Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments

DUNEESHA FERNANDO and MARIA A. RODRIGUEZ, Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
PATRICIA ARROBA, CCS–Center for Computational Simulation, Universidad Politécnica de Madrid, Madrid, Spain
LEILA ISMAIL, Intelligent Distributed Computing and Systems Research (INDUCE) Lab, Department of Computer Science and Software Engineering, United Arab Emirates University, United Arab Emirates
RAJKUMAR BUYYA, Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

Microservice architectures are increasingly used to modularize IoT applications and deploy them in distributed and heterogeneous edge computing environments. Over time, these microservice-based IoT applications are susceptible to performance anomalies caused by resource hogging (e.g., CPU or memory), resource contention, etc., which can negatively impact their Quality of Service and violate their Service Level Agreements. Existing research on performance anomaly detection for edge computing environments focuses on model training approaches that either achieve high accuracy at the expense of a time-consuming and resource-intensive training process or prioritize training efficiency at the cost of lower accuracy. To address this gap, while considering the resource constraints and the large number of devices in modern edge platforms, we propose two clustering-based model training approaches: (1) intra-cluster parameter transfer learning (ICPTL)-based model training and (2) cluster-level model (CM) training. These approaches aim to find a tradeoff between the training efficiency of anomaly detection models and their accuracy. We compared the models trained under ICPTL and CM to models trained for specific devices (most accurate, least efficient) and a single general model trained for all devices (least accurate, most efficient). Our findings show that ICPTL's model accuracy is comparable to that of the model per device approach while requiring only 40% of the training time. In addition, CM further improves training efficiency by requiring 23% less training time and reducing the number of trained models by approximately 66% compared to ICPTL, yet achieving a higher accuracy than a single general model.

CCS Concepts: • **Security and privacy → Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies → Distributed computing methodologies; Artificial intelligence; Machine learning approaches**;

Authors' Contact Information: Duneesha Fernando (corresponding author), Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia; e-mail: dtfernando@student.unimelb.edu.au; Maria A. Rodriguez, Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia; e-mail: marodriguez@unimelb.edu.au; Patricia Arroba, CCS–Center for Computational Simulation, Universidad Politécnica de Madrid, Madrid, Spain; e-mail: parroba@die.upm.es; Leila Ismail,Intelligent Distributed Computing and Systems Research (INDUCE) Lab, Department of Computer Science and Software Engineering, United Arab Emirates University, Al Ain, United Arab Emirates; e-mail: leila@uaeu.ac.ae; Rajkumar Buyya, Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia; e-mail: rbuyya@unimelb.edu.au

ACM 1556-4703/2025/6-ART13
https://doi.org/10.1145/3725736

Additional Key Words and Phrases: Edge computing, Microservices, IoT, Performance anomaly detection, Machine learning, Deep learning

ACM Reference format:
Duneesha Fernando, Maria A. Rodriguez, Patricia Arroba, Leila Ismail, and Rajkumar Buyya. 2025. Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments. *ACM Trans. Autonom. Adapt. Syst.* 20, 2, Article 13 (June 2025), 27 pages.
https://doi.org/10.1145/3725736

1 Introduction

The emergence of the **Internet of Things (IoT)** paradigm, supporting a wide range of smart services for application domains such as healthcare, transportation, industrialization, and agriculture, has led to an exponential rise in the number of IoT devices globally. Initially, the enormous quantities of data generated by IoT devices were sent to the cloud for processing, giving an upsurge to cloud-centric IoT. However, data transmission towards centralized cloud data centers increases network congestion and latencies. To overcome this limitation, edge computing, where processing is performed closer to the IoT devices, was introduced. Edge platforms consist of a wide range of devices (e.g., smart routers and switches, edge servers, micro-data centers) that are heterogeneous in terms of computing, storage, and networking capabilities.

The computing and storage capacities of edge devices are higher than that of IoT devices but lower than that of the cloud [10]. Additionally, the computing and storage capabilities of edge devices are arranged such that the edge devices closer to the IoT devices have lower computing and storage capabilities than the edge devices closer to the cloud [19, 26]. As a result, edge and cloud resources are used dynamically to place IoT applications based on their **Quality of Service (QoS)** and resource requirements. Therefore, as shown in Figure 1, IoT applications that are latency-critical, bandwidth-hungry, and require small-scale processing are deployed in edge devices closer to IoT devices. In contrast, latency-tolerant IoT applications that require large-scale processing are deployed at edge devices closer to the cloud or even at cloud data centers [10, 20]. This article refers to such edge-cloud integrated environments as "edge computing environments."

In order to be deployed in distributed and heterogeneous edge computing environments, IoT applications are modeled as interdependent, lightweight, and granular modules. Microservice architectures that build distributed applications based on loosely coupled, lightweight, independently deployable, and scalable modules are increasingly being used for this purpose. Several studies have utilized the abstraction provided by microservice architectures to model IoT applications [2, 44] and have developed algorithms to schedule those containerized microservice-based IoT applications in edge computing environments [11, 25]. This granular placement ensures that the QoS requirements of each module can be satisfied while maximizing resource usage.

However, long-running microservice-based IoT applications, such as smart health monitoring and smart road traffic management applications, deployed in edge platforms may be prone to a degradation in performance or performance anomalies, mainly due to the environment's highly dynamic and multi-tenant nature [5, 38]. For instance, an upsurge in workload may result in an application hogging a resource (e.g., CPU or memory), or the co-location of microservices may result in resource contention, both cases potentially negatively impacting the performance of a given application. Furthermore, several types of malicious attacks, such as distributed denial-of-service attacks, also indirectly cause performance problems [12]. These performance anomalies occurring in edge computing environments adversely affect the QoS of microservice-based IoT applications and lead to violations of their Service Level Agreements. We define performance anomalies as

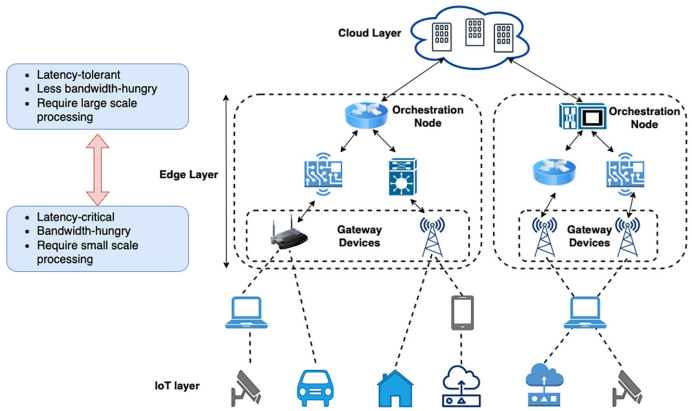


Fig. 1. Properties of applications placed in edge computing environments.

deviations from expected patterns, including outliers, in metrics that indicate the QoS and resource usage of microservices. Therefore, it is crucial to continuously monitor microservice-based IoT applications in edge computing environments with the aim of detecting and eventually mitigating such anomalies. This article focuses on anomaly detection which is the first step towards creating a self-tuning and self-repairing edge computing environment that ensures smooth operations.

Existing research studies on performance anomaly detection for edge computing environments advocate for using unsupervised multivariate time-series methods [5, 36, 38, 42], where a model learns the normal performance data distribution of microservices and identifies deviations as anomalies. Microservices deployed across devices in an edge infrastructure have heterogeneous QoS and resource requirements, resulting in heterogeneous normal data distributions. One approach to **Machine Learning (ML)**-based anomaly detection at the edge is to train a single, generic anomaly detection model (**Generic Model (GM)**) using data from all microservices across all devices [28, 31, 38]. Training a single model is efficient in terms of training time and resource usage and reduces the overhead associated with model management. However, it requires aggregating local (edge device) time series data in a centralized location, usually a cloud data center, thus considerably increasing network utilization and potentially causing network congestion [42]. Importantly, a GM usually yields lower accuracy due to the difficulty of generalizing across the diverse normal data distributions. An alternative approach involves training a "model per edge device" (MPD) using data collected from the microservices deployed in that specific edge device [5, 15, 32, 42]. Since microservices within a single edge device usually have similar QoS and resource requirements [9, 25] and hence have similar normal data distributions, this approach results in models with higher accuracy, as the model is expected to specialize well across similar normal data distributions. Training as many models as edge devices consumes a significant amount of resources and results in longer aggregate training times, as well as increased model management overhead. While a model per device allows for each edge device to train its associated anomaly detection model, performing training, which is a resource-intensive process, might be challenging for most edge devices due to their resource-constrained nature.

The aforementioned approaches (which we identify as baselines) represent extremes in terms of accuracy and efficiency; a GM has low accuracy but can be efficiently trained, while the model per device approach (MPD) yields high accuracy, but the training process is time-consuming and resource-hungry. Considering the resource constraints and large number of devices present in modern edge platforms, this research aims to bridge the gap between GM and MPD by finding

a tradeoff between the training efficiency of anomaly detection models and their accuracy. We propose to achieve this by conducting model training within clusters of edge devices with similar normal data distributions. To that end, we exploit the QoS-aware placement of microservice-based IoT applications [9, 11, 25] in edge computing environments to propose a similarity metric capable of clustering edge devices with similar normal data distributions. Furthermore, we introduce two clustering-based training approaches, namely, (1) **Intra-Cluster Parameter Transfer Learning (ICPTL)**-based model training and (2) **cluster-level model (CM)** training. ICPTL aims to match MPD's accuracy with fewer training cycles, while CM aims to further improve training efficiency by reducing the number of models. To the best of our knowledge, this is the first work to propose anomaly detection model training approaches tailored for the specific characteristics and constraints of edge environments with the aim of increasing training efficiency while achieving high model accuracy.

We conducted a comprehensive and reproducible evaluation of the proposed clustering-based training approaches using a publicly available and widely used performance anomaly dataset called the "**Server Machine Dataset (SMD)**" [39], which consists of devices with heterogeneous normal data distributions. Initially, we compared four common unsupervised multivariate time-series anomaly detection techniques for accuracy and efficiency (in terms of resource utilization and detection time) against the SMD dataset to determine the most suitable algorithm for the rest of our evaluations. Upon evaluating the proposed clustering-based training approaches using the identified algorithm, we demonstrate that the clustering approaches achieve a balance between accuracy and efficiency, as expected. To further validate the results, we conducted a small-scale experiment using data collected from a set of microservices with heterogeneous QoS and resource requirements deployed in an emulated edge computing environment [8].

The rest of the article is organized as follows: Section 2 reviews the existing related works. Section 3 presents the clustering-based training approaches. Section 4 evaluates the performance anomaly detection algorithms and the proposed training approaches. Section 5 discusses how the proposed training approaches handle the dynamic nature of edge computing environments. Section 6 discusses the validity threats of our study. Section 7 concludes the article and draws future research directions.

2 Related Work

In this section, we provide an overview of performance anomaly detection studies conducted in edge computing environments in terms of the algorithms, evaluation metrics, and training methods they have used. We also explore several related works on ML model training approaches employed in edge computing environments.

2.1 Performance Anomaly Detection in Edge Computing Environments

Several works have proposed monitoring solutions for edge computing environments, some of which have incorporated alerting as a functionality [30, 40]. However, none of these monitoring solutions perform any advanced form of anomaly detection other than threshold-based alerting. Threshold-based alerting requires system administrators to manually define thresholds for each metric per device, which is not scalable given the large number of edge devices and the multitude of performance metrics being monitored. Moreover, these static thresholds become obsolete faster in dynamic edge computing environments. In contrast, multivariate time series anomaly detection algorithms can accurately detect more types of anomalies since they consider inter-metric correlations.

Building on the limitations of threshold-based alerting, recent research on performance anomaly detection for edge computing environments suggests using unsupervised multivariate time-series

techniques for anomaly detection [5, 36, 38, 42]. Becker et al. evaluated ARIMA, BIRCH, LSTM, and EMA algorithms, all of which are unsupervised anomaly detection approaches [5]. Similarly, Skaperas et al. evaluated two change point detection approaches: Bayesian and cumulative sum, both of which can be identified as unsupervised approaches [36]. Tuli et al. explored several unsupervised approaches, including OCSVM, **Isolation Forest (IF)**, DILOF, and USAD [42]. Although Soualhia et al. evaluated a group of supervised approaches, they emphasized the infeasibility of such methods due to the need for labeled normal and anomaly data for training [38]. As anomaly data may not be available during the initial execution of applications and labeling can be costly, they suggest using unsupervised anomaly detection approaches. These algorithms are trained on unlabeled normal data to learn the normal data distribution and identify deviations from it as anomalies. Therefore, in our work, we utilize unsupervised multivariate time-series anomaly detection techniques.

Although research on performance anomaly detection in edge computing environments is still emerging, AI-based performance anomaly detection in cloud and general microservices environments is well-established [3, 4, 16, 23, 33, 34]. According to Audibert et al. [4], existing anomaly detection algorithms can be broadly classified into three categories: (1) conventional methods, (2) ML-based methods, and (3) **Deep Neural Network (DNN)**-based methods. Studies on AI-based performance anomaly detection in cloud and microservices environments typically evaluate proposed algorithms against representative techniques from all three categories. Similarly, research on performance anomaly detection in edge environments often adapts cloud anomaly detection approaches [5, 36, 38, 42], following the trend of assessing algorithms across these three groups. In line with this practice, Section 4.2 evaluates common unsupervised multivariate time-series anomaly detection techniques using the SMD dataset to identify the most suitable algorithm for the rest of our evaluations. Four representative algorithms from each of the three categories are selected for this purpose.

Cloud/microservice performance anomaly detection research typically does not account for the constraints of edge computing environments and, therefore, only focuses on evaluating the accuracy of anomaly detection models. All those studies report only accuracy-related metrics such as precision, recall and F1-score [4, 37]. However, due to the resource-constrained nature of edge devices, edge performance anomaly detection studies evaluate both accuracy and efficiency during inference (in terms of resource utilization and detection time) when identifying a suitable performance anomaly detection algorithm to be deployed at the edge devices [5, 36, 42]. Consequently, in Section 4.2, we evaluate the accuracy as well as resource utilization efficiency of four representative algorithms from each category of Audibert's taxonomy to determine the most suitable algorithm for the remainder of our evaluations.

In addition to evaluating the accuracy and efficiency of the algorithms during detection, considering the fact that these models are trained on resource-constrained edge devices, some of these studies have also assessed the time taken and resource utilization during the training process [38, 42]. Although Becker et al. do not evaluate the above aspect, they also propose training a model at the edge device [5]. As discussed in Section 1, while this model per device (MPD) approach yields high accuracy, the presence of a large number of edge devices leads to a high aggregate training time and an increased total consumption of resources during the training process. On the other hand, Soualhia et al. propose training a generic anomaly detection model (GM) using data from all devices [38]. While this approach can be efficiently trained, it results in low accuracy. Notably, the GM approach used by Soualhia et al. aligns with the training strategies typically employed in AI-based performance anomaly detection studies in cloud environments and general microservices, which often overlook the impact of resource constraints during the training phase.

A comparison between prior works on performance anomaly detection in edge computing environments [5, 36, 38, 42] along with our proposed work is shown in Table 1. Under the "Anomaly

Table 1. A Summary of Performance Anomaly Detection Studies in Edge Computing Environments

Work	Anomaly Detection Algorithms	Anomaly Detection Evaluation Aspects		Model Training Approach	Main Objective of Work
		Accuracy	Efficiency		
[38]	ARIMA (Conventional), MC (ML), SVM (ML), RF (ML), NN (ML), BN (ML), TAN (ML)	✓	×	GM	A framework to perform anomaly detection in edge environments.
[5]	ARIMA (Conventional), BIRCH (ML), LSTM (DNN), EMA (Conventional)	×	✓CPU utilization, detection delay	Model per device (MPD)	AIOPS framework for edge environments.
[42]	OCSVM (ML), IF (ML), DILOF (DL), ONLAD (ML), CAE-M (DL), USAD (DL), MAD-GAN (DL), SlimGAN (DL)	✓	✓Memory consumption, inference time	Model per device (MPD)	Propose memory-efficient anomaly detection approaches for edge devices.
[36]	CPD approaches: Bayesian (Conventional), CUSUM (Conventional)	✓	✓Resource utilization (CPU and memory consumption), detection delay	Model per device (MPD)	Framework that facilitates evaluation of CPD algorithms.
Our work	VAR (Conventional), IF (ML), AE (DNN), LSTM-AE (DNN)	✓	✓Detection time, resource requirements (CPU and memory)	Similarity-based clustering approaches	Propose efficient model training approaches for performance anomaly detection in edge environments.

AE, AutoEncoder; BIRCH, balanced iterative reducing and clustering using hierarchies; BN, Bayesian network; CAE-M, convolutional autoencoder-mahalanobis; CUSUM, cumulative sum control chart; DILOF, deep isolation forest; DNN, deep neural network; EMA, exponential moving average; IF, isolation forest; LSTM, long short-term memory; LSTM-AE, LSTM-AutoEncoder; MAD-GAN, multi-scale anomaly detection with GANs; MC, Markov chain; NN, neural network; OCSVM, one-class support vector machine; ONLAD, online anomaly detection via adaptive learning; RF, random forest; SlimGAN, slim generative adversarial network; SVM, support vector machine; TAN, tree-augmented naive bayes; USAD, unsupervised anomaly detection; VAR, vector autoregression.

Detection Evaluation Aspects" column of the table, checkmarks and crosses represent whether that particular study has considered accuracy and/or efficiency when conducting evaluations. Notably, none of these studies have tried to reach a tradeoff between the efficiency and accuracy of performance anomaly detection model training, which presents a gap in current research that we aim to explore further in this article.

2.2 ML Model Training Approaches in Edge Computing Environments

Since none of the edge performance anomaly detection studies have developed approaches that strike a balance between the efficiency and accuracy of performance anomaly detection model training, we reviewed the literature on ML model training for Artificial Intelligence of Things applications as well [15, 28, 31, 32]. Our goal was to identify the model training approaches used in these works. John et al. [15] and Raj et al. [32] discussed the generic and specialized model training approaches that we could also identify from edge performance anomaly detection literature, while Peltonen and Dias [28] and Psaromanolakis et al. [31] focused on training a GM. As mentioned before, since these two training approaches achieve either high accuracy or high efficiency, we use them as the baselines for comparing our proposed clustering-based training approaches.

Additionally, a few works employ novel ML approaches, such as federated learning [35] and distributed ML [45], for model training in edge computing environments. Schneible and Lu utilized federated learning in the context of supervised anomaly detection [35]. Since different edge devices encounter different anomaly data, they distribute the required updates via a centralized model to train the anomaly detection models of other edge devices with newly seen anomaly data. However, adopting this approach for unsupervised performance anomaly detection would result in each

model learning the normal data distributions of all microservices deployed across all edge devices, ultimately leading to lower model accuracy.

On the other hand, Yang et al. [45] used distributed ML to detect network anomalies at edge devices by analyzing network packets. Their approach involves centrally training an ML model using multiple interconnected computing resources and then deploying it to infer anomalies at the edge device level. Although this approach makes the training process faster by parallelizing and utilizing more resources, it results in a GM trained on data aggregated from all edge devices. This can lead to reduced accuracy when applied to performance anomaly detection model training, as the GM is not capable of generalizing well across the heterogeneous normal data distributions of all microservices.

Thus, adapting ML approaches such as federated learning and distributed ML for training performance anomaly detection models results in the creation of a GM where every model learns the normal data distribution of microservices deployed in other devices as well. Although this approach results in high accuracy during situations where all the collected data belong to the same **Independent and Identically Distributed (IID)** distribution, in our case, since the data collected from different microservices belong to non-IIDs, this approach could drift the model away from expected behavior, resulting in low accuracy. Therefore, it is clear that simply adapting existing ML approaches to perform anomaly detection model training in edge computing environments does not allow us to strike a balance between efficiency and accuracy. Therefore, in this article, we propose a new approach to training anomaly detection models tailored to the specific characteristics and constraints of edge environments with the aim of increasing training efficiency while achieving high model accuracy.

3 Clustering-Based Training Approaches

This study explores a hierarchical edge environment with multiple heterogeneous devices [24]. If there are N edge devices, the set of edge devices can be defined as E, where e_i represents the ith edge device.

$$E = \{e_1, e_2, ..., e_N\}. \tag{1}$$

Each device is equipped with a monitoring agent that gathers M performance and resource consumption metrics from each microservice deployed on that device. Consistent with prior research on performance anomaly detection at the edge [5, 31, 38, 42], this study also presumes that an unsupervised multivariate anomaly detection model, which analyzes M performance metrics, is deployed at each edge device. Normal performance data (i.e., data collected under non-anomalous conditions) is used to train unsupervised models, and the set of training data collected from device e_i for T timesteps can be defined as matrix D_{e_i},

$$D_{e_i} = [d_{m,t}]_{M \times T}. \tag{2}$$

For each edge device e_i, the Model per Device (MPD) baseline approach trains a model by taking D_{e_i} as input. The approach optimizes the model parameters w by minimizing the loss function L. The resulting optimal model parameters are denoted as $w_{e_i}^{mpd}$, as shown in Equation (3). Since a model is trained per edge device and microservices within a specific edge device are expected to have similar normal metric distributions (due to the QoS and resource requirement-aware placement of microservices) [9, 25], the accuracy of these models is expected to be high. However, because a large number of models need to be trained, this approach is not very efficient.

$$w_{e_i}^{mpd} = argmin_w L(w; D_{e_i}). \tag{3}$$

On the other hand, the GM baseline approach takes the data belonging to all edge devices, denoted as D_E, as input and trains a single model by minimizing the loss function L to optimize the model parameters w, as shown in Equation (4). The resulting optimal model parameters are denoted as w^{gm}. A single GM compromises accuracy in favor of training efficiency; a single model is unlikely to generalize well across the heterogeneous normal data distributions of all microservices in an edge environment, yet can be trained with fewer resources when compared to training multiple specialized models.

$$w^{gm} = argmin_w L(w; D_E). \tag{4}$$

Since the above-explained baseline model training approaches are extremes in terms of model accuracy and training efficiency, it is important to reach a tradeoff between accuracy and efficiency during the training process. This aim can be achieved by training a few models on groups of similar data, i.e., the heterogeneity of data used to train a single model should be minimal. To accomplish this, we use the concept of clustering to group edge devices with similar normal data distributions and then train the models within these clusters. Towards that, we exploit the QoS-aware placement of microservice-based IoT applications [9, 11, 25] to cluster edge devices with similar normal data distributions. Under a QoS-aware placement strategy, we assume that microservices with similar normal data distributions are placed in the same edge device or edge devices with similar computing, storage, and networking capabilities. Clustering is performed based on edge devices since the smallest unit of model deployment is the edge device level. Towards that, in Section 3.1, we introduce a similarity metric capable of clustering edge devices with similar normal data distributions, followed by Sections 3.2 and 3.3, where we propose the two clustering-based training approaches, ICPTL-based model training and CM training, respectively. Finally, in Section 3.4, we discuss the positioning of the proposed approaches and the workflow for initial model deployment.

3.1 Similarity-Based Clustering

In this context, we assume that microservices are placed based on their QoS and resource requirements. Under such a QoS-aware placement strategy, we assume that microservices with similar normal data distributions are placed in the same edge device or edge devices with similar computing, storage, and networking capabilities. As a result, microservices deployed at different hierarchical levels show a variance in metrics. This variance is more pronounced in some metrics (e.g., CPU and memory consumption metrics) than in others (e.g., error counts, disk read/write throughput). For instance, microservices situated closer to the cloud generally have higher values for CPU and memory consumption metrics and lower values for latency than those closer to the IoT layer. Hence, we identify a subset of H such metrics, representative of the QoS and resource usage of microservices, suitable to perform device clustering from the M metrics.

The first step towards cluster formation is calculating the similarity between each pair of devices, as proposed in Algorithm 1. The proposed algorithm takes as input normal datasets collected from all edge devices $D_E = \{D_{e_1}, D_{e_2}, ., D_{e_i}, ., D_{e_N}\}$ and the list of H representative metrics. It first extracts the H representative metrics from each dataset D_{e_i}, resulting in $D^*_{e_i} = [d_{h,t}]_{H \times T}$. Next, each metric of D^*_E is standardized using min-max normalization as shown in step 5 of Algorithm 1. Subsequently, the algorithm generates a multivariate probability distribution P_{e_i} composed of the probability distributions for each of the H representative metrics, for each device e_i.

Once the multivariate probability distributions for all devices are available, the similarity distance $sim\_dist(e_i, e_j)$ between each pair of devices e_i and e_j is calculated by obtaining the L2-norm of the $JS\_distance$ between each representative metric of the probability distributions P_{e_i} and P_{e_j} corresponding to those devices. The equation corresponding to this step is shown in

Algorithm 1: Similarity Graph Formation Algorithm

Input : Normal datasets collected from all edge devices $D_E = \{D_{e_1}, D_{e_2}, .., D_{e_i}, .., D_{e_N}\}$
Input : List of H representative metrics
Output : Similarity graph SG
Compose $D^*_E = \{D^*_{e_1}, D^*_{e_2}, .., D^*_{e_i}, .., D^*_{e_N}\}$ (where $D^*_{e_i} = [d_{h,t}]_{H \times T}$) by extracting the H representative metrics
Standardize D^*_E s.t. $D^*_E[h,:] \leftarrow \frac{D^*_E[h,:]}{max(D^*_E[h,:]) - min(D^*_E[h,:])}, \quad \forall h; 0 < h \leq H$
Obtain $P_E = \{P_{e_1}, P_{e_2}, .., P_{e_i}, .., P_{e_N}\}$ using D^*_E, where $P_{e_i} = [p_{e_{i,1}}, p_{e_{i,2}}, .., p_{e_{i,H}}]$ is the multivariate probability distribution for each edge device e_i
Initialize similarity graph $SG = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = [e_i, \ \forall i; 0 < i \leq N]$ and $\mathbb{E} = \emptyset$
for each device $e_i; 0 < i \leq N$ **do**
  for each device $e_j; i < j \leq N$ **do**
    $sim\_dist(e_i, e_j) = \sqrt{\sum_{h=1}^{H} JS\_distance(p_{e_{i,h}}, p_{e_{j,h}})^2}$
    $\mathbb{E} \leftarrow \mathbb{E} \bigcup \{(e_i, e_j, sim\_dist)\}$
  end for
end for
Return : SG

Equation (5). The $JS\_distance$ between the probability distributions $p_{e_{i,h}}$ and $p_{e_{j,h}}$ corresponding to a single metric h, is obtained by calculating the square root of their $JS\_divergence$, as shown in Equation (6). The $JS\_divergence$ between any two probability distributions, $\mathbb{P}$ and $\mathbb{Q}$, is the weighted sum of the forward and backward **Kullback–Leibler (KL)** divergence between those probability distributions (refer to Equation (7)). The equation for calculating the KL divergence $KL(\mathbb{P} \parallel \mathbb{Q})$ between any two probability distributions, $\mathbb{P}$ and $\mathbb{Q}$, is shown in Equation (8) [18].

$$sim\_dist(e_i, e_j) = \sqrt{\sum_{h=1}^{H} JS\_distance(p_{e_{i,h}}, p_{e_{j,h}})^2}, \tag{5}$$

$$JS\_distance(p_{e_{i,h}}, p_{e_{j,h}}) = \sqrt{JS\_divergence(p_{e_{i,h}} || p_{e_{j,h}})}, \tag{6}$$

$$JS\_divergence(\mathbb{P}||\mathbb{Q}) = \frac{1}{2} KL\left(\mathbb{P}||\frac{\mathbb{P}+\mathbb{Q}}{2}\right) + \frac{1}{2} KL\left(\mathbb{Q}||\frac{\mathbb{Q}+\mathbb{P}}{2}\right), \tag{7}$$

$$KL(\mathbb{P} \parallel \mathbb{Q}) = \sum_i \mathbb{P}(i) \log\left(\frac{\mathbb{P}(i)}{\mathbb{Q}(i)}\right). \tag{8}$$

The output of Algorithm 1 is a complete graph, where a vertex represents an edge device e_i, and the weight of the edge between each pair of vertices e_i and e_j is the similarity distance $sim\_dist(e_i, e_j)$ between the corresponding edge devices. This graph is hereafter referred to as the similarity graph SG.

The similarity graph SG produced by Algorithm 1 is then used to generate clusters of devices with similar normal data distributions, as depicted in Algorithm 2. Furthermore, the number of clusters K is a hyperparameter and is required as input for this algorithm. In the first step, the $cluster\_map$, which stores the mapping from devices to clusters, is initialized such that each edge device e_k is assigned to a unique cluster c_k. Therefore, initially, there are as many clusters as edge devices. Next, the algorithm identifies vertices corresponding to the shortest edge of the similarity graph SG, i.e., the edge devices e_i, e_j with the highest similarity, and merges clusters containing those edge devices as depicted in steps 6 to 11 of Algorithm 2. These steps are repeated until there

Algorithm 2: Similarity-Based Device Clustering Algorithm

Input : Similarity graph $SG = (\mathbb{V}, \mathbb{E})$ (output of Algorithm 1)
Input : No.of clusters K
Output : Cluster to device mapping *cluster_map*
Initialize $cluster\_map = \{c_k \rightarrow [e_k], \quad \forall k; 0 < k \leq N\}$
while $length(cluster\_map) > K$ **do**
  Find $e_{i,j}^{min} = (e_{i^*}, e_{j^*}, w_{i^*,j^*})$ s.t. $(e_{i^*}, e_{j^*}, w_{i^*,j^*}) = \underset{(e_i, e_j, w_{i,j}) \in \mathbb{E}}{\text{argmin}} \; w_{i,j}$
  $c_i = cluster\_map^{-1}(e_i)$ // Look up the cluster to which the device e_i belongs to
  $c_j = cluster\_map^{-1}(e_j)$ // Look up the cluster to which the device e_j belongs to
  $c_i \leftarrow c_i \bigcup c_j$
  $cluster\_map \leftarrow cluster\_map \setminus c_j$
  $\mathbb{E} \leftarrow \mathbb{E} \setminus e_{i,j}^{min}$
end while
Return : *cluster_map*

are K clusters in the *cluster_map*. This algorithm, inspired by Kruskal's algorithm [17], ensures that clusters are created by grouping together devices corresponding to shorter edges. It returns the *cluster_map* with K clusters as the output. Through the proposed similarity-based clustering approach, we expect that edge devices having microservices with similar normal data distributions fall into the same cluster.

3.2 ICPTL-Based Model Training

The aim of the ICPTL-based model training approach is to achieve a similar level of accuracy as the Model per Device (MPD) approach but with fewer training cycles. In the MPD approach, each anomaly detection model needs to learn the normal data distribution from its respective edge device. However, the normal data distributions among edge devices within the same cluster only differ slightly. Therefore, in the ICPTL approach, instead of training the model for each edge device from scratch, we utilize the concept of parameter transfer learning, which involves transferring the weights learned by one model (the source model) and its hyperparameters to initiate the training of a model for another edge device (the target model) [27]. For instance, if the source device is e_i and the target device is e_j, the ICPTL approach takes dataset D_{e_j} and model parameters of source model w_{e_i} as inputs and trains a target model by minimizing the loss function L to optimize the model parameters w. The resulting optimal model parameters are denoted as $w_{e_j}^{icptl}$, as shown in Equation (9). This allows the target model to have an advantage at the beginning of its training, as the source model has already learned the lower-level features required by the target model, thus reaching a satisfactory accuracy within a few epochs.

$$w_{e_j}^{icptl} = argmin_w L(w; D_{e_j}, w_{e_i}) \tag{9}$$

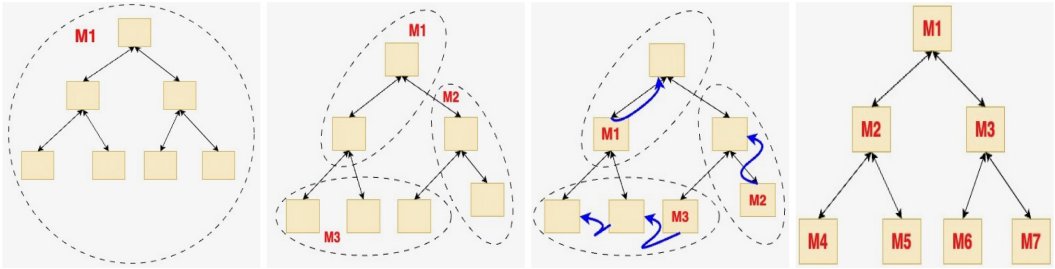
Algorithm 3 comprises the steps of the ICPTL-based model training approach. It consumes the similarity graph *SG*, which is the output of Algorithm 1 and the cluster to device mapping *cluster_map* which is the output of Algorithm 2 as inputs. For a given cluster c_k, the *source_set*, which stores the source devices with already trained models and the *target_set*, which stores the devices where the models are yet to be trained are first initialized. Next, the subgraph SG_k, which consists of edge devices in c_k as vertices, is extracted from the similarity graph *SG*. From SG_k, one of the vertices corresponding to the shortest edge, i.e., one of the edge devices e_i from the pair exhibiting the highest similarity, is arbitrarily selected as the *source* device. Next, a model is trained

Algorithm 3: ICPTL-Based Model Training Algorithm

1: *Input* : Similarity graph $SG = (\mathbb{V}, \mathbb{E})$ (output of Algorithm 1)
2: *Input* : Cluster to device mapping *cluster_map* (output of Algorithm 2)
3: *Output* : Models for all devices
4: **for** each cluster $c_k; 0 < k \leq K$ **do**
5: Initialize $source\_set = \emptyset$
6: Initialize $target\_set = cluster\_map[c_k]$
7: Extract $SG_k = (\mathbb{V}_k, \mathbb{E}_k)$ where $\mathbb{V}_k = cluster\_map[c_k]$ and $\mathbb{E}_k = \{(e_i, e_j, w_{i,j}) \in \mathbb{E}; e_i, e_j \in \mathbb{V}_k\}$
8: Find $e^{min}_{i,j} = (e_{i^*}, e_{j^*}, w_{i^*,j^*})$ s.t. $(e_{i^*}, e_{j^*}, w_{i^*,j^*}) = \underset{(e_i, e_j, w_{i,j}) \in \mathbb{E}_k}{\operatorname{argmin}} w_{i,j}$
9: $source \leftarrow e_i$
10: $target \leftarrow e_j$
11: $w_{source} = \operatorname{argmin}_w L(w; D_{source})$
12: $source\_set \leftarrow source\_set \bigcup source$
13: $target\_set \leftarrow target\_set \setminus source$
14: **while** $target\_set \neq \emptyset$ **do**
15: $w_{target} = \operatorname{argmin}_w L(w; D_{target}, w_{source})$
16: $source\_set \leftarrow source\_set \bigcup target$
17: $target\_set \leftarrow target\_set \setminus target$
18: $\mathbb{E}_k \leftarrow \mathbb{E}_k \setminus e^{min}_{i,j}$
19: Find $e^{min}_{i,j} = (e_{i^*}, e_{j^*}, w_{i^*,j^*})$ s.t. $(e_{i^*}, e_{j^*}, w_{i^*,j^*}) = \underset{(e_i, e_j, w_{i,j}) \in \mathbb{E}_k}{\operatorname{argmin}} w_{i,j}$ and $e_i \in source\_set$
20: $source \leftarrow e_i$
21: $target \leftarrow e_j$
22: **end while**
23: **end for**
24: *Return* : Models for all devices

for that device by following the Model per Device (MPD) approach as shown in step 11 of the algorithm. After training, the *source* device e_i is moved to the *source_set*. Thereafter, the model for the remaining device e_j (i.e., the *target* device) is trained using parameter transfer learning as shown in Equation (9), and it also corresponds to step 15 of the algorithm. After this step, the *target* device e_j is also moved to the *source_set*. Subsequently, as shown in step 19, the algorithm identifies the next shortest edge from the remaining edges, such that the edge device corresponding to one of its vertices belongs to the *source_set*. Next, by utilizing the model for that edge device as the source model, the model for the edge device corresponding to the other vertex is trained using parameter transfer learning. These steps are repeated until the *target_set* becomes empty. At the end of the algorithm, a trained model will be available for all edge devices within c_k. Moreover, ICPTL-based model training can be performed simultaneously across clusters.

Figure 2(c) visually represents the ICPTL approach. It essentially identifies a **Minimum Spanning Tree (MST)** of the cluster as the sequence to perform parameter transfer learning among the edge devices. In the identified sequence, a model is traditionally trained only for the device at the starting point of the sequence, while the other devices in the sequence receive the model from the nearest neighbor device and train a few epochs until they reach satisfactory accuracy. While this approach entails training a model for each edge device, it requires fewer epochs during training compared to the Model per Device (MPD) approach, thus resulting in reduced training resource requirements and training time.



(a) Generic model (b) Cluster-level model training (c) Intra-cluster parameter transfer learning-based model training (d) Model per device

Fig. 2. Visual representation of the two clustering-based training approaches and the two baseline approaches.

3.3 CM Training

Through the CM training approach, we aim to improve training efficiency further by reducing the number of models. To that end, we propose to train a model per cluster using normal data collected from all edge devices in the cluster. The CM approach takes the data belonging to all edge devices in cluster c_k, denoted as D_k, as input and trains a model for that cluster by minimizing the loss function L to optimize the model parameters w as shown in Equation (10). The resulting optimal model parameters are denoted as w_k^{cm}. The CM approach is visually represented in Figure 2(b).

$$w_k^{cm} = argmin_w L(w; D_k) \tag{10}$$

Since similarity-based clustering ensures that devices having similar normal data distributions fall into one cluster, the model trained for each cluster can effectively generalize across all the edge devices in that cluster. This cluster-specific approach is expected to be more accurate than the GM approach.

Since the number of clusters is always less than or equal to the number of edge devices, this means that fewer models need to be trained when using the CM training approach compared to the Model per Device (MPD) approach. This results in reduced training time and resource requirements. Additionally, it is easier to manage a few models per cluster than a relatively large number of models per edge device. As a result, the CM approach is expected to be more efficient than the MPD approach.

3.4 Positioning of the Proposed Approaches and Initial Model Deployment Workflow

Out of the two proposed approaches, the CM training approach is designed to provide more efficiency (in terms of training time, training resource requirements, number of epochs, and model management overhead) compared to ICPTL-based model training approach, while ICPTL is designed to provide more accuracy than CM. Figure 3 depicts the positioning of the two proposed clustering-based training approaches amidst the two baseline model training approaches in terms of accuracy and efficiency.

Figure 4 illustrates the workflow for training models during their initial deployment in edge computing environments. Following the initial cluster generation steps (outlined in Algorithms 1 and 2), the service provider can choose between the two proposed clustering-based training approaches. For scenarios prioritizing efficiency—such as reduced training time, lower training resource requirements, fewer epochs, and minimal model management overhead—the CM approach is recommended. Conversely, if higher accuracy is the primary objective, the ICPTL approach is more suitable.

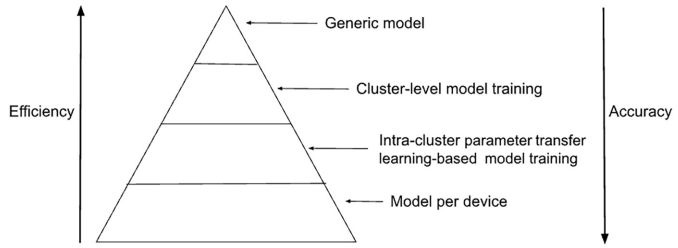


Fig. 3. Positioning of the two proposed clustering-based training approaches.

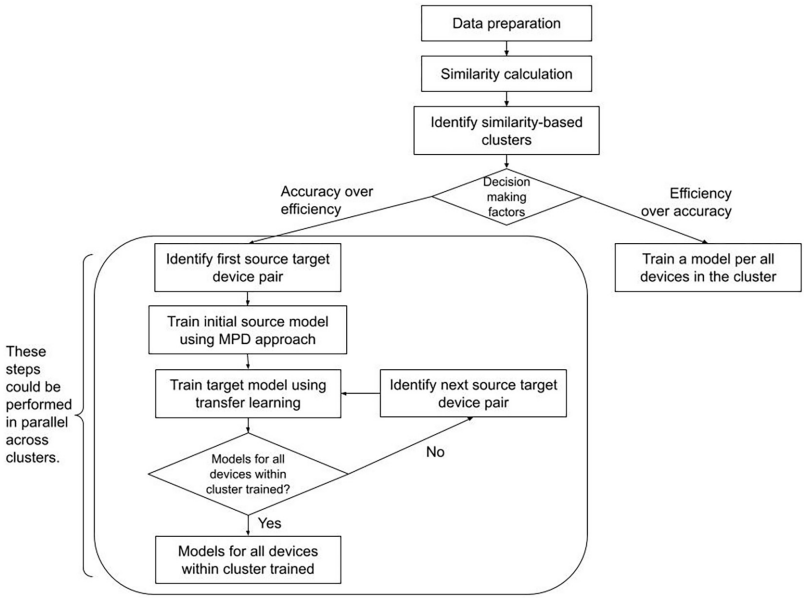


Fig. 4. Initial model deployment workflow.

4 Performance Evaluation

In this section, we discuss the evaluation results of the proposed clustering-based training approaches. First, we explain the details of the experimental setup used for evaluation. Following that, we compare four common unsupervised multivariate time-series anomaly detection techniques to identify the most suitable algorithm for the rest of our evaluations. Finally, we discuss the results of evaluating the two proposed clustering-based training approaches using the algorithm identified from the previous step against the baseline approaches derived from the state of the art.

4.1 Experimental Setup

In order to conduct a comprehensive and reproducible evaluation of the proposed approaches, we utilized a public performance anomaly dataset called the "Server Machine Dataset," which is widely used in the anomaly detection literature [3, 4, 39]. The SMD contains both normal (i.e., data collected under non-anomalous conditions) and anomalous performance data (i.e., data collected under anomalous conditions) from 28 devices across 38 metrics. Following the suggestion of Li et al. [21], we focused on 14 devices without a concept drift between training (normal performance) data and testing (anomalous performance) data for our evaluation. The SMD dataset has also

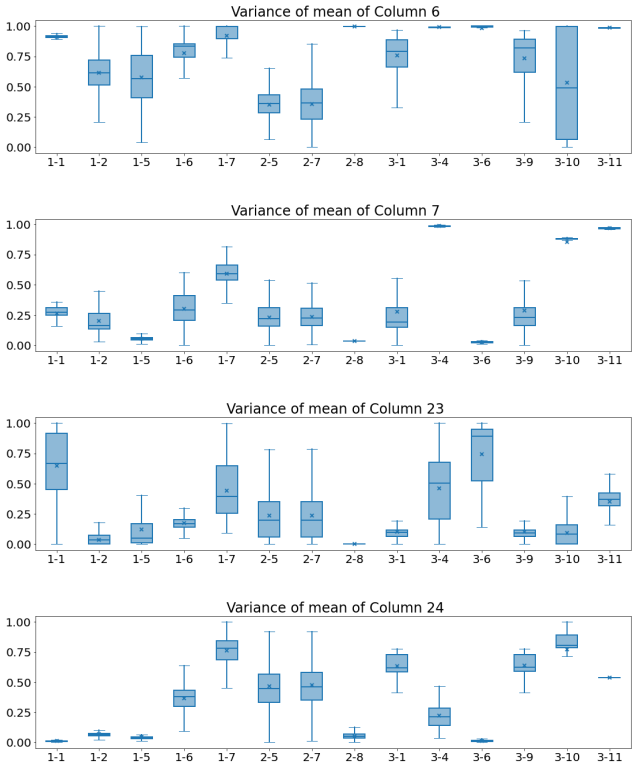


Fig. 5. Distribution of normal data across SMD devices.

been utilized in assessing research on edge computing environments [41, 42], as the data from these devices exhibit heterogeneous normal data distributions akin to those in edge computing environments. This is apparent from the significant variance in mean values of certain selected metrics across the majority of devices, as shown in Figure 5. In our evaluation, we assume that each SMD machine is an edge device with a single microservice deployed in it and that the data reported for each machine (edge device) belongs to its respective microservice.

In order to further verify the results obtained on SMD, we also collected performance anomaly data from a set of microservices with heterogeneous QoS and resource requirements deployed in an emulated edge computing environment. By using the iAnomaly toolkit [8], we emulated 10 heterogeneous edge devices using 10 virtual machines with different resource capacities (2vCPU/8GB, 4vCPU/16GB, 8vCPU/32GB) on Melbourne Research Cloud[1] and emulated the network (by following the communication link parameters used by Pallewatta et al. [24] in their experiments) to represent the heterogeneity of network bandwidth between the edge devices. The iAnomaly toolkit is a full-system emulator with iContinuum [1] (a well-tested edge emulator for discrepancies compared to real-world scenarios) at its core. In addition to that, iAnomaly has extensively verified that both the normal and anomalous data that it generates closely resemble the data generated from real edge environments. We deployed microservices with heterogeneous resource requirements representing a diverse set of IoT applications (from lightweight sensor data processing to heavy applications like camera face detection/recognition) on the emulated edge infrastructure by

[1] https://dashboard.cloud.unimelb.edu.au/.

following the placement algorithm proposed by Pallewatta et al. [25]. Using Pixie,[2] which is a lightweight monitoring tool suitable for edge monitoring, we collected both normal performance data as well as anomalous performance data under a diverse set of anomalies, such as resource saturation, service failures, network saturation, etc., at a granularity of 10 seconds. Normal workloads were generated using Jmeter[3] while anomalies were injected using Chaos Mesh.[4] Both normal and anomalous performance data are available for 12 metrics, covering both system and application metrics.

Experiments on both datasets, including hyperparameter tuning, were conducted on the Spartan[5] HPC cluster. More details on training, including the hyperparameters selected for training, are explained in Sections 4.2, 4.3, and 4.4.

4.2 Evaluating Performance Anomaly Detection Algorithms against the SMD Dataset

Prior to evaluating the proposed clustering-based training approaches using the SMD dataset, in this section, we compare four common unsupervised multivariate time-series anomaly detection techniques to identify the most suitable algorithm for the rest of our evaluations in terms of accuracy and efficiency during inference.

As mentioned in Section 2.1, while research on performance anomaly detection in edge computing environments is still emerging, AI-based performance anomaly detection techniques are well-established for cloud and microservices environments [3, 4, 16, 23, 33, 34]. Audibert et al. [4] classify anomaly detection algorithms into three categories: (1) conventional methods, (2) ML-based methods, and (3) DNN-based methods. Research on performance anomaly detection in edge environments often adapts cloud-based techniques [5, 36, 38, 42], benchmarking them across these categories. Therefore, in this section, we select four representative performance anomaly detection algorithms from these categories to identify the most suitable algorithm for further evaluation.

We selected the **Vector AutoRegression (VAR)** algorithm from conventional methods because it extends ARIMA to multivariate data, making it highly effective for time series modelling. ARIMA has consistently outperformed other techniques in multiple studies, including those by Soualhia et al. [38], Becker et al. [5], and Pitakrat et al. [29]. Furthermore, as highlighted by Audibert et al. [4], VAR is widely recognized as one of the most commonly used techniques for multivariate time series anomaly detection. IF is a widely used ML algorithm frequently employed in anomaly detection studies [3, 16, 33, 39, 42]. It has also been identified as the top-performing technique in studies by Kardani et al. [16] and Ramamoorthi [33]. These factors led us to select IF to represent the "ML-based methods" category. We selected two algorithms representing DNN-based methods: **AutoEncoder (AE)** and **LSTM-AutoEncoder (LSTM-AE)**. AE, as well as approaches that have extended the AE concept [3, 7, 42, 46], are a class of popular DNN-based methods. AE has also been recognized as the top-performing method in studies conducted by Audibert et al. [3] and Borghesi et al. [7]. In addition to AE, we also considered LSTM-AE, since LSTM layers are capable of capturing time-series dependencies better [4, 5, 23, 39]. Such approaches where AEs are enhanced with RNN layers have been identified as top-performing techniques in several studies conducted by Islam et al. [13, 14].

Therefore, we select the following four representative performance anomaly detection algorithms from each category.

[2] https://docs.px.dev/.
[3] https://jmeter.apache.org/.
[4] https://chaos-mesh.org/.
[5] https://dashboard.hpc.unimelb.edu.au/.

Table 2. Hyperparameters Tuned for Each Anomaly Detection Algorithm

ML Model	Tuned Hyperparameters
AE	num_layers, window_size, hidden_size, batch_size, learning_rate
IF	n_estimators, max_features, max_samples, k (for the k-point moving average)
LSTM-AE	num_layers, window_size, batch_size, learning_rate

Table 3. Evaluation Results (F1-Score, AUC) of Anomaly Detection Algorithms

	AE	IF	LSTM-AE	VAR
AUC	0.89	0.84	0.86	0.74
F1-score	0.79	0.69	0.73	0.50

(1) AE—DNN-based methods
(2) IF—ML-based methods
(3) LSTM-AE—DNN-based methods
(4) VAR—conventional methods

We implemented the VAR algorithm using the Statsmodels[6] library. For the IF implementation, we began with the original approach provided by the Scikit-learn library for IF.[7] However, since it only considers the spatial dependencies among data points when performing anomaly detection, we modified it following Kardani et al. [16] who have incorporated the k-point moving average into the feature space of each sample. It captures the temporal dependencies of the performance data by obtaining the average of a window of k-previous samples. The AE and LSTM-AE ML models were implemented using the Pytorch[8] library. We employed the Tree-structured Parzen Estimator [6], which is a Bayesian optimization technique, to tune the hyperparameters listed in Table 2 corresponding to each algorithm.

We use F1-score and AUC to evaluate the accuracy of performance anomaly detection algorithms similar to other works in the domain [16, 22]. F1-score is the harmonic mean between precision and recall, while AUC refers to the area under the receiver operating characteristic curve.

We evaluate the four anomaly detection algorithms on a random subset of devices of the SMD dataset by following the Model per Device (MPD) approach. Table 3 shows the evaluation results in terms of average AUC and F1-scores. Figure 6 shows the distribution of AUC and F1-scores across devices for each anomaly detection algorithm. AE achieves the highest mean AUC (0.89) and F1-score (0.79) out of all algorithms. As shown in Figure 6, AE also has the lowest variance.

During our evaluation, we consider both the accuracy and efficiency of the four algorithms. The VAR algorithm does not require a training phase, but it has a high inference time. This results in high latency when detecting anomalies. It also has the least mean and highest variance for its AUC and F1-scores among the compared algorithms. Therefore, VAR is the least suitable anomaly detection algorithm in terms of accuracy and efficiency. IF is a low overhead algorithm with low linear-time complexity and small memory requirements. However, IF-based approaches require a certain percentage of anomaly data for training to achieve their best possible accuracy [16]. This is also evident through the low mean and high variance of AUC and F1-scores obtained in this evaluation, where the IF algorithm was trained only on normal data. Out of the remaining two algorithms, AE has the best mean and least variance for its AUC and F1-scores. LSTM-AE also has

[6]https://www.statsmodels.org/.
[7]https://scikit-learn.org.
[8]https://pytorch.org/.

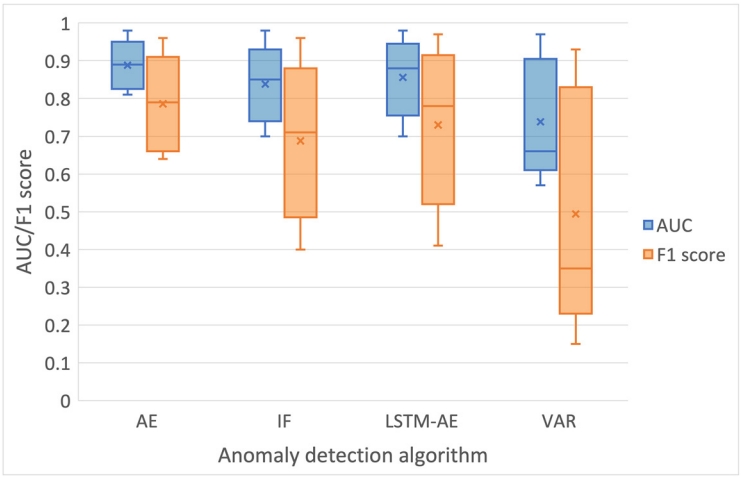


Fig. 6. Distribution of AUC and F1 score of anomaly detection algorithms across devices in SMD.

a nearly equal mean but higher variance on its AUC and F1-scores than AE. LSTM-AE does not outperform the AE algorithm, although it is designed to capture time-series dependencies. Out of these two DNN-based approaches, when considering efficiency, the AE model is less complex, and its inference can be parallelized, unlike an LSTM-AE model. Therefore, we conclude that AE is the most suitable performance anomaly detection algorithm for the rest of our evaluations. AE models used in the rest of the evaluations are trained by minimizing the MSE loss between the original and reconstructed windows using the Adam optimizer. However, the proposed clustering-based training approaches are independent of the ML approach. Hence, a reader can use any other independent model to evaluate the proposed training approaches.

4.3 Evaluating Clustering-Based Training Approaches Using the SMD Dataset

In this section, we evaluate the proposed clustering-based training approaches on the SMD dataset. As the first step, we assign edge devices to clusters. However, since metrics are undefined in this dataset, we cannot use domain knowledge to select the subset of metrics suitable for performing device clustering. Therefore, we use the variance of mean to identify significantly varying metrics across devices to identify the subset of metrics. The variance of mean represents how much the mean of each metric varies across devices. Figure 7 is a bar chart that represents the variance of mean of each metric across devices. As can be seen, columns 23, 6, 24, 26, 7, and 5 have a high variance of mean. However, column 5 has all-zero occurrences in 9 out of 14 devices. Hence, we leave it out of consideration for clustering.

Next, we obtain the collinearity between those metrics. Figure 8 contains the pairplot, which shows the collinearity between metrics shortlisted from the previous step. Since 24 and 26 are linearly correlated, one of those (26) was dropped from the set of metrics for clustering.

Then, we clustered the 14 devices based on the final set of metrics using Algorithms 1 and 2. All the algorithms corresponding to similarity-based clustering and clustering-based training approaches were implemented using Python. The same AE model implementation mentioned in Section 4.2 was used for evaluations in this section as well. We employed the Tree-structured Parzen Estimator [6] Bayesian optimization technique to optimize the hyperparameters of the AE model listed in Table 2 as well as to determine the optimal number of clusters (K) for Algorithm 2. Subsequent to hyperparameter tuning, the optimal number of clusters, K, was identified to be five.

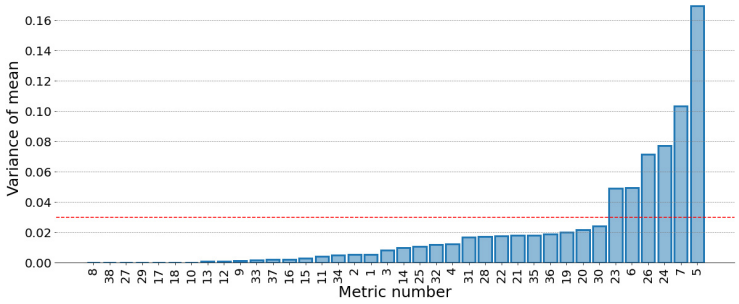


Fig. 7. Variance of mean of each metric across SMD devices.

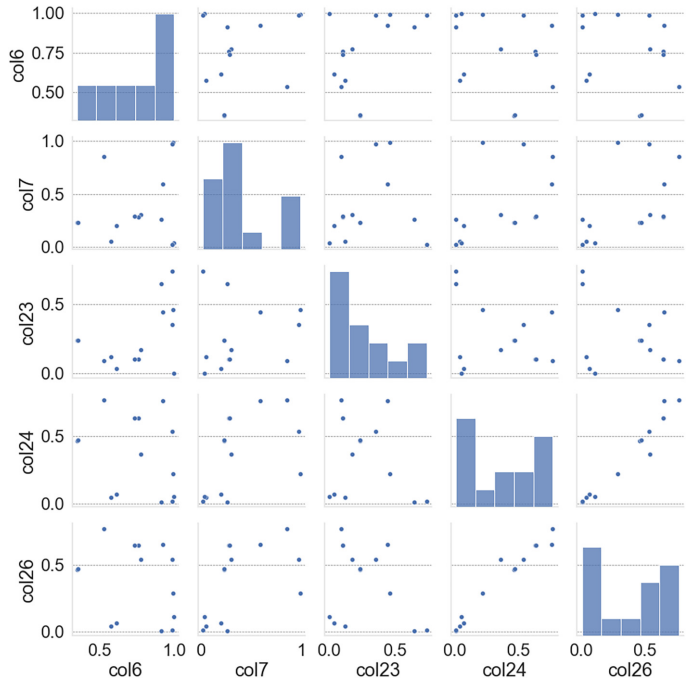


Fig. 8. Collinearity between shortlisted metrics across SMD devices.

Next, we present the evaluation results for the two proposed clustering-based training approaches.

Table 4 contains the accuracy (in terms of average AUC and F1-scores) and training time results of the two proposed clustering-based model training approaches and the two baseline approaches. Figure 9 shows the distribution of AUC and F1-scores across these training approaches. As predicted, the Model per Device (MPD) approach yielded the highest average AUC (0.88) and F1-score (0.77), but it required the longest total training time (19 min 40 secs). On the other hand, the GM approach had the lowest average AUC (0.66) and F1-score (0.35) with the shortest total training time (47 secs). As expected, the ICPTL-based model training approach had the second-highest mean AUC (0.85) and F1-score (0.7) while consuming only 40% of the total training time required by the MPD approach (8 min 2 secs). The CM training approach has further increased training efficiency by consuming 23% less training time than the ICPTL approach (6 min 8 secs) while reducing the

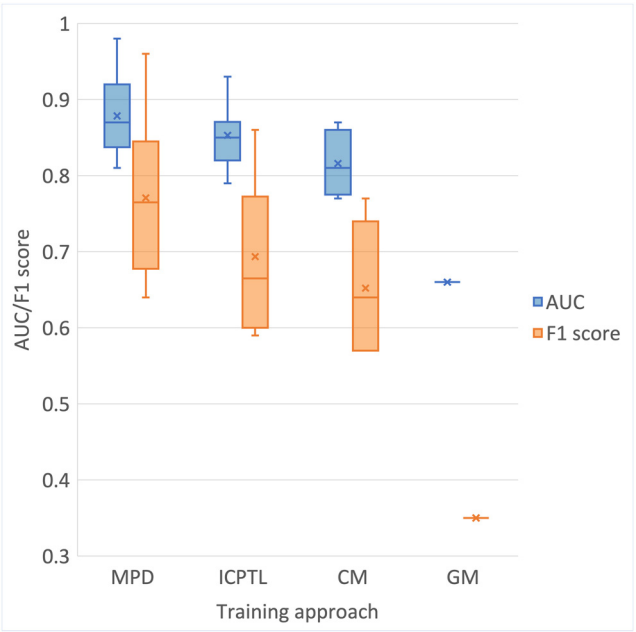


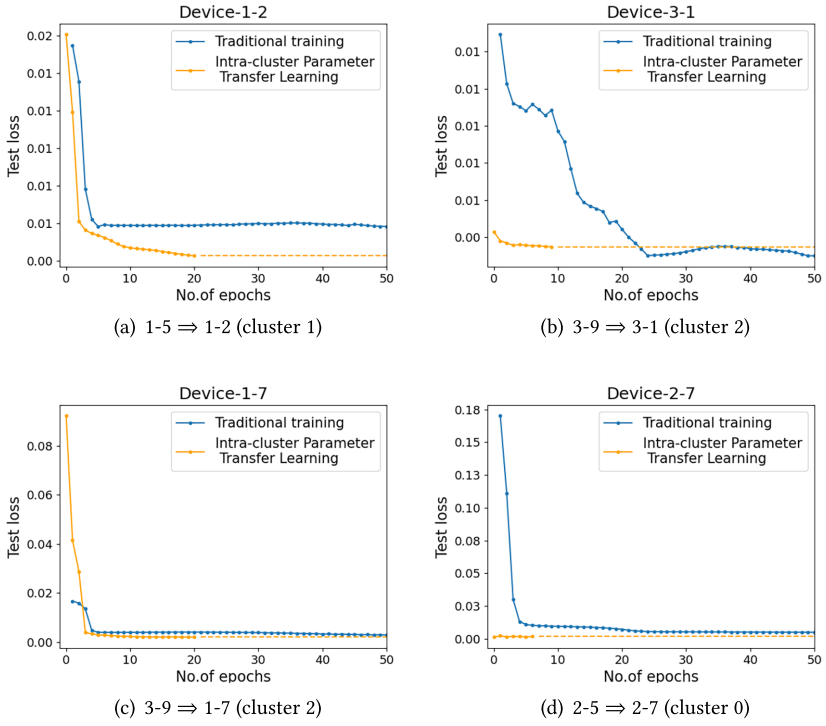
Fig. 9. Distribution of AUC and F1 score of training approaches against SMD.

Table 4. Evaluation Results of Training Approaches against SMD

	MPD	**ICPTL**	**CM**	**GM**
AUC (mean)	0.88	0.85	0.81	0.66
F1-score (mean)	0.77	0.7	0.65	0.35
Total training time	19 min 40 secs	8 min 2 secs	6 min 8 secs	47 secs

total number of models from 14 to 5. Furthermore, the CM approach is 16% and 30% more accurate in AUC (0.81) and F1-score (0.65), respectively, than the GM approach.

Next, we further explore how the ICPTL-based model training approach aims to achieve an accuracy similar to that of the Model per Device (MPD) approach with fewer training cycles. In order to achieve this target, ICPTL should be able to converge to a lower test loss (similar to that of MPD) faster. Towards this, for a few selected devices, we compare the test loss for each epoch in the ICPTL approach against that of the MPD approach (for a duration of 50 epochs). For instance, Figure 10(a) depicts the scenario where the model for "Device-1-2" was traditionally trained using the MPD approach as well as trained from the model for "Device-1-5" using the ICPTL approach. In some scenarios (e.g., Figure 10(b) and (d)), the test set loss obtained with parameter transfer learned model (at its 0th epoch, i.e., without performing any retraining) is already low. Even in some cases where the test set loss of the parameter transfer learning approach is high (e.g., Figure 10(a) and (c)), it converges faster within the first few epochs itself. Regardless of the initial test loss value, the ICPTL approach can converge to a test loss similar to that of the MPD approach faster, thus reaching a satisfactory accuracy within a few epochs, i.e., with no/minimal retraining. Although we have provided results for a few selected scenarios, we could observe the same behavior in other devices as well. In all scenarios, we could observe that the ICPTL approach could reach the



(a) 1-5 ⇒ 1-2 (cluster 1)

(b) 3-9 ⇒ 3-1 (cluster 2)

(c) 3-9 ⇒ 1-7 (cluster 2)

(d) 2-5 ⇒ 2-7 (cluster 0)

Fig. 10. Comparison between test loss of the ICPTL approach and the MPD approach for selected devices.

maximum accuracy (i.e., a low test loss value) before 20 epochs. Therefore, we set the maximum number of epochs to perform parameter transfer learning to be 20.

Furthermore, we deployed an algorithmic stopping approach that stops training when validation loss does not change significantly for five epochs. As a result, in 4 out of 9 scenarios, parameter transfer learning stops before 20 epochs. Figure 11 presents a comparison between the training time results of the Model per Device (MPD) approach and the ICPTL-based model training approach. In terms of training time efficiency, all nine parameter transfer learning scenarios were completed within a maximum of 12.33 secs, whereas the corresponding traditional model training took a minimum of 45 secs. While training the models for the nine devices using the MPD approach took a total of 12 min and 25 secs, all nine parameter transfer learning scenarios completed training, taking only a total of 47 secs, which is only 6% of the training time required by the MPD approach.

Based on our findings, the ICPTL-based model training approach achieves a comparable level of accuracy to the Model per Device (MPD) approach but requires fewer training cycles. The CM training approach demonstrates higher efficiency than the ICPTL approach in terms of total training time and model management overhead while still maintaining a higher accuracy than the GM approach. These results are further validated in the next section by conducting a small-scale experiment using data collected from the emulated edge computing environment.

4.4 Evaluating Clustering-Based Training Approaches Using the Emulated Dataset

In this section, we further evaluate the proposed clustering-based training approaches using the dataset collected from the emulated edge computing environment. First, we clustered the 10 edge devices based on the collected system and application metrics using Algorithms 1 and 2. We identified the optimal number of clusters, K to be three through hyperparameter tuning.

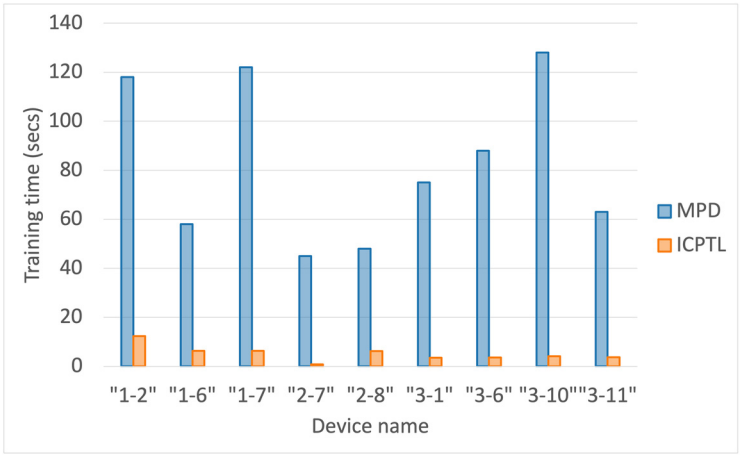


Fig. 11. Training time comparison between ICPTL and MPD approaches against SMD.

Table 5. Evaluation Results of Training Approaches against the Collected Dataset

	MPD	**ICPTL**	**CM**	**GM**
AUC (mean)	0.88	0.88	0.86	0.73
F1-score (mean)	0.95	0.96	0.95	0.9
Total training time	13.98 secs	4.88 secs	3.8 secs	1.14 secs

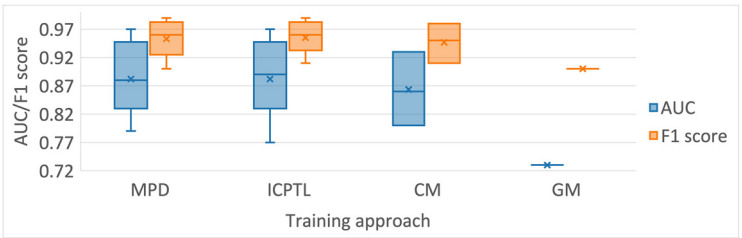


Fig. 12. Distribution of AUC and F1 score of training approaches against the collected dataset.

Table 5 contains the accuracy (in terms of average AUC and F1-scores) and training time results of the two proposed clustering-based model training approaches and the two baseline approaches. Figure 12 shows the distribution of AUC and F1-scores across these training approaches. During this evaluation, it can be observed that the ICPTL-based model training approach had reached the accuracy of the Model per Device (MPD) approach (which is also the highest accuracy: average AUC = 0.88 and average F1-score = 0.96) while consuming only 34% of the total training time required by the MPD approach (4.88 secs). The CM training approach has further increased training efficiency by consuming 22% less training time than the ICPTL approach (4.88 secs) while reducing the total number of models from 10 to 3. Furthermore, the CM approach is 13% and 5% more accurate in AUC (0.86) and F1-score (0.95), respectively, than the GM approach.

Thus, based on our experiment, we can assert that the ICPTL-based model training approach achieves the same accuracy as the Model per Device (MPD) approach in a shorter training time. Furthermore, the CM training approach is capable of reaching higher efficiency than the ICPTL approach while maintaining a greater accuracy than the GM approach.

Table 6. Comparison of Efficiency Metrics across All Approaches in Terms of the Number of Devices (N), the Number of Clusters (K), and the Number of Epochs (l,L) ($1 \leq K \leq N, l << L$)

Approach	Model management overhead	Training time (in terms of the no. of epochs)
GM	1	L
CM	K	$K * L$
ICPTL	N	$K * L + (N - K) * l$
MPD	N	$N * L$

4.5 Discussion

In the evaluations conducted on the SMD (refer to Section 4.3) and the emulated dataset (refer to Section 4.4), we observed that our proposed clustering-based training approaches successfully strike a balance between accuracy and efficiency. In both experiments, the ICPTL-based model training approach achieved a level of accuracy comparable to the Model per Device (MPD) approach but required a shorter training time. Meanwhile, the CM training approach demonstrated higher efficiency, both in terms of total training time and the number of models to manage, than the ICPTL approach while still maintaining a higher accuracy than the GM approach.

Since the dataset size is constant for all approaches during evaluation, we can explain the training time results using parameters such as the number of devices, the number of clusters, and the number of epochs. In addition to training time, as shown in Table 6, we can also compare the number of models that need to be managed across all approaches. If we denote the number of devices as N, the MPD approach needs to train N models, while the GM approach needs to train only one model. If we consider the number of epochs to be L, the MPD approach has to train for $N * L$ number of epochs, whereas the GM approach has to train for only L number of epochs. This explains why the MPD approach has the longest training time, while the GM approach has the shortest training time observed in our evaluations.

Similarly, if the number of clusters is K ($K \leq N$), the CM approach requires training a total of K models. Consequently, to train K models, the CM approach requires $K * L$ number of epochs. On the other hand, the ICPTL approach necessitates training of N models. This approach requires training a source model per each cluster from scratch and fine-tuning the remaining models using a smaller number of epochs l (where $l << L$). Hence, ICPTL approach requires $K * L + (N - K) * l$ number of epochs to train the source and target models. Thus, the training time required for the two proposed approaches is greater than that of the GM approach but less than that of the MPD approach. Additionally, among these two proposed approaches, the CM approach requires less training time than the ICPTL approach.

The number of models to manage is an important factor in ML deployments, as these models require periodic retraining and fine-tuning. As mentioned earlier, the GM approach trains only one model, resulting in the lowest model management overhead. Both MPD and ICPTL approaches require training and managing N number of models, which incur the highest model management overhead. Among the two proposed approaches, the CM approach offers better model management efficiency since it is easier to manage K models (one model per cluster) compared to N individual models (one model per device) in the ICPTL approach.

Although the GM approach requires the least amount of training time, and has the lowest model management overhead, it also results in the least accuracy. In contrast, the MPD approach offers

the highest accuracy at the cost of the longest training time. Thus, the proposed clustering-based model training approaches strike a balance between accuracy and efficiency of model training.

5 Dynamism Handling of Clustering-Based Training Approaches

We proposed the clustering-based training approaches and evaluated them considering the scenario of training models for initial deployment in edge computing environments. However, in dynamic edge computing environments, new edge devices could be added after the first point of deployment, and existing devices could be removed from the infrastructure. Furthermore, the normal data distribution of microservices deployed in edge devices can change due to data drift or osmotic movement of microservices [43]. This dynamism of edge computing environments poses the requirement to retrain performance anomaly detection ML models. Through this section, we explain how our proposed clustering-based training approaches are capable of handling this requirement while maintaining their initial claims regarding accuracy and efficiency.

When a new edge device is added to the infrastructure, it will be assigned a cluster. In the ICPTL-based model training approach, the model corresponding to that device will retrain from its nearest neighbor. In the CM training approach, the existing model of the cluster can be used unless cluster composition is changed.

When an existing edge device is removed from the infrastructure, ICPTL approach does not require any actions. Whereas, in the CM approach, the model can be retrained only if the cluster composition is significantly changed.

Data drift or osmotic movement can sometimes change the cluster to which an edge device is assigned. In that case, in the ICPTL approach, the model corresponding to the device can retrain from its new nearest neighbor's model. In contrast, the CM approach is required to check for changes in cluster composition of both source and destination clusters and retrain if cluster composition has significantly changed. Inclusion of the above-mentioned dynamism handling capabilities in edge environments ensures a self-adaptive edge/microservices environments for detecting performance anomalies.

6 Threats to Validity

Internal validity specifically refers to whether an experimental treatment or condition provides sufficient evidence to support the established claims. In our evaluation, we used the SMD along with our own emulated performance anomaly dataset to conduct experiments. As explained in Section 4.1, we selected the publicly available and widely used SMD dataset for a comprehensive and large-scale evaluation of our proposed clustering-based training approaches. This choice was primarily due to the heterogeneous normal data distributions exhibited by the SMD devices, which align with the expectations of a real edge environment. Additionally, many other edge-related studies have utilized the SMD dataset for evaluation [41, 42], making it the most relevant publicly available dataset that includes real anomalies.

However, during the SMD-based evaluation, we made an assumption that each SMD machine serves as an edge device with a single microservice deployed on it and that the data reported for each machine (edge device) is associated with its respective microservice. To overcome the effect of this assumption on our claim and to further validate our results, we created our own emulated edge dataset. In generating this dataset, we deployed multiple microservices on each edge device to better replicate the microservice deployments found in real edge environments. However, since this dataset is synthetically generated, we believe that presenting results from both of these complementary datasets demonstrates the effectiveness of our proposed approaches.

External validity primarily concerns the generalizability of treatment or condition outcomes. A key question that arises is whether these approaches can be applied to cloud servers and cloud

applications. The answer is that the proposed approaches could be utilized for cloud servers and cloud applications, provided that the edge properties are met. A prime example of this is SMD, where cloud servers exhibit characteristics similar to edge environments. However, these approaches may be less effective in cloud environments, as achieving a tradeoff between accuracy and efficiency is often more crucial in edge settings than in cloud applications. Nevertheless, the proposed approaches are expected to perform effectively in edge-cloud integrated computing environments.

7 Conclusions and Future Work

Since current performance anomaly detection studies in edge computing environments employ different model training approaches that either prioritize training efficiency or accuracy, this article introduces two clustering-based model training approaches : (1) ICPTL-based model training and (2) CM training that aims to find a middle ground between the training efficiency and accuracy of anomaly detection models. These methods involve training models on clusters of edge devices that have similar normal data distributions. The ICPTL approach aims to achieve the same accuracy as the Model per Device (MPD) approach but with fewer training cycles, by identifying a MST for each cluster and then training only the device at the beginning of the tree, while the other devices receive the model from the nearest neighbor device and undergo parameter transfer learning to train for a few epochs until they reach acceptable accuracy. On the other hand, the CM approach aims to improve training efficiency even further by reducing the number of models, by training a single model for each cluster, using normal data collected from all edge devices in the cluster. Self-tuning/repairing performance is a potential beneficiary of this study because the proposed efficient training approaches ensure the accuracy of anomaly detection in a timely manner, thus leading to successful automated mitigation.

The comprehensive and reproducible evaluation conducted on the publicly available and widely used “SMD” revealed that the AE algorithm is the most suitable for anomaly detection due to its high efficiency during inference and the best mean AUC and F1-score compared to other unsupervised approaches.

Upon evaluating the proposed clustering-based training approaches using the AE algorithm, we could observe that the ICPTL approach achieved the second-highest mean AUC and F1-score while consuming only 40% of the total training time required by the MPD approach. Furthermore, the CM approach increased training efficiency by consuming 23% less training time than the ICPTL approach while reducing the total number of models by two-thirds. Additionally, the CM approach demonstrated a 16% and 30% improvement in AUC and F1-score, respectively, compared to the GM approach.

Further experiments conducted on the performance anomaly dataset collected from an emulated edge computing environment reaffirmed the previous results, with the ICPTL approach achieving the accuracy of the MPD approach (which is also the highest accuracy) while requiring only 34% of the total training time. The CM approach further increased training efficiency by consuming 22% less training time than the ICPTL approach while reducing the total number of models by two-thirds. In addition, the CM approach showed a 13% and 5% improvement in AUC and F1-score, respectively, compared to the GM approach.

In summary, the ICPTL approach matches the accuracy of the MPD approach with fewer training cycles, and the CM approach further improves training efficiency by reducing the number of models.

As part of future work, we plan to investigate how to adapt the suggested training approaches to suit various placement strategies apart from QoS and resource requirement-aware placement. Additionally, we aim to explore ways to optimize (especially in terms of efficiency) the subsequent stages of anomaly detection, including anomaly localization and root cause analysis, to better align with the unique features of edge computing environments.

References

[1] Negin Akbari, Adel N. Toosi, John Grundy, Hourieh Khalajzadeh, Mohammad Sadegh Aslanpour, and Shashikant Ilager. 2024. iContinuum: an emulation toolkit for intent-based computing across the edge-to-cloud continuum. In *IEEE 17th International Conference on Cloud Computing (CLOUD '24)*. IEEE, Shenzhen, China, 468–474.

[2] Firas Al-Doghman, Nour Moustafa, Ibrahim Khalil, Nasrin Sohrabi, Zahir Tari, and Albert Y. Zomaya. 2023. AI-enabled secure microservices in edge computing: Opportunities and challenges. *IEEE Transactions on Services Computing* 16, 2 (2023), 1485–1504.

[3] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. 2020. USAD: Unsupervised anomaly detection on multivariate time series. In *26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20)*. ACM, Virtual Event, CA, 3395–3404.

[4] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. 2022. Do deep neural networks contribute to multivariate time series anomaly detection? *Pattern Recognition* 132 (2022), 108945.

[5] Soeren Becker, Florian Schmidt, Anton Gulenko, Alexander Acker, and Odej Kao. 2020. Towards AIOps in edge computing environments. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, Atlanta, GA, USA, 3470–3475.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *25th Annual Conference on Advances in Neural Information Processing Systems*. Curran Associates, Inc., Granada, Spain.

[7] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. 2019. A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems. *Engineering Applications of Artificial Intelligence* 85 (2019), 634–644.

[8] Duneesha Fernando, Maria A. Rodriguez, and Rajkumar Buyya. 2024. iAnomaly: A toolkit for generating performance anomaly datasets in edge-cloud integrated computing environments. In *17th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '24)*. ACM, Sharjah, UAE.

[9] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, Xin Peng, Wenli Zheng, and Minyi Guo. 2021. QoS-aware and resource efficient microservice deployment in cloud-edge continuum. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, Virtual Event*, 932–941.

[10] Fatemeh Golpayegani, Nanxi Chen, Nima Afraz, Eric Gyamfi, Abdollah Malekjafarian, Dominik Schäfer, and Christian Krupitzer. 2024. Adaptation in edge computing: A review on design principles and research challenges. *ACM Transactions on Autonomous and Adaptive Systems* 9 (2024), 1162–1182.

[11] Mirsaeid Hosseini Shirvani and Yaser Ramzanpoor. 2023. Multi-objective QoS-aware optimization for deployment of IoT applications on cloud and fog computing infrastructure. *Neural Computing and Applications* 35, 26 (2023), 19581–19626.

[12] Jonathan Hunter. 2022. *Deep Learning-Based Anomaly Detection for Edge-Layer Devices*. Master's thesis. University of Tennessee at Chattanooga. Retrieved from https://scholar.utc.edu/theses/740/

[13] Mohammad S. Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. 2021. Anomaly detection in a large-scale cloud platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, Madrid, Spain, 150–159.

[14] Mohammad Saiful Islam, Mohamed Sami Rakha, William Pourmajidi, Janakan Sivaloganathan, John Steinbacher, and Andriy Miranskyy. 2025. Anomaly detection in large-scale cloud systems: An industry case and dataset. Retrieved from https://arxiv.org/abs/2411.09047

[15] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. 2020. AI on the edge: Architectural alternatives. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Portoroz, Slovenia, 21–28.

[16] Sara Kardani-Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. 2019. Performance anomaly detection using isolation-trees in heterogeneous workloads of web applications in computing clouds. *Concurrency and Computation: Practice and Experience* 31, 20 (2019), e5306.

[17] Jon Kleinberg and Eva Tardos. 2005. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA.

[18] Solomon Kullback and R. A. Leibler. 1951. On information and sufficiency. *Annals of Mathematical Statistics* 22 (1951), 79–86. Retrieved from https://api.semanticscholar.org/CorpusID:120349231

[19] Juyong Lee and Jihoon Lee. 2018. Hierarchical mobile edge computing architecture based on context awareness. *Applied Sciences* 8, 7 (2018), 1160.

[20] David Chunhu Li, Chiing-Ting Huang, Chia-Wei Tseng, and Li-Der Chou. 2021. Fuzzy-based microservice resource management platform for edge computing in the internet of things. *Sensors* 21, 11 (2021).

[21] Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. 2021. Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21)*. ACM, Virtual Event, Singapore, 3220–3230.

[22] Yoichi Matsuo and Daisuke Ikegami. 2021. Performance analysis of anomaly detection methods for application system on Kubernetes with auto-scaling and self-healing. In *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, Izmir, Turkey, 464–472.

[23] Priyanka Prakash Naikade. 2020. *Automated Anomaly Detection and Localization System for a Microservices Based Cloud System*. Master's thesis. The University of Western Ontario. Retrieved from https://ir.lib.uwo.ca/etd/7109

[24] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. 2019. Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments. In *12th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'19)*. ACM, Auckland, New Zealand, 71–81.

[25] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. 2022. QoS-aware placement of microservices-based IoT applications in fog computing environments. *Future Generation Computer Systems* 131 (2022), 121–136.

[26] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. 2023. Placement of microservices-based IoT applications in fog computing: A taxonomy and future directions. *Computing Surveys* 55 (2023), 43.

[27] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[28] Ella Peltonen and Savidu Dias. 2023. LinkEdge: Open-sourced MLOps integration with IoT edge. In *3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum (ESAAM '23)*. ACM, New York, NY, 67–76.

[29] Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. 2018. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* 137 (2018), 669–685.

[30] Vivek Prasad, Madhuri Bhavsar, and Sudeep Tanwar. 2019. Influence of montoring: Fog and edge computing. *Scalable Computing* 20 (2019), 365–376.

[31] Nikos Psaromanolakis, Vasileios Theodorou, Dimitris Laskaratos, Ioannis Kalogeropoulos, Maria-Eleftheria Vlontzou, Eleni Zarogianni, and Georgios Samaras. 2023. MLOps meets edge computing: An edge platform with embedded intelligence towards 6G systems. In *2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, Gothenburg, Sweden, 496–501.

[32] Emmanuel Raj, David Buffoni, Magnus Westerlund, and Kimmo Ahola. 2021. Edge MLOps: An automation framework for AIoT applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, San Francisco, CA, USA, 191–200.

[33] Vijay Ramamoorthi. 2020. Machine learning models for anomaly detection in microservices. *Quarterly Journal of Emerging Technologies and Innovations* 5, 1 (2020), 41–56. Retrieved from https://vectoral.org/index.php/QJETI/article/view/145

[34] Areeg Samir and Claus Pahl. 2019. DLA: Detecting and localizing anomalies in containerized microservice architectures using Markov models. In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, Istanbul, Turkey, 205–213.

[35] Joseph Schneible and Alex Lu. 2017. Anomaly detection on the edge. In *2017 IEEE Military Communications Conference (MILCOM)*. IEEE, Baltimore, MD, USA, 678–682.

[36] Sotiris Skaperas, Georgios Koukis, Ioanna Angeliki Kapetanidou, Vassilis Tsaoussidis, and Lefteris Mamatas. 2024. A pragmatical approach to anomaly detection evaluation in edge cloud systems. In *International Workshop on Intelligent Cloud Computing and Networking (ICCN '24)*. IEEE, Vancouver, Canada, 1–6.

[37] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) Service-Based cloud applications: A survey. *Computing Surveys* 55, 3 (2022), 39.

[38] Mbarka Soualhia, Chunyan Fu, and Foutse Khomh. 2019. Infrastructure fault detection and prediction in edge cloud environments. In *4th ACM/IEEE Symposium on Edge Computing (SEC '19)*. ACM, Arlington, Virginia, 222–235.

[39] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '19)*. ACM, Anchorage, AK, 2828–2837.

[40] Salman Taherizadeh, Andrew C. Jones, Ian Taylor, Zhiming Zhao, and Vlado Stankovski. 2018. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software* 136 (2018), 19–38. https://doi.org/10.1016/j.jss.2017.10.033

[41] Shreshth Tuli, Fatemeh Mirhakimi, Samodha Pallewatta, Syed Zawad, Giuliano Casale, Bahman Javadi, Feng Yan, Rajkumar Buyya, and Nicholas R. Jennings. 2023. AI augmented edge and fog computing: Trends and challenges. *Journal of Network and Computer Applications* 216 (2023), 103648.

[42] Shreshth Tuli, Shikhar Tuli, Giuliano Casale, and Nicholas R. Jennings. 2021. Generative optimization networks for memory efficient data generation. Retrieved from https://arxiv.org/abs/2110.02912

[43] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. 2016. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing* 3, 6 (2016), 76–83.

[44] Chunrong Wu, Qinglan Peng, Yunni Xia, Yong Jin, and Zhentao Hu. 2023. Towards cost-effective and robust AI microservice deployment in edge computing environments. *Future Generation Computer Systems* 141 (2023), 129–142.

[45] Kai Yang, Hui Ma, and Shaoyu Dou. 2020. Fog intelligence for network anomaly detection. *IEEE Network* 34, 2 (2020), 78–82.

[46] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Dae Ki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations (ICLR '18)*. Vancouver, Canada.

Received 12 August 2024; revised 12 March 2025; accepted 17 March 2025