, respectively.

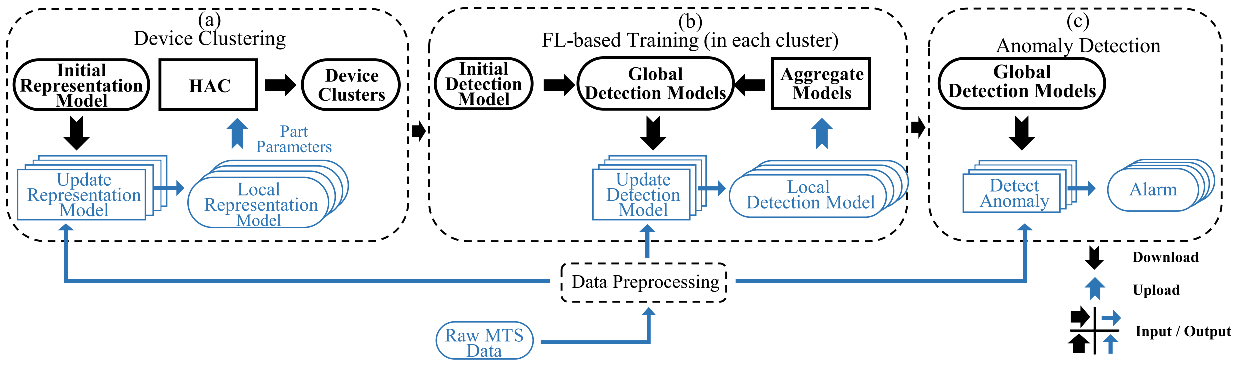


Fig. 5. Overview of *OmniFed*. The processes of device vendors are represented by thick black boxes (upper part), while the processes of customer devices are represented by thin blue boxes (lower part).

3.2. Data preprocessing

In reality, missing values are common in the MTS data, and the magnitude of different time series can vary greatly. Therefore, *OmniFed* preprocesses the raw MTS data to reduce the impact of these issues with the following steps:

Missing values imputation. The MTS data collected from devices may contain missing values due to improper collection or transmission. Training clustering and anomaly detection models with such data can seriously degrade their accuracy. Therefore, it is necessary to impute missing values before inputting the data into the model. To this end, we use linear interpolation to fill these values, as it has been widely proven to be effective and easy to automate [4,5,7,8,24].

Data normalization. Each time series of MTS represents a specific metric of each device. Different metrics represent different aspects of device performance and have different orders of magnitude (*e.g.*, CPU usage ranges from 0 to 1, while disk read/write rate takes any value greater than 0). We need to normalize the data to eliminate the influence of different orders of magnitude. Specifically, the value of metric i at time t, x_i^t, is normalized by its maximum and minimum values (max_i and min_i) with the following formula:

$$\hat{x}_i^t = \frac{x_i^t - min_i}{max_i - min_i} \tag{4}$$

3.3. Device clustering

To address the challenges introduced by non-iid device data and privacy-preserving requirements. We propose a simple yet effective AE-model-based clustering algorithm. More specifically, we train a local AE model for each device and upload the parameters of each model to the device vendor to cluster the devices in a privacy-preserving manner. Devices are often clustered into different clusters, and devices in each cluster have similar data patterns.

3.3.1. Device representation

Due to the large amount of MTS data of each device and the significant number of devices, an unsupervised learning approach is necessary for capturing the MTS data pattern of each device. As aforementioned, unsupervised AE-based or VAE-based models have demonstrated exceptional performance in capturing the normal patterns of MTS data [4,5]. In this paper, we choose an AE-based model to represent a device's MTS data pattern. AE maps the original MTS window data $\mathbf{W}$ to a low-dimensional space Z using an encoder and then reconstructs $\mathbf{Z}$ to the original dimensions using a decoder. This process can be represented as:

$$\mathbf{Z} = Encoder(\mathbf{W}) \tag{5}$$

$$AE(\mathbf{W}) = Decoder(\mathbf{Z}) \tag{6}$$

where the encoder and decoder are composed of a series of dense layers. The objective is to minimize the reconstruction error between the input data and its reconstructed form. This is achieved by optimizing the parameters in the dense layers of the encoder and decoder to minimize the following loss function:

$$\mathcal{L}_{AE} = \|\mathbf{W} - AE(\mathbf{W})\|_2 \tag{7}$$

As discussed in §2.3, the parameters of the trained AE can be used to cluster devices. AE was chosen to represent the devices for two reasons. (1) Effectiveness: AE can effectively learn the normal pattern of MTS data by minimizing the loss function defined in Eq. (7). (2) Efficiency: AE's structure is straightforward, consisting of a single encoder and decoder. This simplicity leads to less training time, memory usage, and calculation costs when computing the *model distance*, simplifying the process and yielding compelling results. To better avoid privacy leakage [10], we uploaded only a portion of the AE parameters (either the decoder or encoder parameters) to S. This presents the reconstruction of raw MTS data and has minimal impact on clustering results.

3.3.2. Clustering

Based on the uploaded parameters of AE models, *OmniFed* uses HAC (Hierarchical Agglomerative Clustering) to cluster the devices. HAC iteratively merges the closest pair of clusters and efficiently performs bottom-up clustering using a pre-computed distance matrix. We chose HAC because it is insensitive to initial parameters such as the number of clusters or distance thresholds, making it easy to visualize the clustering process, thereby enhancing the operator's comprehension of the clustering result [6]. Additionally, HAC has been demonstrated to be suitable for clustering AE models [14]. The device clustering stage comprises three steps, as illustrated in Fig. 5, and we also analyze from protecting data privacy and dealing with non-independent and identically distributed data sets. The HAC is a method of clustering that does not require raw data and relies only on a distance or similarity measure between data. In the *OmniFed* system, this method protects data privacy [28,29] by preventing the device vendors from accessing the raw multivariate time series (MTS) data of the customer's devices. In practice, the Non-iid data [30] generated by network devices are different, i.e., the data from different devices may have different statistical properties. The HAC method can group the device data into different clusters based on their similarity so that a specific anomaly detection model can be customized for each cluster to improve the accuracy of anomaly detection.

Step 1: Initialize the AE model ϕ. For capturing the MTS data pattern of each device, *OmniFed* initializes a simple AE model. Specifically, the device vendor S initializes the neural network layers of the AE model with preset hyperparameters such as window size, dimensions of the hidden layer, and training epochs. The AE model is then built with a randomly set group of parameters in each layer, and S obtains an initial AE model ϕ. This model is then distributed to all customer devices.

Step 2: Training local AE model ι_i. After receiving the initial AE model ϕ, each customer device e_i trains its local AE model ι_i using its dataset $\mathbf{X}_i$ and minimizing Eq. (7). To balance network bandwidth and privacy concerns, e_i uploads only the parameters of the encoder from ι_i to the device vendor S, which does not significantly affect the clustering outcome (discussed in §4.4). This strategy prevents the device vendor from reconstructing the original data from the complete AE model while preserving the ability to reflect the customer device's MTS data patterns. Furthermore, this training step can be executed in parallel on all customer devices

Step 3: Clustering devices based on ι_i. Upon receiving the local models ι_i from each device, the device vendor S computes the *model distance* between each model pair. Next, *OmniFed* applies HAC with *model distance* (Eq. (3)) to group the local models iteratively, starting with the closest pair. This process continues until the most relative distance exceeds the distance threshold (the higher the threshold, the greater the number of devices in each class, while the similarity will be worse). For the outlier devices that cannot be grouped into existing clusters, *OmniFed* assigns the well-trained anomaly detection model of the closest cluster to these devices until there are enough similar devices to form a new cluster. The detailed clustering process is shown below.

(1) **Parameter upload.** Each device uploads some parameters of AE model to server S. Here are the weight parameters of the encoder.
(2) **Distance metric.** In the latent space, the distance between the latent representations of different devices is measured by the Euclidean distance metric. After training the AE, clustering can be performed based on the latent representation of the devices and the compressed features. These features should capture the most important patterns in the input data, and grouping devices with similar latent vectors helps to identify similar devices.
(3) **Stepwise aggregation.** The HAC starts with each device as a separate cluster and gradually aggregates the devices according to the above distance metric to form a hierarchical dendrogram. The two closest clusters are selected and merged at each iteration into a new cluster. This process is repeated until all data points are merged into one cluster or a predetermined number of clusters is reached.
(4) **Hierarchical clustering.** Devices are gradually merged according to the model distance to form a hierarchical dendrogram. The merging process at each level exhibits the similarity between devices, with the closest devices being merged into one class first. The HAC can identify the multi-level similarity between devices, classify them into different clusters, and help identify the operation status, potential failure mode, etc.
(5) **Aggregation process.** The central server S aggregates each device's uploaded parameters according to the federated learning algorithm (such as FedAvg) to generate the global encoder parameters. This global model captures the common characteristics of all devices and can summarize the characteristics of different devices.

With clustering devices with the parameters of the AE model, on the one hand, the devices with different standard patterns can be divided into various clusters, which can solve the first challenge introduced by the non-iid data. On the other hand, the progress of clustering avoids accessing the raw MTS data of customer enterprises, which solves the second challenge induced by the privacy-preserving requirement.

3.4. FL training

OmniFed iteratively trains a cluster-specific anomaly detection model based on the clustering result. We first introduce the anomaly detection model and then present the training process. The main process is to group clients according to data distribution, perform federated learning within each cluster, and train anomaly detection models at the edge via encoders. The local model parameters are uploaded to the device vendor for clustering, and the cloud server builds a more accurate cluster-specific model through bottom-up HAC clustering.

3.4.1. *Anomaly detection model*

Many anomaly detection models can identify anomalous device states, mainly falling into three paradigms: statistical [31,32], supervised methods [33,34] , and unsupervised methods [4,5,23]. Among these, the GAN-AE-based method [8], a type of unsupervised method, stands out for its distinct advantages. Compared to statistical-based methods, GAN-AE-based methods require less operator experience [8]. Compared to supervised methods that heavily rely on labeling information from experts, GAN-AE-based methods eliminate the need for any labeling information [33]. Moreover, GAN-AE-based methods exhibit superior performance compared to other unsupervised solutions while consuming fewer computing and memory resources, such as RNN-based methods [35], AE-based methods [36], and RNN-VAE-based methods [4,5]. Hence, we mainly employ the GAN-AE-based method, USAD, [8] to detect anomalies in *OmniFed*. Please note that *OmniFed* can work with various anomaly detection methods, as discussed in Section 4.6. we have also analyzed the advantages of GAN and AE, respectively, as shown below.

(1) **The advantages of Generative Adversarial Network (GAN).** The GAN [37,38] can learn the complex distribution of data through the adversarial training of generators and discriminators to better capture the normal patterns of data and improve the accuracy of anomaly detection.
(2) **Compression and encoding capabilities of Auto-Encoder (AE).** AE [39,40] can compress the data into a low-dimensional representation, which helps to extract the key features of the data while removing the noise, which is beneficial for anomaly detection. GAN-AE combines the strengths of Generative Adversarial Networks (GAN) and Auto-Encoders (AE). The GAN improves the quality of the generated model through an adversarial process, while AE is an effective technique for encoding and reconstructing data. Combining the two creates a powerful model for generating high-quality representations of normal data while identifying anomalous patterns that differ significantly from these representations. In anomaly detection, this approach is effective in learning normal patterns from complex data and identifying anomalies by comparing the differences between the generated and actual data, which is particularly useful for dealing with high-dimensional and complex datasets.

USAD uses a GAN structure to train two AE methods synchronously, as shown in Fig. 6. The method consists of a shared encoder and two decoders, constituting two AEs that act as a generator (AE_1) and discriminator (AE_2). At time t, a sliding window $\mathbf{W}^t$ is input into the method, and AE_1 and AE_2 are adversarially trained. Specifically, the goal of AE_1 is to deceive AE_2 by reconstructing $\mathbf{W}^t$ with $\mathbf{W}^{t'}$, while AE_2 should distinguish between $\mathbf{W}^{t'}$ and $\mathbf{W}^t$. The loss function of GAN is defined as follows:

$$\min_{AE_1}\max_{AE_2}\|\mathbf{W} - AE_2\left(AE_1(\mathbf{W})\right)\|_2 \tag{8}$$

Meanwhile, AE_1 and AE_2 should also learn the normal pattern of the data. Thus, the loss function of AE_1 and AE_2 is given in Eq. (9) and Eq. (10), respectively.

$$\mathcal{L}_{AE_1} = \frac{1}{n}\mathcal{L}_{AE_1} + \left(1 - \frac{1}{n}\right)\|\mathbf{W} - AE_2\left(AE_1(\mathbf{W})\right)\|_2 \tag{9}$$

$$\mathcal{L}_{AE_2} = \frac{1}{n}\mathcal{L}_{AE_2} - \left(1 - \frac{1}{n}\right)\|\mathbf{W} - AE_2\left(AE_1(\mathbf{W})\right)\|_2 \tag{10}$$

where n denotes the training epoch. With an increase in epochs, the focus of both loss functions shifts from improving data reconstruction ability to adversarial training.

In the detection stage, it will calculate the anomaly score as the following equation,

$$S = \|\mathbf{W} - AE_1(\mathbf{W})\|_2 + \|\mathbf{W} - AE_2\left(AE_1(\mathbf{W})\right)\|_2 \tag{11}$$

where S is the anomaly score, and S will be used to determine the anomaly severity.

3.4.2. *Training*

From the above, we can obtain a cluster-specific models. We describe the specific model as follows. In federated learning, a cluster-specific model is an anomaly detection model specifically trained for a group (or cluster) of clients with similar characteristics. Using bottom-up hierarchical clustering (HAC) on the server side, clients are grouped according to their data distribution, and federated learning is executed within each cluster to train detection models. Each cluster has a proprietary model that is optimised to better fit the unique characteristics of that cluster, thus providing better performance and personalised experience compared to a single global model. With the anomaly detection method, as shown in Fig. 5, the FL training stage consists of three steps:

Step 1: Initialize the cluster-specific detection model G^0. In the beginning, in a specific cluster, *OmniFed* initializes a cluster-specific detection model G^0 and sends G^0 to the device in this cluster.

Step 2: Train local detection model L_i^ϵ. As soon as receiving the cluster-specific detection model G^ϵ, e_i will use local dataset $\mathbf{X}_i$ to update the received anomaly detection model. In this step, each customer device's local detection model L_i^ϵ can undertake certain anomaly detection tasks on local devices, but the performance of L_i^ϵ is limited because of the small amount of training data on e_i. Finally, e_i uploads L_i^ϵ to device vendor S.

Step 3: Aggregate cluster-specific detection model $G^{\epsilon+1}$. After gathering all local detection models, the device vendor will aggregate the parameters of models with FedAvg [9] as follows

$$G^{\epsilon+1} = \sum_{i=1}^{m}\frac{v_i}{v}L_i^\epsilon \tag{12}$$

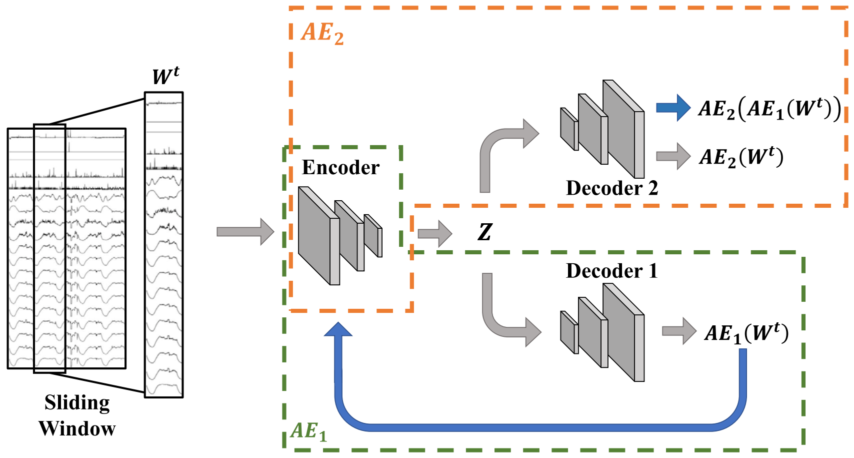


Fig. 6. The network architecture of GAN-AE-based models. AE_1 and AE_2 are adversarially trained and used to detect real-time anomalies.

where v_i denotes the amount of training data at e_i, v denotes the total amount of all training data of all the devices in this cluster, and m denotes the number of devices in this cluster. With aggregation, the cluster-specific model $G^{\epsilon+1}$ can take advantage of the large amount of training data in the cluster. In this way, the cluster-specific detection model $G^{\epsilon+1}$ outperforms each customer device's local detection model L_i^{ϵ}.

After getting a new cluster-specific detection model $G^{\epsilon+1}$, S sends $G^{\epsilon+1}$ to the devices in this cluster and iteratively runs **Step 2** and **Step 3** until the model convergence. The anomaly detection model G can be replaced by any other unsupervised MTS anomaly detection method. This way, *OmniFed* eliminates dependence on the labeled data, which solves the third challenge.

3.5. Device anomaly detection

The unsupervised MTS anomaly detection model iteratively trains local and cluster-specific anomaly detection models for anomaly detection, predictive maintenance, or other tasks. After the FL-based training stage, each cluster will have a cluster-specific MTS anomaly detection model G, and G will be deployed on the corresponding device as shown in Fig. 5. At time t, $\mathbf{W}^t$ is preprocessed before inputting to model G, and G will get an anomaly score. A higher anomaly score implies that $\mathbf{W}^t$ is more likely to be anomalous. To detect anomalies more automatically, *OmniFed* sets anomaly score thresholds for each device with Peaks-Over-Threshold (POT) [41]. When the anomaly score is above the threshold, the device is regarded as anomalous, and the operator is alerted.

Adaptation to new devices. Adding new devices dynamically is very common in MaaS. The operators hope to immediately apply the MTS anomaly detection method to the new devices. *OmniFed* can achieve this goal in the following ways. After the new device uploads its local AE model ι_{new}, S will compare the distance between ι_{new} and the current AE model clusters and assign the cluster-specific MTS anomaly detection model of the closest cluster to the new device. Note that, with a small amount of MTS data, the clustering algorithm can accurately determine the device's cluster, but it is difficult to train an accurate MTS anomaly detection model for this device.

4. Evaluation

In this section, we aim to answer the following research questions (RQs):

RQ1: Is *OmniFed* effective in detecting anomalies?

RQ2: How does the clustering stage affect the performance of *OmniFed*?

RQ3: How do the hyperparameters of *OmniFed* affect its performance?

RQ4: Can *OmniFed* adapt to newly added devices?

RQ5: Is *OmniFed* efficient enough to be deployed to real-world scenarios? Can *OmniFed* be compatible enough to work with different lightweight anomaly detection models other than GAE-AE-based methods?

4.1. Experiment setup

Dataset. In order to ensure the diversity and realisticness of the datasets, we collected two datasets (the non-public dataset $\mathbf{D_1}$ and the public dataset $\mathbf{D_2}$[3]) from the Web servers of a top global online video service provider in application server scenarios. We use two datasets, $\mathbf{D_1}$ and $\mathbf{D_2}$, to validate the performance of *OmniFed* and the baseline methods. For $\mathbf{D_1}$, we request the operators to label

[3] https://s3-us-west-2.amazonaws.com/telemanom/data.zip.

Table 2
Detailed information of the experimental datasets.('#' denotes the amount of data, while the symbol '%' denotes the percentage of anomalies.)

Data	#Entities	#Metrics	Time Interval	#Train	#Test	Anomalies(%)
D1	303	19	30 sec	1440	576	3.70
D2	12	19	5 min	8640	4320	4.61

Table 3
The F1-Score, Precision, and Recall of *OmniFed* and baseline methods on two datasets.

Method	Kernel Parameter	$\mathbf{D_1}$			$\mathbf{D_2}$		
		F1	Precision	Recall	F1	Precision	Recall
OmniFed	$N = 4,\ Z = 10$	**0.921**	**0.873**	**0.975**	**0.932**	**0.907**	0.958
CNN-LSTM	$\eta = 1e-3,\ \rho = 0.3$	0.806	0.790	0.822	0.803	0.671	**0.999**
AMCNN-LSTM	$\eta = 1e-3,\ \rho = 0.3$	0.802	0.711	0.919	0.762	0.742	0.784
FedAnomaly	$\eta = 1e-5,\ h = 128$	0.565	0.522	0.615	0.871	0.855	0.887
PeFad	$l = 100,\ h = 8$	0.430	0.655	0.375	0.608	0.744	0.536

the anomalies in the testing data according to the server's performance, and the labeled dataset is available[4]. More details can be seen in §2.3. To show the generality of *OmniFed*, we experimented with another public dataset, $\mathbf{D_2}$, provided by [5]. The application server datasets ($\mathbf{D_2}$) for multivariate time series anomaly detection and interpretation, which contains 45-day-long MTS data from 12 different entities (each for a server). Each server is monitored with 19 metrics characterizing the status of the server every 5 minutes. The first 30-day data are used for training (the last 30% data in the training set are kept for validation), and the last 15-day data are used for testing. As shown in Table 2, the characteristics of the datasets in the experiment are described, including number of entities, metrics, train and test, time interval, and anomaly rate. The number of entities ranges from 12 to 303, while the number of metrics is 19. It is worth noting that the percentage of anomalies ranges from 3.70% to 4.61%, and that anomalies occur at least once a day almost every day. Specifically, for $\mathbf{D_1}$, the distribution of all types of anomalies is more balanced. While $\mathbf{D_2}$ mainly consists of global anomalies, contextual anomalies and pattern anomalies.

Experimental setup. We list the kernel parameters of each baseline method, as shown in the second column of Table 3, to fit the federated learning clustering scenario in *OmniFed*. In a federated learning clustering scheme, ρ is the gradient compression threshold that can determine a similar threshold to balance communication efficiency and model accuracy. η is the learning rate to ensure stable convergence, h is the hidden layer size to manage the complexity of the clustering task, l is the sliding window size to capture temporal patterns in the data, and N is the number of clusters affecting the convergence of the detection model for each cluster. Z, as the dimension of the latent space, can capture valuable information and make the ***F1-Score*** converge. By tuning these kernel parameters of each method to perform better in a running joint learning environment, the anomaly detection model is ensured to be both efficient and effective for the specific data patterns encountered in each cluster.

Performance metrics. To evaluate anomaly detection performance, we compare the output of an MTS anomaly detection method with the anomaly labels. We denote a false positive (FP) as wrongly reporting a normal MTS sliding window as an anomaly, a false negative (FN) as wrongly reporting an anomalous MTS sliding window as normal, and a true positive (TP) & true negative (TN) as correctly reported for normal and abnormal MTS sliding windows. We use $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F1\text{-}Score = \mathbf{2} \times \frac{\textbf{Precision}\times\ \textbf{Recall}}{(\textbf{Precision+Recall})}$ to represent the anomaly detection accuracy, where ***F1-Score*** reflects the comprehensive capability that considers both ***Precision*** and ***Recall***, so we prefer to pay more attention to ***F1-Score***. Criteria for the ***F1-Score*** evaluation function is a point-adjusted (range-based) [42] evaluation metric. The entire anomalous segment is deemed correctly detected if any point in a ground truth anomaly segment is detected. In our study, we employ the point-adjusted ***F1-Score*** to evaluate the performance of the algorithms. We directly count the TP, FP, and FN to calculate the ***F1-Score*** for each instance. For a dataset, we aggregate the TP, FP, and FN from all entities in the dataset and compute the overall ***F1-Score***.

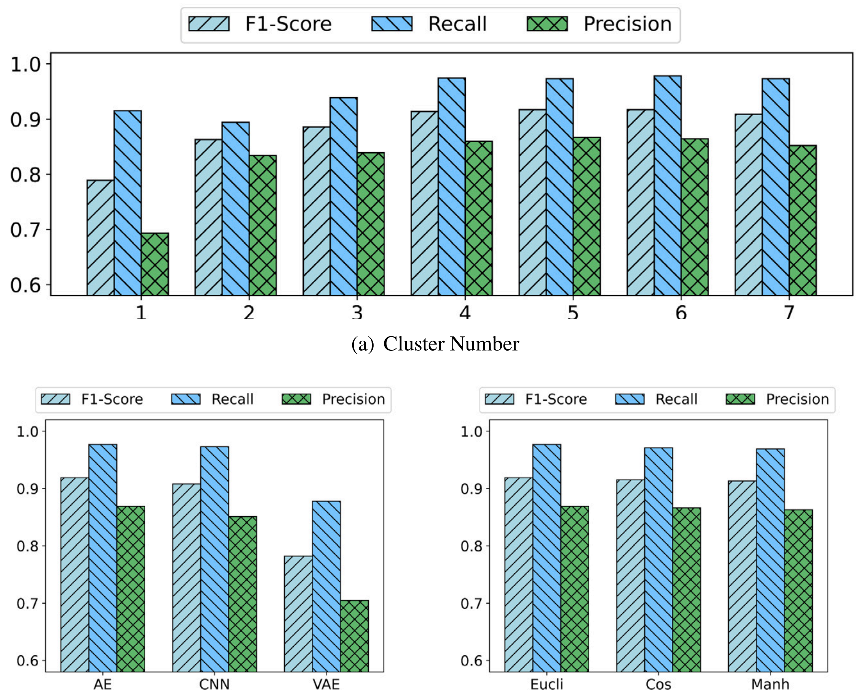
Implementation & Environment. We implement *OmniFed* with Python 3.6 and set hyperparameters best for accuracy, which is investigated in §4.4. We simulated the context of MaaS on a machine with 20 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz and 155 GB RAM in a docker container running Linux version 5.13.0.

Baseline approaches. We compare *OmniFed* with three state-of-the-art FL anomaly detection frameworks: CNN-LSTM [43], AMCNN-LSTM [44], and FedAnomaly [45]. The parameters of these methods are set best for ***F1-Score***.

4.2. Overall Performance (RQ1)

The Table 3 compares the accuracy of *OmniFed* and baseline approaches in anomaly detection. Overall, *OmniFed* gets the best average F1-score of 0.926 on the two datasets, significantly higher than the best baseline methods. Because clustering eliminates the effect of non-iid data, FL improves USAD's performance by indirectly using the device's local dataset to train the cluster-specific

[4] https://github.com/OmniFedCD/OmniFed-dataset



(a) Cluster Number

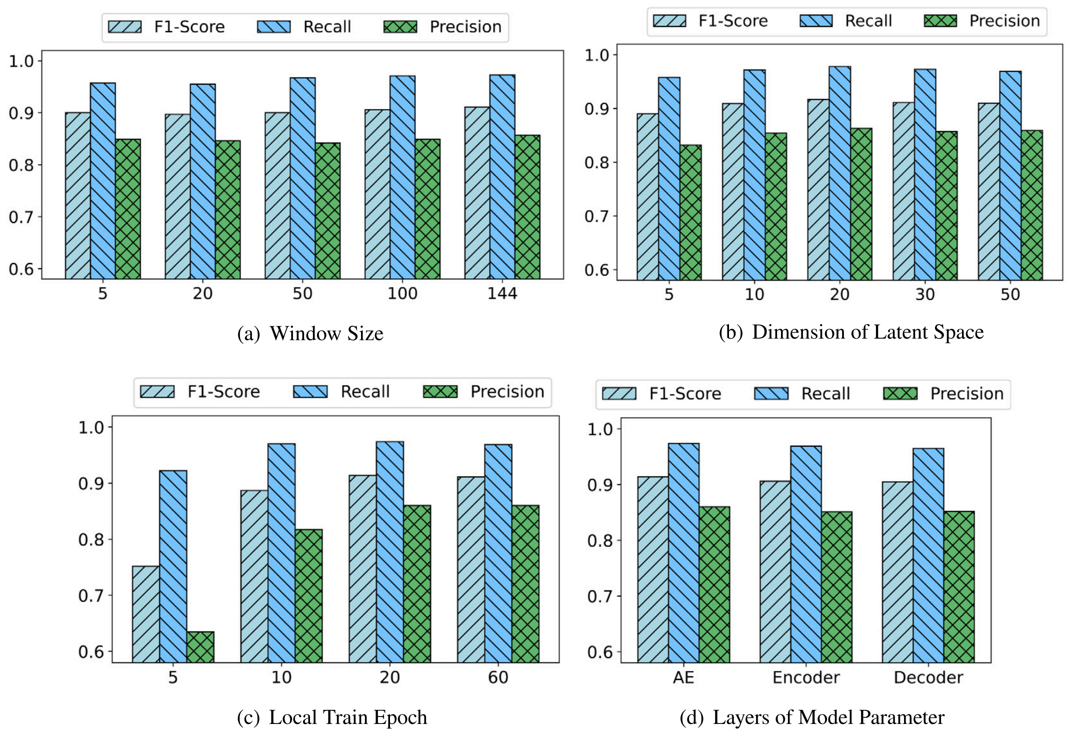
(b) Representation Model

(c) Similarity Measurement Method

Fig. 7. Ablation Study. The effect of (a) clustering and cluster number (1 represents no clustering); (b) representation model; (c) similarity measurement methods.

detection model. FedAnomaly presents the second worst from last performance on $\mathbf{D_1}$ and the second best on $\mathbf{D_2}$ because there is a large proportion of anomalies with pattern shifts in $\mathbf{D_1}$, which FedAnomaly can handle well. PeFad [46] has the worst performance in three indicators on both datasets. Since it detects anomalies in specific time series datasets, this limits the model's generalization ability. CNN-LSTM and AMCNN-LSTM both adopt the strategy of gathering gradients from the local detection model instead of the parameters of the detection model. Although CNN-LSTM achieves the highest Recall on $\mathbf{D_2}$, its precision on $\mathbf{D_2}$ is low. The reason is that $\mathbf{D_2}$ contains numerous anomalies with extreme values that can be easily detected, and the global detection model fails to adapt to diverse data patterns, thus treating normal data as an anomaly.

In addition, we compare the performance metrics of precision and recall between *OmniFed* and other baseline methods in anomaly detection. *OmniFed* has the highest average precision and recall scores of 0.89 and 0.97 on the two datasets, significantly higher than the best baseline method. However, the recall score on In addition, we compare the performance metrics of precision and recall between *OmniFed* and baseline methods in anomaly detection. *OmniFed* has the highest average Precision and recall scores of 0.89 and 0.97 on the two datasets, significantly higher than the best baseline method. However, the recall score on the $\mathbf{D_2}$ dataset is not as high as CNN-LSTM. This is because the $\mathbf{D_2}$ dataset has fewer entities and fewer devices for the federated learning scenario, resulting in fewer kinds of clustering. Specifically, the model tends to predict the majority class and ignore the minority class, so the positive class samples are missed, and more minority class samples cannot be identified, which affects the recognition of positive class samples by the global model and leads to a decrease in recall. However, CNN-LSTM can extract useful local features through filters, and even if the number of entities is small, it can ensure that positive class samples are successfully detected and avoid the situation that the model is biased towards the majority class, so the recall performance is high. In addition, FedAnomaly, PeFad, and AMCNN-LSTM have poor performance on the two big datasets. This is due to the fact that when the model is trained locally, the model updates with more false positives may introduce more noise, and the global model converges slowly in the aggregation process, resulting in extremely low precision and recall performance. The dataset is not as high as CNN-LSTM. This is because the $\mathbf{D_2}$ dataset has fewer entities and fewer devices for the federated learning scenario, resulting in fewer kinds of clustering. Specifically, the model tends to predict the majority class and ignore the minority class, so the positive class samples are missed and more minority class samples cannot be identified, affecting the recognition of positive class samples by the global model and decreasing recall. However, CNN-LSTM can extract useful local features through filters, and even if the number of entities is small, it can ensure that positive class samples are successfully detected and avoid the situation that the model is biased towards the majority class, so the recall performance is high. In addition, FedAnomaly, PeFad, and AMCNN-LSTM have poor performance on the two big datasets. This is since when the model is trained locally, the model updates with more false positives may introduce more noise, and the global model converges slowly in the aggregation process, resulting in extremely low precision and recall performance. In short, *OmniFed* outperforms baseline models thanks to the clustering strategy.



(a) Window Size (b) Dimension of Latent Space (c) Local Train Epoch (d) Layers of Model Parameter

Fig. 8. Effect of hyperparameters. Precision, Recall, and F1-score as a function of (a) the window size; (b) the dimension of the latent space; (c) the local train epoch; and (d) the layers of the model used for clustering.

4.3. Ablation Study (RQ2)

We study the impact of device clustering, representation models, and different model similarity measurement methods. The results are shown in Fig. 7.

Number of clusters. This hyperparameter determines the number of clusters and cluster-specific models. As shown in Fig. 7 (a), cluster number 1 means that models with different standard patterns are aggregated together, and as a result, the F1-Score is only 0.789, the lowest. With the number of clusters increasing, models with different standard patterns are divided into various groups, and the performance of cluster-specific models improves simultaneously. After the number reaches by $N = 4$, the performance converges. A moderate number of clusters not only reduces the differences between devices but also ensures that the device characteristics within each cluster are similar enough to improve the convergence speed of the model.

Different representation model. In Fig. 7 (b), we compare AE with CNN and VAE to show the representation models' differences. AE achieves the best result among these three models, and CNN's performance is close to AE's because both AE and CNN attempt to reconstruct data shape. On the other hand, VAE's performance drops significantly because it focuses on reconstructing the distribution, which does not precisely represent the devices' MTS data.

Different distance method. *Euclidean Distance*, *Manhattan Distance*, and *Cosine Similarity* can be used to measure the similarity between two models. As shown in Fig. 7 (c), the performances of *OmniFed* using these three distances are close to each other because they all effectively reflect the data pattern. As long as the similarity between representation models is measured, the downstream anomaly detection models can achieve satisfactory results.

4.4. Effect of Hyper-parameters (RQ3)

We investigate the impact of different hyperparameters and factors in clustering that can affect anomaly detection performance on $\mathbf{D_1}$.

Window size. The first hyperparameter is the window size of the sliding window L, which determines the length of MTS data for model learning. Fig. 8 (a) presents the obtained results for five different window sizes $L \in [5, 20, 50, 100, 144]$. The best outcome is achieved by $L = 144$. Because the dataset's period is one day, and the values are collected every 30 seconds, there will be $144 \times 20 = 2880$ time points for one day, and the window size of 144 can better capture the patterns of the dataset.

Dimension of latent space. AE maps original data into a low-dimension latent space Z; the higher the dimension of the hidden space, the more practical information can be captured and the less likely it is to overfit. We thus study the effect of latent space and present the result in Fig. 8 (b) with $Z \in [5, 10, 30, 60]$. When $Z = 5$, the dimension of latent space is too small to capture valuable information. When Z reaches 10, the F1-score converges. Hence, we adopt ten as the dimension of latent space.

Table 4
Anomaly detection results of the local and cluster-specific models of the corresponding cluster.

Method	F1	Precision	Recall
Local Model	0.863	0.836	0.892
Cluster Model	**0.919**	0.894	0.945

Table 5
The time consumption of *OmniFed* and those of each stage.

Total Time	Clustering	Local Training	Global Aggregating
1924.69s	443.60s (23%)	1383.23s (73%)	97.86s (4%)

Table 6
The F1-Score, Precision, and Recall of different anomaly detection methods deployed with *OmniFed* or without *OmniFed*.

Method	without *OmniFed*			with *OmniFed*		
	F1	Precision	Recall	F1	Precision	Recall
GAN-AE-based	0.784	0.695	0.898	**0.921**	0.873	0.975
LSTMAE	0.812	0.717	0.935	**0.878**	0.812	0.957
VAEpro	0.587	0.512	0.686	**0.745**	0.693	0.807

Local training epoch. Local training epochs decide the length of AE's training. Generally, with the number of epochs increasing, the training time and training cost also increase. As shown in Fig. 8 (c), when the local training epoch reaches 10, *OmniFed* 's F1-score converges. Considering the performance and training cost, we set the local training epoch at 10.

Layers of AE. The final factor is the layers used for clustering. To further ensure the security of privacy, we only upload part of AE (the encoder or the decoder) to cluster devices because the encoder (decoder) works as a complete module, and the parameters of the encoder (decoder) can be regarded to represent the patterns of the MTS data. As shown in Fig. 8 (d), compared with using the whole AE, using only the encoder (decoder) will not have a noticeable effect on the results but will improve privacy security.

4.5. New Device Addition (RQ4)

The devices in FL are dynamic because enterprises always buy new devices. To consider the newly deployed devices, we do the simulation experiment on $\mathbf{D_1}$ by extracting 10% of the devices as new devices and training models based on the remaining devices. Then, we detect anomalies on a new device with the local model trained with the local data and the corresponding cluster-specific model according to § 3.3, respectively. As listed in Table 4, the F1-score of the cluster-specific model is 5% higher than the local model. So, *OmniFed* can handle the scenario of deploying new network devices.

4.6. Feasibility Study (RQ5)

To evaluate the feasibility of *OmniFed* in real-world scenarios, we simulate and measure the running time of *OmniFed* on the real dataset $\mathbf{D_1}$. As listed in Table 5, *OmniFed* takes 1924 seconds (0.53 hours) to train global anomaly detection models in a MaaS scenario with 303 devices in 10 global epochs, which is an acceptable time compared with the length of the training data, *i.e.*, seven days or even more. Training the local model accounts for 73% of the total time due to the large volume of MTS data collected from each device and the repeated training epochs. In addition, the training step is distributed on each device, so it does not require too many computing resources from the device vendor. The clustering stage accounts for 23% of the time due to the training of AE. Compared with the whole workflow, this is not the majority of the consumed time and is acceptable. The aggregating stage only accounts for 4% of the time because this stage does not involve complicated neural network training. Moreover, to ensure the time efficiency of *OmniFed*, we adopt a lightweight anomaly detection model with a size of only **6.6 MB**. With this size, the AE model will not have too many parameters in the network, which helps to save time in *training, uploading, aggregating, and downloading models*. In summary, *OmniFed* is efficient enough to be employed in real-world scenarios to train the anomaly detection models. After being deployed in the devices, these models will detect anomalies in time, which will cost only a little time (decided by the complexity of the anomaly detection model self).

To show the compatibility of *OmniFed*, we deploy *OmniFed* with different lightweight unsupervised anomaly detection models, including the GAN-AE-based method [8], LSTMAE [24], VAEpro [47]. As listed in Table 6, with *OmniFed*, the performance of all the unsupervised anomaly detection models is improved by an average of 12.1%. Most unsupervised anomaly detection works learn the normal patterns and detect the anomaly when the input data differs significantly from the normal pattern. Still, if the local model's normal patterns differ, the aggregated model will confuse true normal patterns. With the clustering strategy of *OmniFed*, local models with similar normal patterns are grouped and generated into one model specified for their data pattern. In this way, the negative impact of non-iid data is minimized.

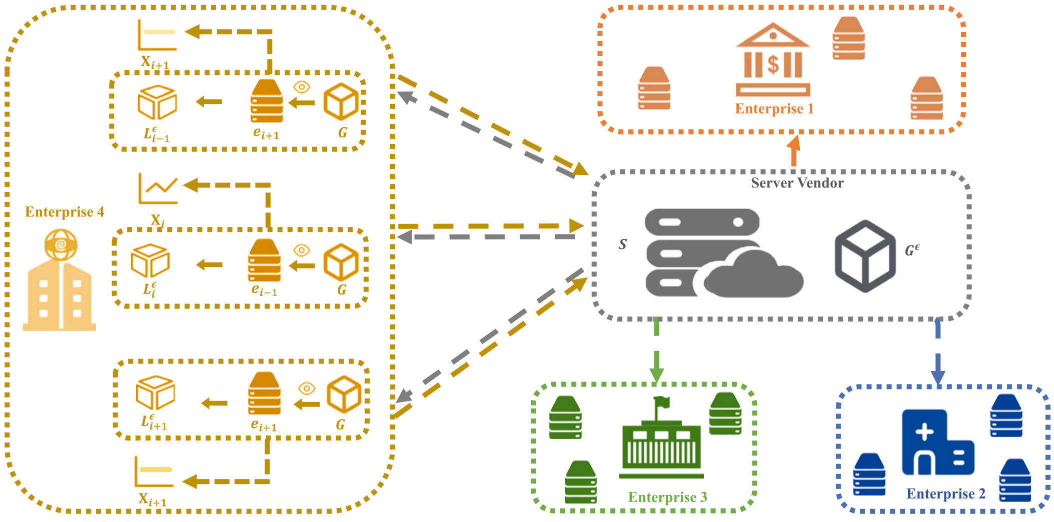


Fig. 9. In the MaaS, OmniFed is applied in FL to detect anomalies on servers from different business operations scenarios.

Conclusion: *OmniFed* performs better detecting anomalies than CNN-LSTM, AMCNN-LSTM, and FedAnomaly (§4.2). Moreover, the ablation study proves the necessity of clustering. The hyperparameters of *OmniFed* can be set to a suitable value by referencing §4.4. Besides, *OmniFed* is suitable in the scenario where new devices will be deployed. *OmniFed* is also promising in deploying in real-world scenarios, for it is time-efficient and compatible with various downstream anomaly detection methods.

5. Discussion

5.1. The application scenarios of OmniFed

In the MaaS scenario, to investigate the feasibility of *OmniFed* application in FL, we detect anomalies on servers from different business operations scenarios. As shown in Fig. 9, the experiment can be performed by simulating a MaaS scenario in a running docker container. This contains multiple server nodes, each representing a client device.

We conducted an empirical study on a real-world dataset $\mathbf{D_2}$ collected from application server devices of large Internet companies. It includes 12 servers deployed in 4 different customer enterprises (e.g., $e_1, e_2, \ldots, e_{12}$ are deployed in finance, hospitals, government, and internet) of 45 days long MTS data. Three experienced operations experts manually clustered the 12 devices into different clusters, each with the same MTS data pattern. Each server was monitored every 5 minutes using 19 metrics, including CPU-related metrics, memory-related metrics, network metrics, virtual machine metrics, etc. The first 30 days of data are used for training, and the last 15 days are used for anomaly detection across devices. The sensitive information in the data (e.g., IP address, metric names, concrete timestamps) has been removed to protect the confidentiality of data providers. To provide the Maas, the central server S distributes the global model G^{ϵ}. The server providers distribute the global anomaly detection model G^{ϵ} as a maintenance tool to these local servers e_i. e_i uses its local dataset $\mathbf{X}_i$ to update the received global model G^{ϵ} and obtains the local model L^{ϵ}_i, which is then uploaded to the central server S. The server e_i then uploads the new global anomaly detection model $G^{\epsilon+1}$ to monitor its MTS data continuously. We use an online tool to monitor the status of individual client enterprise servers and determine if there are any anomalies. Upon testing, it can be seen that *OmniFed* takes about 0.46 hours to train the global anomaly detection model using 12 servers in 10 global epochs. This is an acceptable time compared to the time taken to train the data, which is 30 days or more.

In addition, we found that G^{ϵ} detects anomalous MTS data over one month, and operators can detect anomalies once a day on average. By performing the bottom-up HAC clustering in federated learning, we achieve that the cluster-specific model achieves about 15% improvement in detection accuracy over the single-server-trained model, as well as about 20% reduction in false positives for the federated-trained model. In this way, the *OmniFed* provides server providers with high-quality and efficient MaaS operation and maintenance services with high detection rates, low false positives, and low risk of data leakage, thus enhancing customer satisfaction. Moreover, *OmniFed* reduces O&M costs by automating anomaly detection and predictive maintenance in the MaaS, achieving anomaly pattern recognition capability and response speed, reducing manual intervention, and reducing repair costs and downtime losses due to failures. Through the above-detailed explanation and experimental description, we can clearly see the practical application and significant benefits of the *OmniFed* framework in the MaaS scenarios.

5.2. Limitations and future work

5.2.1. The privacy protection of OmniFed

In order to better protect the privacy security of local device data in practical application scenarios, we reduce the potential privacy risks through multi-level protection in both the model design phase and model training phase and by combining the following techniques.

(1) **Model Design Phase**
Avoiding partial parameter leakage. We upload only some of the AE model parameters to avoid leaking certain information about the original data. In particular, the decoder or encoder parameters contain sensitive features, and the attacker cannot directly obtain information about the original data, which reduces the risk of data leakage.
Reconstruction attack. We only upload part of the model parameters, reducing the data leakage risk. However, the attacker may still be able to reverse reason about the original data by inferring the model structure and training data distribution. The exposure of decoder parameters may allow an attacker to reconstruct part of the original input data, especially if the latent representation generated by the encoder contains sufficient information.
To mitigate the risk of reverse engineering attacks, we designed the AE model to minimize the amount of identifiable personal information contained in the latent representation. Combined with a priori knowledge [48–50], the model structure is optimized to make it difficult to reconstruct the original data even if the decoder is compromised.

(2) **Model Training Phase**
Avoid breaching the server. In federated learning, randomly initializing the unuploaded partial parameters in each round of training can reduce the risk of information leakage caused by fixed parameters. In addition, multiple participants slice and dice the AE model parameters, randomly selecting different segments to upload their partial parameters each time, making it more difficult for attackers to reconstruct the complete model. This random initialization and parameter slicing allows us to learn training data patterns without leaking private information. An attacker cannot combine partial parameters uploaded by multiple clients to reconstruct the original data or infer sensitive information to break the central server.
Clustering leakage attack. Even if partial parameters are uploaded using parameter slicing, and the clustering model uses the output of the auto-encoder as input, the decoder or encoder parameters may reveal critical information about the clustering of the data. An attacker can indirectly leak the information structure of the original data by analyzing the uploaded parameters and inferring the centroids of the clusters or the similarities between the classes.
To counteract the risk of back-propagation of clustering results, we cluster devices based on AE models trained on their MTS data (AE model-based clustering) instead of their original MTS data. The gradients shared during model aggregation do not leak information about the local data, which prevents the server from directly accessing or inferring the local model parameters of each participant. This makes it difficult to recover the original data even if the server is compromised.

5.2.2. The scalability of OmniFed

To extend *OmniFed* 's scalability in scenarios involving many devices or diverse data sources, it's essential to tackle challenges related to clustering, edge device synchronization, and dimensionality reduction. Below is an in-depth discussion of these aspects.

(1) **Clustering and Selecting Representative Devices or Data Types**
The *OmniFed* employs a clustering mechanism to handle a large number of devices efficiently. The key is that the number of clusters does not scale proportionally with the number of devices, remaining within the range of 10-20 clusters, where each cluster represents a distinct data source type.
Challenge. The risk here lies in ensuring that each cluster accurately represents the diversity of data types within the system. If the clustering becomes too generalized, it might lead to over-representation of some data types while under-representing others.
Solution. To maintain scalability, selecting highly representative devices within each cluster ensures that only the most critical updates are sent to the central server. This approach balances the trade-off between efficiency and diversity, ensuring that the system doesn't become overwhelmed with redundant updates as more devices join the network. Additionally, dynamic clustering [16] could be used to recalibrate clusters based on evolving data patterns, thus maintaining robust model performance across various data types.

(2) **Synchronizing Training Across Edge Devices with Resource Matching**
The *OmniFed* emphasizes the synchronization of edge devices during distributed training. Each device contributes model updates based on its computational capacity, which is carefully matched to its task. Importantly, central aggregation handles updates without significant overhead, ensuring the system doesn't incur increased training costs, even as device numbers grow.
Challenge. In large-scale systems, heterogeneous devices with varying computational powers and connectivity could disrupt synchronization. If slower or less capable devices fail to communicate efficiently, the overall system performance may be affected.
Solution. Asynchronous updates can be introduced to address this issue. Instead of requiring all devices to synchronize simultaneously, devices submit updates [51] based on their schedules and capabilities. This reduces bottlenecks and ensures that more capable devices continue contributing without being delayed by slower ones. Moreover, the system could prioritize receiving updates from more representative [52] clusters to maintain model quality without waiting for every device to report.

(3) **Dimensionality Reduction to Minimize Communication Overhead**

Enhancing *OmniFed* 's scalability is one of the core strategies to reduce the communication burden by applying downsizing techniques. The model parameters, typically in high-dimensional space [53], are mapped into lower dimensions, thus reducing the volume of data transmitted between edge devices and the central server.
Challenge. The major risk in this approach is information loss. If the dimensionality reduction is too aggressive, important features might be lost during compression, negatively impacting the model's accuracy.
Solution. To mitigate this risk, *OmniFed* could adopt advanced dimensionality reduction methods such as model pruning, quantization, and federated dropout. These methods allow for selective reduction, ensuring that only the least important model parameters are dropped or compressed. This approach ensures the system can scale to large device networks without compromising model performance by preserving critical information while minimizing the data payload.

Through the above analysis, the *OmniFed* framework can effectively address the challenges posed by the number of devices and diverse data sources, maintaining good scalability. These strategies not only help control computational and communication overhead but also ensure the model's performance and data diversity, making the framework more adaptable for real-world applications. However, challenges such as device heterogeneity and network instability may still arise in practical deployments. Future research could further explore ways to tackle these issues, enhancing *OmniFed* 's practicality and robustness.

6. Related work

6.1. Federated learning

FL is a distributed framework proposed for more privacy-preserving learning [20,22]. Briggs et al. [14] use the supervised model's parameters to cluster devices into groups to soften the impact of non-iid data. Wang et al. [15] prove that the number of labels can directly act on the model's parameters. Abdel-Basset et al. [13] uses the semi-supervised model to detect anomalies in intelligent grids with FL. CNN-LSTM [44] and AMCNN-LSTM [43] are proposed to detect anomaly with using gradient compression to upload parameters. FedAnomaly [45] conducts a light-weight detection model to detect anomalies in FL, and only shares model updates (such as gradients or model parameters) with a central server, rather than sharing raw data. However, it shows that the different data shapes can lead to a worse global model performance. Fed-ExDNN [55] allows local devices to learn and send only the necessary model updates to the central server, which ensures that sensitive data remains on the local devices. But, it does not adapt to the differences between edge devices. Fed-SMAE [54] specifically uses the distributed training framework to keep raw data on local devices, thereby reducing the risk of data leakage and privacy concerns. However, it does not take into account the problem of data heterogeneity and differences in characteristics between device groups in different customer enterprises. PeFad [46] ensures the privacy of the original data and the validity of the synthesized data by constraining the mutual information and the model distance, but longer synthesized time series can introduce redundant or noisy information, which can degrade the model performance. These studies protect data privacy by deploying a federated learning framework on edge devices, which enables individual devices to collaboratively train anomaly detection models without the need to transmit the raw dataset to a central server. However, they cannot solve the problem of different normal patterns for MTS anomaly detection. *OmniFed* studies this topic on an unsupervised model and proposes the clustering solution to combine FL with anomaly detection.

6.2. Anomaly detection

The anomaly detection model for MTS can be roughly classified into the statistical-based model, supervised DL model, and unsupervised DL model. Due to the absence of labeling, we mainly use the unsupervised DL model.

Statistical-based model. This model type identifies all data far from the mean center as anomalies [31,32], which uses statistical methods to implement this target. Statistical-based models are lightweight and require no training but usually require lots of manual setups.

Supervised model. Supervised strategies are adopted to train the anomaly detection model for more complex MTS. This work uses machine learning methods, such as SVM [34] and KNN [33], to achieve anomaly detection. Supervised models need a labeled dataset, which is difficult to obtain in a real-world environment.

Unsupervised model. For unsupervised training, OmniAnomaly [4], InterFusion [5], and SDFVAE [23] respectively add various modules to the model to improve its effectiveness, resulting in the model parameter scale being too large. Li et al. [22] prove that the model in FL should be lightweight for the communication cost. Based on this need, LSTMAE [24] uses stacked LSTM networks for anomaly/fault detection. VAEpro [47] proposes an approximate probabilistic model to find better latent distribution for anomalous input. USAD [8] uses adversarial training to train the model for detecting anomalies.

6.3. Clustering strategies

Existing studies focus on clustering the MTS data with raw data. ROCKA [17] cluster time series with DBSCAN with normalized cross-correlation distance. SPCA+AED [26] uses fuzzy clustering to consist of the PCA similarity factor with the average-based Euclidean distance. Mc2PCA [25] constructs common projection axes as the prototype of each cluster and uses the reconstruction error to reassign the cluster member. OmniCluster [7] uses 1D-CNN and future extract before clustering MTS to reduce calculating time and improve clustering accuracy.

All these methods cluster on the raw data, including sensitive information, and *OmniFed* adopts a strategy of using model parameters instead, which is less likely to result in a privacy breach.

7. Conclusion

In the MaaS scenario, the device vendors need to protect the privacy of the customer enterprises' data in MTS anomaly detection. Our pilot experiments on the MTS of 303 devices show that the MTS data patterns are highly consistent with the parameters of AE models trained based on them. To preserve privacy in anomaly detection, we propose *OmniFed*, an unsupervised FL-based MTS anomaly detection framework. *OmniFed* clusters devices based on the parameters of AE models trained using the MTS data of each device and then trains cluster-specific anomaly detection models in a privacy-preserving manner. Finally, *OmniFed* distributes cluster-specific models to edge devices for real-time anomaly detection. Evaluation using two real-world datasets shows that *OmniFed* outperforms existing approaches and exhibits excellent promise for deployment in real-world scenarios.

CRediT authorship contribution statement

Shenglin Zhang: Writing – review & editing, Formal analysis, Data curation. **Ting Xu:** Writing – review & editing, Supervision, Conceptualization. **Jun Zhu:** Writing – original draft, Conceptualization. **Yongqian Sun:** Writing – review & editing, Validation, Supervision, Software, Formal analysis. **Pengxiang Jin:** Software, Resources, Investigation. **Binpeng Shi:** Project administration, Formal analysis. **Dan Pei:** Writing – review & editing, Validation, Resources, Project administration, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (62272249, 62302244, 62072264).

Data availability

Data will be made available on request.

References

[1] INFRASPEAK, Maintenance-as-a-service: what is maas?, https://blog.infraspeak.com/maintenance-as-a-service/, 2022.
[2] P. Notaro, J. Cardoso, M. Gerndt, A survey of aiops methods for failure management, ACM Trans. Intell. Syst. Technol. 12 (2021) 1–45.
[3] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, D. Pei, et al., Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems, USENIX ATC 2021, 2021, pp. 413–426.
[4] Y. Su, Y. Zhao, C. Niu, W. Sun, D. Pei, et al., Robust anomaly detection for multivariate time series through stochastic recurrent neural network, in: Proceedings of the 25th ACM SIGKDD, 2019, pp. 2828–2837.
[5] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, D. Pei, et al., Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding, in: Proceedings of the 27th ACM SIGKDD, 2021, pp. 3220–3230.
[6] M. Sun, Y. Su, S. Zhang, Y. Cao, Y. Liu, D. Pei, et al., CTF: anomaly detection in high-dimensional time series with coarse-to-fine model transfer, in: IEEE INFOCOM 2021, IEEE, 2021, pp. 1–10.
[7] S. Zhang, D. Li, Z. Zhong, J. Zhu, M. Liang, J. Luo, Y. Sun, Y. Su, S. Xia, Z. Hu, Y. Zhang, D. Pei, J. Sun, Y. Liu, et al., Robust system instance clustering for large-scale web services, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 1785–1796.
[8] J. Audibert, P. Michiardi, et al., Usad: unsupervised anomaly detection on multivariate time series, in: Proceedings of the 26th ACM SIGKDD, 2020, pp. 3395–3404.
[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. Arcas, et al., Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.
[10] L. Fowl, J. Geiping, W. Czaja, M. Goldblum, T. Goldstein, et al., Robbing the fed: directly obtaining private data in federated learning with modified models, preprint, arXiv:2110.13057, 2021.
[11] J. Kwon, B. Jung, H. Lee, S. Lee, et al., Anomaly detection in multi-host environment based on federated hypersphere classifier, Electronics 11 (2022) 1529.
[12] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, et al., Federated learning with non-iid data, preprint, arXiv:1806.00582, 2018.
[13] M. Abdel-Basset, N. Moustafa, H. Hawash, Privacy-preserved generative network for trustworthy anomaly detection in smart grids: a federated semi-supervised approach, IEEE Trans. Ind. Inform. (2022).
[14] C. Briggs, Z. Fan, P. Andras, Federated learning with hierarchical clustering of local updates to improve training on non-IID data, in: IJCNN 2020, IEEE, 2020, pp. 1–9.
[15] L. Wang, S. Xu, X. Wang, Q. Zhu, et al., Addressing class imbalance in federated learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2021, pp. 10165–10173.
[16] L. Abualigah, Metaheuristic Optimization Algorithms: Optimizers, Analysis, and Applications, Elsevier, 2024.

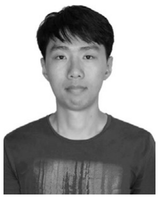
[17] Z. Li, Y. Zhao, R. Liu, D. Pei, et al., Robust and rapid clustering of kpis for large-scale anomaly detection, in: 2018 IEEE/ACM 26th IWQoS, IEEE, 2018, pp. 1–10.
[18] Y. Qin, H. Matsutani, M. Kondo, A selective model aggregation approach in federated learning for online anomaly detection, in: 2020 International Conferences on iThings and GreenCom and CPSCom and SmartData and Cybermatics, IEEE, 2020, pp. 684–691.
[19] H. Wen, Y. Wu, C. Yang, H. Duan, et al., A unified federated learning framework for wireless communications: towards privacy, efficiency, and security, in: 2020-IEEE INFOCOM WKSHPS, IEEE, 2020, pp. 653–658.
[20] J. Wettlaufer, Property inference-based federated learning groups for collaborative network anomaly detection, Electron. Commun. EASST 80 (2021).
[21] Y. Zhao, J. Chen, D. Wu, J. Teng, S. Yu, et al., Multi-task network anomaly detection using federated learning, in: Proceedings of the Tenth SOICT, 2019, pp. 273–279.
[22] T. Li, A.K. Sahu, A. Talwalkar, V. Smith, Federated learning: challenges, methods, and future directions, IEEE Signal Process. Mag. 37 (2020) 50–60.
[23] L. Dai, T. Lin, C. Liu, B. Jiang, Y.W. Liu, X. Zhen, Z. Zhang, et al., SDFVAE: static and dynamic factorized vae for anomaly detection of multivariate cdn kpis, in: Proceedings of the Web Conference 2021, 2021, pp. 3076–3086.
[24] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, et al., Long short term memory networks for anomaly detection in time series, in: ESANN, 2015, p. 89.
[25] H. Li, Multivariate time series clustering based on common principal component analysis, Neurocomputing 349 (2019) 239–247.
[26] C.H. Fontes, H. Budman, A hybrid clustering approach for multivariate time series–a case study applied to failure analysis in a gas turbine, ISA Trans. 71 (2017) 513–529.
[27] K.G. Dizaji, A. Herandi, C. Deng, W. Cai, H. Huang, Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5736–5745.
[28] Z. Liu, J. Guo, W. Yang, J. Fan, K.Y. Lam, J. Zhao, Privacy-preserving aggregation in federated learning: a survey, IEEE Trans. Big Data (2022).
[29] X. Yin, Y. Zhu, J. Hu, A comprehensive survey of privacy-preserving federated learning: a taxonomy, review, and future directions, ACM Comput. Surv. 54 (2021) 1–36.
[30] Z. Lu, H. Pan, Y. Dai, X. Si, Y. Zhang, Federated learning with non-iid data: a survey, IEEE Int. Things J. (2024).
[31] W.A. Chaovalitwongse, Y. Fan, R.C. Sachdeo, On the time series k-nearest neighbor classification of abnormal brain activity, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 37 (2007) 1005–1016.
[32] E.W. Grafarend, Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models, de Gruyter, 2006.
[33] N. Laptev, S. Amizadeh, I. Flint, Generic and scalable framework for automated time-series anomaly detection, in: Proceedings of the 21th ACM SIGKDD, 2015, pp. 1939–1947.
[34] J. Ma, S. Perkins, Time-series novelty detection using one-class support vector machines, in: Proceedings of the IJCNN 2003, IEEE, 2003, pp. 1741–1745.
[35] K. Hundman, V. Constantinou, C. Laporte, L. Colwell, et al., Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding, in: Proceedings of the 24th ACM SIGKDD, 2018, pp. 387–395.
[36] B. Zong, Q. Song, M.R. Min, W. Cheng, C. Lumezanu, D. Cho, H. Chen, et al., Deep autoencoding gaussian mixture model for unsupervised anomaly detection, in: International Conference on Learning Representations, 2018.
[37] X. Wang, H. Guo, S. Hu, M.C. Chang, S. Lyu, Gan-generated faces detection: a survey and new perspectives, ECAI 2023 (2023) 2533–2542.
[38] W. Xia, Y. Zhang, Y. Yang, J. Xue, B. Zhou, M.H. Yang, Gan inversion: a survey, IEEE Trans. Pattern Anal. Mach. Intell. 45 (2022) 3121–3138.
[39] K. Han, Y. Wang, C. Zhang, C. Li, C. Xu, Autoencoder inspired unsupervised feature selection, in: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2018, pp. 2941–2945.
[40] M. D'Souza, C.E. Van Munster, J.F. Dorn, et al., Autoencoder as a new method for maintaining data privacy while analyzing videos of patients with motor dysfunction: proof-of-concept study, J. Med. Internet Res. 22 (2020) 16669.
[41] A. Siffer, P.A. Fouque, A. Termier, C. Largouet, et al., Anomaly detection in streams with extreme value theory, in: Proceedings of the 23rd ACM SIGKDD, 2017, pp. 1067–1075.
[42] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, H. Qiao, et al., Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 187–196.
[43] Y. Liu, N. Kumar, et al., Communication-efficient federated learning for anomaly detection in industrial Internet of things, in: GLOBECOM 2020-2020, IEEE, 2020, pp. 1–6.
[44] Y. Liu, S. Garg, et al., Deep anomaly detection for time-series data in industrial IoT: a communication-efficient on-device federated learning approach, IEEE Int. Things J. 8 (2020) 6348–6358.
[45] K. Zhang, Y. Jiang, L. Seversky, C. Xu, D. Liu, H. Song, et al., Federated variational learning for anomaly detection in multivariate time series, in: 2021 IEEE IPCCC, IEEE, 2021, pp. 1–9.
[46] R. Xu, H. Miao, S. Wang, P.S. Yu, J. Wang, PeFAD: a parameter-efficient federated framework for time series anomaly detection, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2024, pp. 3621–3632.
[47] Y. Ikeda, K. Tajiri, Y. Nakano, K. Watanabe, K. Ishibashi, Estimation of dimensions contributing to detected anomalies with variational autoencoders, preprint, arXiv:1811.04576, 2018.
[48] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Dwarf mongoose optimization algorithm, Comput. Methods Appl. Mech. Eng. 391 (2022) 114570.
[49] J.O. Agushaka, A.E. Ezugwu, A.K. Saha, J. Pal, L. Abualigah, S. Mirjalili, Greater cane rat algorithm (GCRA): a nature-inspired metaheuristic for optimization problems, Heliyon (2024).
[50] A.E. Ezugwu, J.O. Agushaka, L. Abualigah, S. Mirjalili, A.H. Gandomi, Prairie dog optimization algorithm, Neural Comput. Appl. 34 (2022) 20017–20065.
[51] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Gazelle optimization algorithm: a novel nature-inspired metaheuristic optimizer, Neural Comput. Appl. 35 (2023) 4099–4131.
[52] M. Ghasemi, M. Zare, A. Zahedi, M.A. Akbari, S. Mirjalili, L. Abualigah, Geyser inspired algorithm: a new geological-inspired meta-heuristic for real-parameter and constrained engineering optimization, J. Bionics Eng. 21 (2024) 374–408.
[53] G. Hu, Y. Guo, G. Wei, L. Abualigah, Genghis Khan shark optimizer: a novel nature-inspired algorithm for engineering optimization, Adv. Eng. Inform. 58 (2023) 102210.
[54] J. Huang, D. Xu, T. Yang, Fed-SMAE: federated-learning based time series anomaly detection with shared memory augmented autoencoder, in: 2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS), 2024, pp. 1–6.
[55] W. Zhu, D. Song, Y. Chen, W. Cheng, B. Zong, T. Mizoguchi, Deep federated anomaly detection for multivariate time series data, in: 2022 IEEE International Conference on Big Data (Big Data), IEEE, 2022, pp. 1–10.

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

Ting Xu received the M.E. degree in Software Engineering from Central South University in 2020. She is currently a Ph.D. student at the College of Software, Nankai University. Her research interests include anomaly detection and failure diagnosis. She is an IEEE Member.

Jun Zhu received the B.E. and M.S. degrees in in software engineering from Nankai University, Tianjin, China. His research interests include anomaly detection and anomaly localization.

Yongqian Sun received a B.S. degree in statistical specialty from Northwestern Polytechnical University, Xi'an, China, in 2012, and Ph.D. in computer science from Tsinghua University, Beijing, China, in 2018. He is currently an assistant professor at the College of Software, Nankai University, Tianjin, China. His research interests include anomaly detection and root cause localization in service management.



Pengxiang Jin received the B.E. and M.S. degrees in software engineering from Nankai University, Tianjin, China. His research interests include anomaly detection and anomaly localization.

Binpeng Shi received the B.S.in software engineering from the College of Software, Nankai University, Tianjin, China, in 2023. He is currently a master student at the College of Software, Nankai University, Tianjin, China. His research interests include failure detection and failure diagnosis.

Dan Pei received the B.E. and M.S. degrees in computer science from the Department of Computer Science and Technology, Tsinghua University in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA) in 2005. He is currently an associate professor at the Department of Computer Science and Technology, Tsinghua University. His research interests include network and service management in general.He is an IEEE senior member and an ACM senior member.