Machine Learning and Unlearning for IoT Anomaly Detection

by

Jiamin Fan
B.Sc., Nanjing University of Posts and Telecommunications, 2014
M.Sc., University of Victoria, 2018

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

Dr. Issa Traore, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Despite the booming market of the Internet of Things (IoT), the weak security protection of IoT devices makes anomaly detection in IoT systems extremely challenging. This dissertation tackles three critical problems in the anomaly detection of IoT: i) fast update of deep learning-based detection models, ii) the non-independent and identically distributed (non-IID) problem in federated learning (FL) based anomaly detection, and iii) root cause analysis of anomalies.

First, to update deep learning-based detection models in IoT anomaly detection systems, we propose a new machine unlearning method called *ViFLa*, which groups training data based on estimated unlearning probability and treats each group as a virtual client in the federated learning framework. Since the virtual clients are physically in the same machine, *ViFLa* only leverages the concept of data/local model isolation in federated learning without incurring any network communication. To tackle the non-IID problem caused by the data grouping strategy, *ViFLa* designs an enhanced class distribution weighted sum (ECDWS) aggregation method based on Kullback–Leibler divergence and attention mechanism. It also introduces a new state transition ring mechanism into the statistical query (SQ) learning framework to update the local model of each virtual client quickly. Using real-world IoT traffic data, we showcase the benefit of *ViFLa* regarding its efficiency and completeness for model updates in the context of IoT traffic anomaly detection.

Second, we develop a new anomaly detection approach, called *ClusterFLADS*, which depends on clustered federated learning to address the issue of non-IID data amongst different clients in traditional federated detection systems. *ClusterFLADS* takes advantage of the false predictions of inappropriate global models, together with knowledge of temperature scaling and catastrophic forgetting, to reveal distributional similarities between the training data of different clusters and the test data. To improve the clustering speed, we introduce an efficient feature extraction scheme by exploiting the difference in the role each layer of a neural network plays in the learning process. We evaluate the performance of *ClusterFLADS* using real-world IoT trace data in various scenarios. The results show that *ClusterFLADS* can cluster clients accurately and efficiently, with a 100% true positive rate and no false positives over various data distributions.

Third, to cope with the diverse anomaly scenarios that may be encountered by FL-based IoT anomaly detection systems, we design *Score-VAE*, a new framework to

identify the root causes of anomalies based on variational autoencoder (VAE) network. *Score-VAE* can work with existing IoT anomaly detection systems built over the FL framework. To achieve lifelong learning, *Score-VAE* builds a separate global model for each abnormal scenario, so the intervention of new scenarios will not render the existing system unusable. To obtain better generalization and collaboration capacities required by the IoT systems, *Score-VAE* adopts a privacy-preserve training scheme and a Hamming tests scheme. To further improve model performance, *Score-VAE* employs a VAE network with dynamic loss, which exploits knowledge of multi-task learning, stopping gradients and distributions. Evaluation results with real-world IoT trace data collected from different scenarios demonstrate that *Score-VAE* can accurately discover the root causes of alarms triggered by the IoT anomaly detection system.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Supervisor, Dr. Kui Wu, for giving me the valuable opportunity to study with you. Along the way, there are setbacks and gains. It is your guidance, encouragement, and patience that enables me to persevere. Your teaching will benefit me for the rest of my life.

My family for giving me the most selfless love. Always trust me, support me in the low moments, and be happy for the little progress I've made.

My lab classmates for their accompany and useful suggestion, working with you is a very memorable time.

Thank you.
Jiamin Fan

DEDICATION

To my family

Chapter 1

Introduction

1.1 Background: IoT Systems, IoT Security, and Machine Learning

1.1.1 The Internet of Things System

The Internet of Things (IoT), also named the Web of Things, is an application extension of the Internet. The concept of IoT was proposed by Kevin Ashton [3] in 1999, who first worked for Procter & Gamble and then worked for the automatic identification of the Massachusetts Institute of Technology center. In 2004, NetSilicon CEO Cornelius "Pete" Peterson considered that the Internet of Things would be the protagonist of the next era of information technology, far surpassing today's networked computers in pervasiveness and importance. Peterson [4, 5] also predicts that the medical equipment and industrial control fields will be two important stages of applications of this technology. As shown in Fig. 1.2, from 2021 to 2026, the IoT market size is expected to increase from $ 300.3 billion to $ 650.6 billion [6].

We can consider IoT [7] as the "virtual social network", with which users and various devices communicate with each other through the Internet. If the Internet has brought an information age to people, the bounds of information interchange have been significantly widened by IoT, which has also really achieved the intercommunication between people, machines, and things. With such a rich connectivity, the IoT technology can obviously change how we interact with each other and with the environments. As shown in Fig. 1.1, IoT technology has been widely used in household, medical, agricultural, and other fields to make our lives more convenient. For

example, smart home [8] users can control smart appliances with voice commands; By utilizing the most cutting-edge IoT technology to facilitate communication between doctors, medical equipment, and patients, remote consultation [9] can expand medical services to rural areas and ease the pressure of medical demands and resource shortages; IoT sensors can collect environmental information, such as water pollution, in real-time to minimize potential risks; Smart factory [10] can realize intelligent scenarios, such as production safety monitoring and predictive maintenance, by integrating mechanical equipment, computer networks, and artificial intelligence (AI).

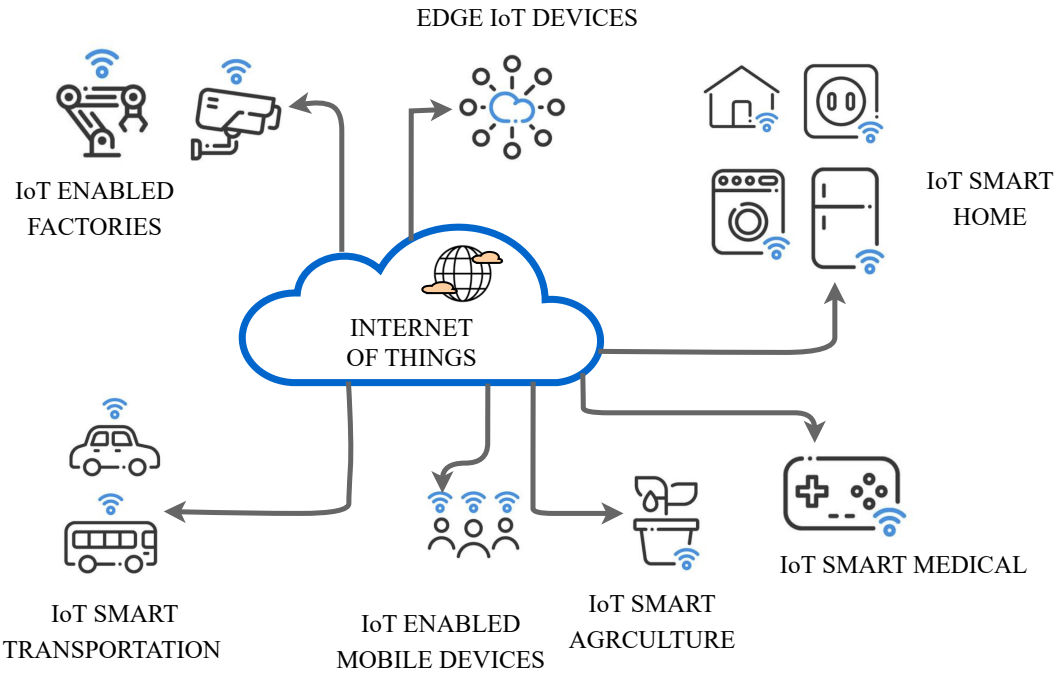


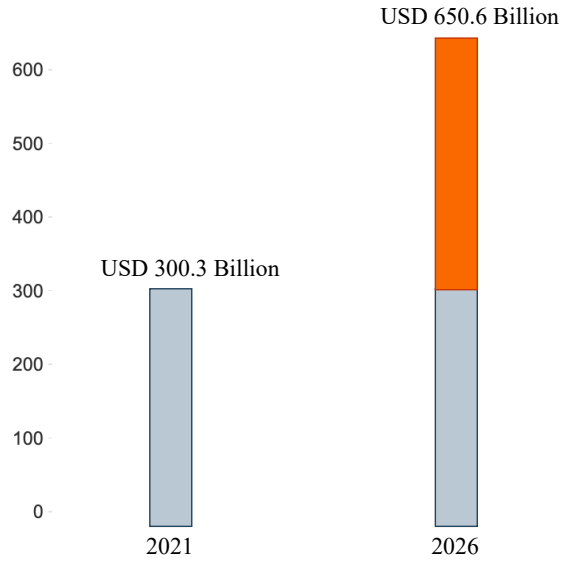
Figure 1.1: The Internet of Things (IoT) can be used in many industries, considerably improving our quality of life.

With the help of sensing devices like laser scanners, radio frequency identification (RFID) [11], the global positioning system (GPS), and infrared sensors, the IoT system interconnects physical objects to the Internet for intelligent monitoring and control. Conceptually, the IoT technology can be divided into three layers [12, 13] from bottom to top: perception layer, network layer, and application layer. As the bottom layer, the perception layer uses devices such as sensors to gather data from the environment. The data from the perception layer is packaged and transmitted to the network layer, which is the middle layer. The application layer, which is the top layer of the IoT system, realizes the control, management, and decision-making

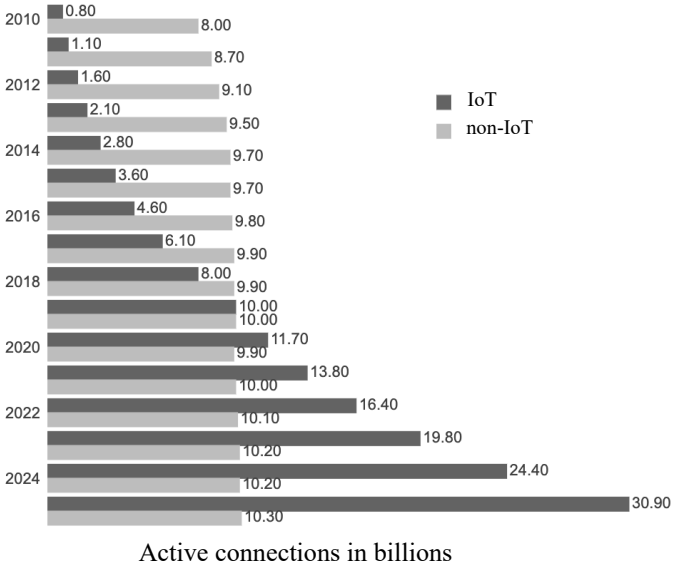
of the real physical world by analyzing the data of the perception layer. A detailed explanation of the functionality of the different layers is described below.

- The perception layer, also referred to as the physical layer, employs sensors to detect and gather meaningful data from the surroundings.
- The network layer, sometimes referred to as the transport layer, serves as a bridge between the application layer and the perception layer. The network layer communicates the data obtained from the perception layer using wired or wireless communication. The routing layer and the encapsulation layer are the two sub-layers of the network layer. The encapsulation layer shapes the packets, while the routing layer transfers them from source to destination. Note that in the IoT domain, the meaning of network layer is broader than the traditional Internet network layer protocol stack.
- The top layer of the IoT system is called the application layer. By calculating and processing the information gathered by the perception layer, the application layer realizes the management, control, and decision-making of the physical environment.

As shown in Fig. 1.3, IoT devices are physical objects equipped with sensors, microprocessors, and IoT software. Via an Internet connection, they are controlled by end users and can communicate with other devices. Compared to traditional Internet applications, IoT applications have the following special features: First, IoT devices are generally small and serve a special purpose. In most applications, users can obtain real-time information via IoT devices. Second, combined with AI [14] technology, IoT devices usually have a certain level of intelligence such as voice recognition and adaptive control. Third, most IoT devices are cheap and thus can be deployed on a large scale. These special features make IoT applications more and more popular in different domains. As shown in Fig. 1.2, it is predicted that there will be 30.9 billion IoT devices deployed in 2025 [15, 16], making them an essential part of our daily life.



(a) Global Internet of Things (IoT) Market.



(b) Estimated global connections (billions) of IoT and non-IoT active devices between 2010 and 2025.

Figure 1.2: The share of the IoT market and the quantity of installed IoT devices.

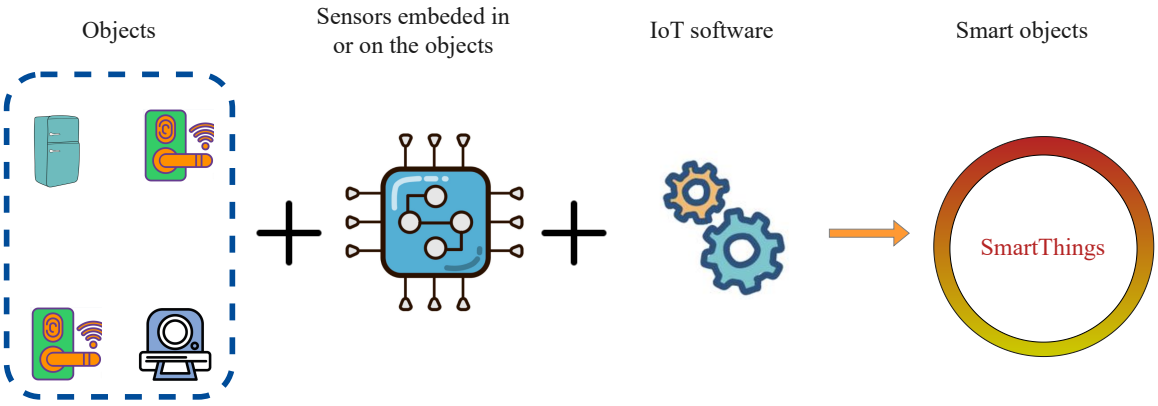


Figure 1.3: IoT devices are physical objects equipped with sensors, microprocessors, and IoT software.

1.1.2 Security of IoT Systems

IoT systems pose both opportunities and challenges. On the one hand, IoT systems have evolved into a fundamental supporting technology in traditional and emerging industries, ranging from smarthome, autonomous vehicles, to metaverse. On the other hand, IoT systems have caused serious concerns in security and privacy breaches. Compared to traditional computers, IoT devices are smaller and constrained in computing/storage resources, making them vulnerable to attacks [17]. To make things worse, many IoT manufacturers quickly launched a large number of new IoT products to occupy the market without carefully considering security, and most users take the security of IoT devices for granted and do not take serious countermeasures against malicious attacks. The consequences of ignoring IoT security has been reportedly disastrous.

Main Threats

By far, the three security threats [18] that are very common in IoT systems are botnets [19], data breaches [20], and shadow IoT [21]. We briefly introduce each threat below.

- Botnets. IoT systems are frequently threatened by Distributed Denial of Service (DDoS) botnet assaults [22]. IoT devices have become the perfect targets for botnets due to their rapid development and rapid popularisation. As shown in Fig. 1.4, IoT devices, such as smart cameras, fire alarms, and door locks, can

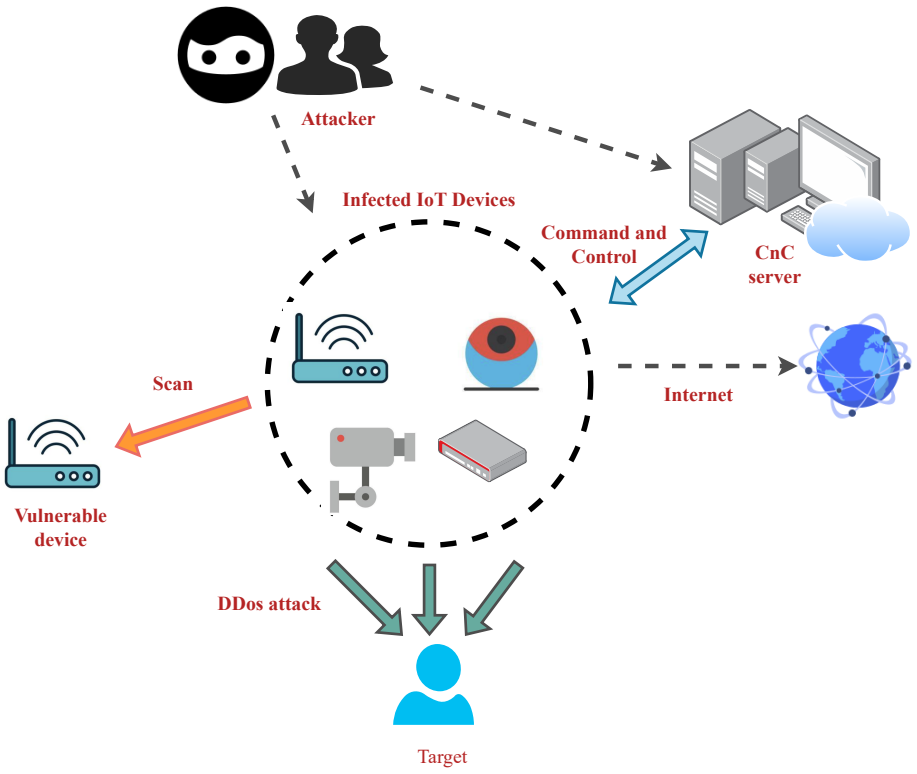


Figure 1.4: Mirai botnet.

be easily compromised due to their weak security design and limited built-in security. What's more serious is the cascading effect: attacking and compromising other IoT devices become much easier with the help of compromised IoT devices.

- Data breaches. When an IoT device is compromised, attackers can gain access to the devise's data to steal sensitive information. Therefore, attacks on IoT devices can lead to serious data breaches, particularly when the IoT devices are tightly linked to people's daily lives. For instance, if a smartwatch is compromised, the owner's daily routine and health data may be immediately visible to the attacker.

- Shadow IoT. Shadow IoT devices refer to those that are connected to an organization's network without being authorized. Shadow IoT devices are common because it is not easy for the administrators of the organization to control the safety of all devices connected to the network if the number of devices become

huge. Even if the users of these shadow devices are not malicious, these devices may cause a huge damage if they are compromised and become bots.

Noteworthy Security Incidents

In addition to the threats previously mentioned, IoT systems also suffer from many other threats. These threats can cause serious consequences, constituting a severe test for IoT systems' healthy and rapid development. Below, we list several noteworthy security incidents in recent years, spanning from 2013 to 2017.

- In 2013, the first botnet in IoT systems was found [23]. The botnet consists of many IoT devices such as smart TVs, baby monitors, and home appliances. In fact, malware such as Crash Override, VPN Filter, and Triton had already been used routinely at that time to compromise IoT systems.

- In 2015, researchers [24] hacked a Jeep remotely and performed a series of operations, such as changing the radio channel, turning on the wipers, and turning on the air conditioner. They claim they can do even more dangerous maneuvers, such as disabling the brakes, causing the engine to stall, slow down, or shut down entirely.

- In 2016, the largest IoT botnet incident [25] was reported. The Mirai malware [25, 26] attacked the website of European data center provider OVH and compromised hundreds of thousands of IoT devices to create a botnet. IoT devices like routers and IP cameras are the most common equipment that are infected. The botnet took down many Internet services, such as Krebs on Security, DYN, and OVH.

- In 2017, two Black Hat security researchers, Billy Rios and Jonathan Butts, demonstrated that they could remotely shut off a heart pacemaker, which would endanger a patient's life [27].

- In 2019, hackers gained control of a couple's smart home devices to carry out a series of disturbing actions [28, 29]. They communicated with the couple through the kitchen camera, turned on music, and made the indoor temperature a scorching 90 degrees Fahrenheit.

- In 2021, Verkada, a security services company located in Silicon Valley, suffered a significant data breach [30]. Hackers compromised numerous security cameras under its management to gain unauthorized access to the company's databases.
- In 2022, a Mirai variant called V3G4 was found [31] to utilize 13 vulnerabilities to spread itself among IoT devices.

Countermeasures

As the IoT industry continues to evolve, the threats now known may be just the tip of the iceberg. There may be many unknown risks threatening the IoT industry's security. Therefore, the IoT system's security protection has received the most investment and attention in the IoT industry, covering the IoT hardware, IoT software, and IoT services. In response to the current IoT security threats, researchers have proposed many countermeasures. The most commonly-used ones are user authentication, network segmentation, and IoT intrusion detection. The user's identity is verified by using advanced authentication techniques like multifactor authentication (MFA). Network segmentation improves the security of IoT systems by dividing the network into smaller segments. Intrusion detection detects attacks by analyzing data streams.

- User authentication [32]. One safeguard of IoT security is the authentication of users. Authentication methods include password verification, biometrics, and more advanced multi-factor authentication (MFA). User authentication has become the first line of defense to hence the security of IoT devices.
- Network segmentation [33]. Network segmentation refers to dividing the network into smaller network segments to improve the IoT system's security. Frequently, users can access only a small portion of the network. Network segmentation not only prevents unauthorized devices from connecting to the network but also prevents infected devices from further infecting other devices. This IoT security technology is often applied to enterprise-level systems.
- IoT instruction detection [34]. An intrusion detection system based on machine learning is currently the most prevalent strategy. Using machine learning models, we can analyze the benign and malicious data streams collected from the IoT network platform. When the system detects abnormal behavior, it issues

an alarm to the network administrator in time. The solutions to IoT intrusion detection generally fall into two categories [34]. The first one is signature-based instruction detection [35]. The underlying assumption is that a particular attack usually has some recognizable patterns (e.g., the attack data packet has a specific byte length or intrusion sequence), which can be used to build the signature of the attack. Clearly, intrusion detection in this category is hard to defend against unknown attacks. The other category is anomaly-based intrusion detection [1], which detects outliers that deviate from the majority of data points or events that do not match defined normal behaviors. The training set for such solutions may contain both normal and abnormal data or only normal data. The former uses a combination of labeled "normal" and "abnormal" data to learn a classifier but has significant limitations due to the imbalance between normal and abnormal categories. The latter is based on the underlying assumption that an IoT device is not a general-purpose computer and it has regular and relatively stable traffic patterns in its normal operation. An IoT device is considered anomalous when its behavior deviates from the normal behavior learned by the model. Benefit from this, solutions in this category can detect unknown attacks.

While the above three approaches are critical for IoT security, we only focus on IoT intrusion detection in this dissertation, largely because recent advancement in machine learning creates unprecedented opportunities to empower IoT intrusion detection techniques, in particular, anomaly-based intrusion detection.

1.1.3 Machine Learning Techniques

Machine learning, a branch of artificial intelligence, primarily creates models for prediction or decision-making by learning from training data. A good machine learning model depends on high-quality massive data and can solve many problems that traditional algorithms are incapable to solve. For example, computer vision techniques can automatically interpret Computed Tomography (CT) to reduce the workload of doctors and the rate of misdiagnosis; The recommendation algorithm can predict the items that the user is most likely to select in the future based on the user's past behavior. Machine learning technology also finds new applications in autonomous driving, such as classifying driving style, recognizing obstacles and targets, and trajectory planning.

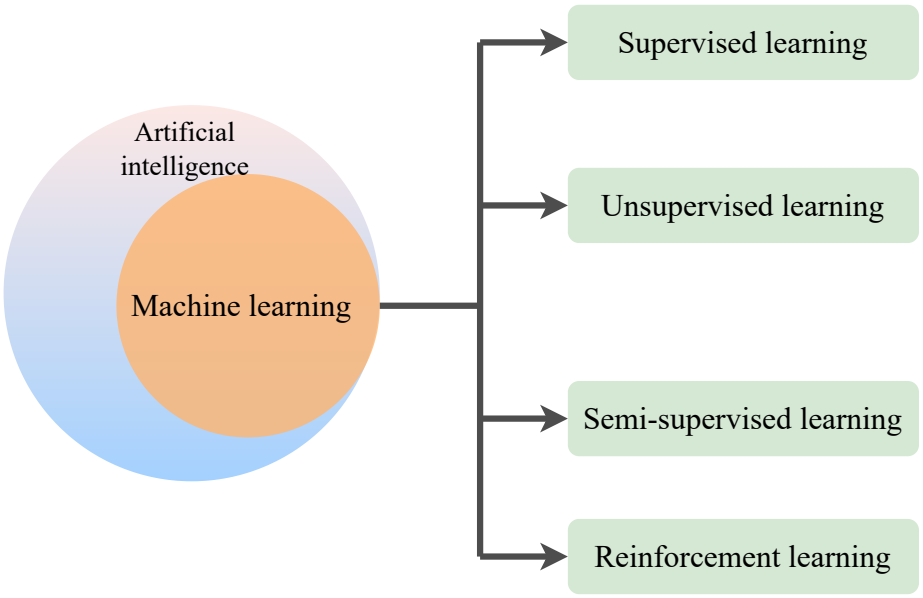


Figure 1.5: Machine learning is a sub-field of artificial intelligence and can be divided into four categories.

As shown in Fig. 1.5, depending on the type of data and task, machine learning algorithms can be categorized into four categories [36]: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Supervised learning [37] uses labeled data samples to train a classification and regression model. Unsupervised learning approaches [38] train the model using unlabeled data's similarities and differences. Semi-Supervised learning [39] takes advantage of both supervised and unsupervised learning. Reinforcement learning [18] learns from the experience by rewarding desired behaviors and punishing undesired behaviors.

- Supervised learning. A model is trained by supervised learning using samples of labeled data. It is further divided into two subcategories: regression and classification. A binary or categorical response is the result of the classification model. The classification algorithm is mainly applied for the recognition of speech, detection of spam, identification of diseases and medical diagnosis, and segmenting and classifying images. A regression algorithm's prediction is based on the correlation between the input and output variables, and the output is a continuous value. Typical applications of regression algorithms include house price and stock price forecasting.

- Unsupervised learning. Unsupervised learning identifies categories based on similarities and differences in data. In this category, training samples are unlabelled. Unsupervised learning is highly suited for clustering techniques like k-means [40, 41]. The data points are grouped into several clusters according to how similar and distinct their properties are, and the meaning concealed in each cluster is then determined. Unsupervised learning has many successful uses. For instance, the recommendation system divides users into different clusters according to their historical purchase behavior, so as to recommend suitable items to them; Banks use unsupervised learning to identify illegal money laundering in transactions. Although we cannot know in advance whether a customer is normal or illegal, we can classify users by their features. Based on the assumption that the behavioral characteristics of normal and illegal customers are very different and the number of illegitimate customers is substantially lower than the number of regular clients, we may easily identify illegal customer groups.

- Semi-supervised Learning. The benefits of supervised and unsupervised machine learning are combined in semi-supervised learning. This category's model is trained using a mixed dataset of labeled and unlabeled data. This category is more suitable for situations where obtaining labeled data is very difficult and expensive. Labeled and unlabeled training data losses are combined to form the loss function for semi-supervised learning. This type of approach is common in image recognition.

- Reinforcement learning. In this category, machine learning models learn from experience rather than from labeled data. The learning agent and the environment are the two components in the framework of reinforcement learning. The agent learns from the experience when it communicates with the environment (rewarding desired behaviors and punishing undesired behaviors). Reinforcement learning does not learn from human-labeled data but learns and explores the environment by itself. Therefore, it has the opportunity to achieve better performance than human ability. A very famous example is AlphaGo, which beat the world champion in the board game Go.

1.2 Motivation of Dissertation Research

As mentioned above, IoT intrusion detection is either anomaly-based or signature-based. The underlying assumption of signature-based intrusion detection is that a particular attack usually has recognizable patterns (e.g., the attack data packet has a specific byte length or intrusion sequence), which can be utilized to build the attack's signature. The main drawback of signature-based methods is that they cannot detect unknown attacks. The anomaly-based solutions detect outliers using the model trained on defined normal behaviors of IoT devices. Building "signatures" for attacks or IoT devices essentially belongs to the pattern recognition problems, for which machine learning methods have been particularly powerful [42]. Due to the above reasons, this dissertation focuses on IoT anomaly detection based on machine learning.

1.2.1 Benefits of Using Machine Learning for IoT Anomaly Detection

The combination of IoT and AI technologies has already attracted widespread attention, called AIoT [43–45]. Machine learning (ML), as a subfield of AI, enables us to effectively analyze and make decisions on the vast amounts of data gathered from IoT devices. While the benefits of applying ML in IoT are many, we only summarize the following two that explain why ML techniques are frequently utilized in IoT anomaly detection systems.

First, ML can learn complex patterns hidden in the data. As introduced above, the booming IoT market has brought us massive amounts of data, whose value can be realized only if we can find the hidden rules and patterns in data. Learning these patterns manually is very expensive and difficult. Compared with expensive and limited manual analysis, ML can process massive data and learn richer hidden patterns and knowledge in a short period of time. ML can thus make more effective predictions and decisions about future events.

Second, ML methods can help preserve data privacy. Despite the fact that a lot of IoT devices produce enormous volumes of data, the owners of these devices may not be reluctant to share data for privacy concerns. In fact, in the absence of privacy protection measures, shared data may be misused and cause irreparable consequences. Federated learning (FL) [46] can effectively solve this problem. In a nutshell, FL

iterates the following steps until convergence: the initial global detection model is downloaded by every client. After that, each client updates the global model with its private data and uploads the updated model to the server. The server aggregates the local models from the clients. The privacy of local users is protected since only model parameter updates are provided during each communication round.

One of the typical works of federated learning-based IoT anomaly detection is introduced by [1]. It designs an efficient IoT anomaly detection system by taking advantage of some special features of IoT applications. For instance, traffic from the same type of IoT devices usually has similar traffic patterns, traffic amount is usually not high for a single IoT device, and so on. They identify the type of IoT devices based on their special traffic patterns and train a three-layer Gated Recurrent Units (GRU) [47–49] network as the anomaly detection model for the same type of IoT devices via the architecture of federated learning. Gated Recurrent Units (GRU), like Long Short Term Memory [50] (LSTM), is a type of recurrent neural network (RNN) [51, 52]. It uses the concept of memory gates to mine time series information and semantic information in data. GRU is suitable for processing sequence data and is mainly used for tasks such as natural language processing. The FL-based approach to anomaly detection [1] laid the foundation for our follow-up research in this dissertation.

1.2.2 Open Challenges

Although machine learning techniques have demonstrated their potential to enhance IoT anomaly detection performance, many problems remain unanswered to address the special requirements in IoT anomaly detection. To be specific, a good IoT anomaly detection system needs to meet the following requirements.

First, the detection model should have a strong ability to update itself when encountering unexpected situations, e.g., training data was previously labeled as normal but identified as abnormal at a later time; an IoT device may update its firmware and cause traffic pattern changes. The problem of incrementally updating the detection model by forgetting the impact of some training data and integrating the impact of newly labeled data is referred to as *machine unlearning*. A naïve method is to retrain the model from scratch with the current training data. Nevertheless, this method is very time-consuming. Cao and Yang [53] trained a naïve Bayes malware detector using 142,350 malware samples and found that it takes nearly a day to delete a

sample and re-train the model from scratch [53]. Taking hours, let alone a day, to update an anomaly detection model in the production environment is not acceptable. This inefficiency is even more prohibitive for time-sensitive applications such as IoT intrusion detection systems. The IoT system is integral to people's daily lives, so it will be extremely inconvenient if it is temporarily unavailable due to model upgrades. Therefore, it's necessary to find a new machine unlearning strategy that can update the model quickly.

Second, the data from different clients (e.g., IoT devices) may not be independent and identically distributed (i.e., non-IID[1]). As introduced above, federated learning is an effective way for using a variety of IoT users' data to train a global anomaly detection model without worrying about privacy issues. Following the experimental setting and the FL-based solution provided in D̈IoT [1], we were able to achieve comparable performance. Nevertheless, good performance is obtained based on the assumption that the data from different clients (e.g., IoT devices) are independent and identically distributed (i.e., IID). Unfortunately, we found this assumption may not hold in reality due to the unstable network environment of the FL architecture, e.g., duplicate TCP packets triggered by TCP retransmissions due to link failures or network congestion. Actually, we have observed quite different traffic patterns when the same smart camera with the exactly same configuration is used in a home and in a university lab. The global model's performance will suffer a considerable decline when the data from different clients (e.g., IoT devices) are non-IID. Although many improvements have been introduced [54–57], most of them still aim at building one global model for all clients. These solutions can only improve the effect of aggregation to a certain extent, and cannot fundamentally solve the problem. In other words, aggregating the models trained on non-IID clients may not achieve the same performance as IID clients. The reason is that local models trained on non-IID data may negatively affect each other when we aggregate them [54, 58, 59] using the aggregation algorithms in conventional federated learning (such as FedAvg). Thus, a single global model cannot satisfy the needs of all clients in the case of non-IID data.

Third, in practice the anomaly alarms could go off for a number of different reasons, not all of which are malicious attacks. For example, the unstable IoT network environment may trigger a false alarm. A normal behavior that has been seen in

[1]The term "IID" is an abbreviation that expands to "Independent and Identically Distributed". The concept of independence and identical distribution assumes that the samples are independent of each other and identically distributed. Non-IID refers to any settings beyond IIDnesses.

one environment (e.g., a smart home) may be reported as an anomaly in another environment (e.g., an office). Even under the same environment, the performance of the local/global models may vary significantly over time. If the alarm's underlying reason remains unclear, the operator must immediately rush to deal with each alarm, which will greatly increase the workload of the operator. By identifying the false alarms caused by benign network environment changes from real malicious attacks, operators can focus on alarms that are real threats that need to be dealt with urgently. Therefore, it is critical to identify the underlying cause that triggers the alarms.

1.3 Research Objectives and Contributions

The purpose of this dissertation is to tackle the aforementioned challenges in IoT anomaly detection: i) fast model update, ii) non-iid problem during model aggregation, and iii) root cause analysis of anomalies. To address the first problem, we propose a machine learning approach called *ViFLa* to update ML-based detection models in IoT systems, which can ensure both efficiency and completeness. To address the second problem, we design a clustered federated learning approach for IoT anomaly detection. To address the third issue, we introduce a brand-new system for root cause analysis, called *Score-VAE*. *Score-VAE* can work together with existing IoT anomaly detection systems built over the FL framework. The background knowledge, motivations, challenges, contributions, and evaluation results of each research question are succinctly described below.

1.3.1 Fast Model Update for IoT Traffic Anomaly Detection with Machine Unlearning

Background Knowledge

It is often needed to update machine learning-based models in traffic anomaly detection systems for the Internet of Things (IoT) when device firmware upgrades cause small traffic changes or when some training data that were previously labeled as normal but identified as abnormal at a later time [60]. The task of updating an existing ML model to remove the impact of certain training data on the model is termed as *machine unlearning*. Clearly, a naïve method for machine unlearning is to re-train the ML model from scratch using the complete set of updated training data. This

method is also called *naïve unlearning*.

Research Motivation

Unfortunately, *naïve unlearning* is very time-consuming, which is ineffective for time-sensitive applications such as an intrusion detection system. As such, we focus on machine unlearning methods that quickly update the anomaly detection model without re-training the model from scratch. The criteria to evaluate a machine unlearning approach should consider both efficiency and completeness. Here, efficiency refers to how fast the approach can update the model, and completeness refers to how close the performance of the updated model is to that of naïve unlearning. There is a clear trade-off between efficiency and completeness: naïve unlearning achieves the best completeness but the worst efficiency. A natural question is: *could we achieve efficiency and completeness at the same time?*

Challenges and Main Contributions

We propose a novel machine unlearning method called *ViFLa* to update deep learning-based detection models in IoT anomaly detection systems, either because of mislabelled samples or due to device firmware upgrades. The time for updating the model depends on the amount of data to be unlearned and the complexity of the learning algorithm. This type of solution mainly includes two approaches. One is to reduce the amount of data affected by the unlearning process. The other is to reduce the time complexity of the unlearning algorithm. *ViFLa* belongs to the second category. It groups training data based on their estimated unlearning probability and treats each group as a virtual client in the federated learning framework. By doing this, when an unlearning request arrives, the samples included in the unlearning request may be relevant to only one or very few local models. The majority of local models do not need to be re-trained. We, however, need to address two technical challenges in machine unlearning with *ViFLa*.

- First, intentionally partitioning data with high unlearning beliefs into a small number of clients will cause data from different virtual clients to become non-independently and identically distributed (non-IID). Handling non-IID data in federated learning is a challenging task [54].

- Second, each virtual client needs to quickly update its local model without re-training from scratch.

To address the first question, *ViFLa* adopts an attention-based aggregation method called enhanced class distribution weighted sum (ECDWS) to tackle the non-IID data problem caused by the data grouping strategy. It aggregates the outputs of local models to obtain the final prediction result based on the concepts of KL-divergence [61] and self-attention [62, 63]. For the second question, *ViFLa* introduces a new state transition ring mechanism into the statistical query (SQ) learning framework to update the local model of each virtual client quickly.

Evaluation Results

Using real-world trace data, we thoroughly evaluate the performance of *ViFLa*, covering not only the effectiveness of its individual components but also its benefit in different application scenarios. Overall, *ViFLa* can achieve similar accuracy of re-training from scratch with significant speedup. We also performed a theoretical analysis of the efficiency of *ViFLa* using SISA [64] as a baseline. Compared to the baseline, *ViFLa* can reduce the computational complexity and the amount of data affected by unlearned samples. This work has been published in [65].

1.3.2 Taking Advantage of Mistakes: Rethinking Clustered Federated Learning for IoT Anomaly Detection

Background Knowledge

In an FL-based anomaly detection system, non-IID data among various clients might result in poor performance of the trained global model, because local models can adversely affect each other when they are aggregated [54, 58, 59]. In this situation, a single global model is not sufficient to serve the demands of every client. The non-IID problem in the spatial domain (i.e., non-IID data among different clients) for FL can be addressed by Clustered federated learning (CFL) [59, 66–69]. Instead of building one global model [54–57], CFL aggregates clients with similar data distributions into the same cluster and trains multiple global models that adapt to different clusters.

Research Motivation

Existing clustered federated learning (CFL) solutions can be generally classified into two categories according to whether there is knowledge (e.g., model parameters) sharing between clusters. Nevertheless, we encountered several problems when we try to apply these solutions to the IoT anomaly detection system.

- First, the non-IID problem in the IoT network affects both the spatial domain (i.e., data from different clients are non-IID) and temporal domain (i.e, the training and test data collected in sequence from the same environment can also be non-IID). The spatial domain non-IID problem has been well understood and studied, but very few research has touched the temporal domain non-IID problem. The temporal domain non-IID problem is mostly caused by two factors: i) the variation of network configuration and condition, and ii) duplicate TCP packet transmissions caused by link failure and/or network congestion. The non-IID problem of IoT systems in the temporal domain could degrade the accuracy of CFL, since the test data (of each cluster) may differ significantly from the training data used to create the global model (in that cluster). Thus, the first question we need to answer is: *how can we increase the precision of IoT anomaly detection by addressing the spatial-temporal non-IID issue under the existing CFL scheme?*

- Second, the requirements for the time delay are typically strict for IoT anomaly detection systems. A long time delay in model updates may lead to the temporary unavailability of services, which can cause severe impacts on clients. Nevertheless, the current clustering procedure in existing CFL solutions requires multiple iterations [59, 67, 70], which is time-consuming and may cause an unacceptable time delay in IoT anomaly detection. Thus, we also need to answer: *how can we re-design the clustering procedure in the current CFL to improve its time efficiency while ensuring the clustering performance?*

Challenges and Main Contributions

To answer the above two questions for more accurate and efficient anomaly detection in IoT systems/applications, we are faced with two major challenges.

- First, with newly-collected test data from an IoT system, the distributional difference between the test data and the training data cannot be easily measured.

This difference may change over time in IoT systems: for one period, it's possible that the training data and the test data are quite similar, while for another period, they may differ signficantly due to network congestion or configuration changes.

- Second, model filtering (i.e., weeding out useless noisy parameters) can be used to slim down the cluster models and speed up the clustering process. How to discard noisy parameters and leave useful ones is non-trivial since most model parameters from neural networks may not directly correspond to a practical meaning in the real-world.

In this dissertation, we provide a new cluster federated learning method called *ClusterFLADS* for IoT anomaly detection. In order to solve the non-IID problem in the temporal domain, *ClusterFLADS* uses knowledge of temperature scaling [71, 72], and catastrophic forgetting [73, 74], and takes advantage of the false predictions from inappropriate global models to reveal distributional similarities between the training data of different clusters and the test data. We also design an efficient feature extraction scheme by exploiting the difference in the role each layer of a neural network plays in the learning process. By strategically selecting model parameters and using the PCA method for dimensionality reduction, *ClusterFLADS* can effectively improve clustering speed.

Evaluation Results

To evaluate the performance of *ClusterFLADS*, we use real-world IoT trace data from diverse scenarios. For comparison, we introduce two baselines, which i) adopt traditional federated learning, and 2) adopt the key algorithm of the clustered federated learning framework called IFCA [59], respectively. Evaluation results demonstrate that our technique significantly outperforms the baselines, and can cluster clients accurately and efficiently, with a 100% true positive rate and no false positives over various data distributions.

1.3.3 *Score-VAE*: Root Cause Analysis for Federated Learning-based IoT Anomaly Detection

Background Knowledge

FL-based IoT anomaly detection systems may trigger alarms for various reasons, not all of which are malicious attacks. This is because the traffic patterns in the network environment vary over time and over different locations (we call these spatial-temporal dynamics in network environments), e.g., duplicate TCP packets triggered by TCP retransmissions due to link failures or network congestion. If the problem is not handled well, a large number of false alarms could overwhelm the network administrator. Hence, it becomes critical to tell apart the root causes for the raised alarms. Existing root cause analysis methods mainly rely on expert knowledge or machine learning models to infer the root cause of anomalies. In the former, expert knowledge is used to formulate specific rules [75–77] for determining the root cause of alarms. The the latter, machine learning methods such as Bayesian networks, decision trees, and neural networks, are used to train root cause classifiers.

Research Motivation

In our context, expert knowledge-based solutions may not work well since we usually do not have enough expertise to establish effective rules for IoT systems. Different from traditional Internet applications, IoT is an expanding sector that has grown quickly in recent years. Short development history, rapid replacement, and uneven products make it particularly difficult to acquire the expertise to identify different IoT anomaly scenarios. Due to this reason, we mainly focus on ML-based root cause analysis.

Challenges and Main Contributions

In this research, we need to address the following problems.

- First, existing solutions lack the generalization ability required by IoT systems. Due to environmental factors such as network congestion, abnormal data with the same root cause will also exhibit large fluctuations. Additionally, cunning attacks may take new forms. Hence, we hope that the system can correctly find the most-likely cause of the abnormality, instead of being confused by the fluctuation from the same cause.

- Second, existing solutions are mainly trained and applied locally, ignoring the collaboration capabilities in the IoT system. The variety and complexity of IoT anomaly scenarios make it challenging for local users to train a powerful root cause analysis model alone. Hence, we need to design a root cause analysis system that fits well with the FL-based anomaly detection system.
- Third, IoT root cause analysis systems should have the capability of lifelong learning to adapt to new anomaly scenarios. IoT technology is changing rapidly, and the known anomaly scenarios may only be the tip of the iceberg. It is challenging to train a root cause analysis model that can account for all possible anomalous circumstances. Therefore, the system should have the ability to continuously learn new knowledge and self-renew.

To address the above problems, we propose *Score-VAE*, a novel root cause analysis system that can collaborate with existing IoT anomaly detection systems built on the FL architecture. *Score-VAE* fully leverages the generation and reconstruction power of the variational autoencoder (VAE) network. To improve the generalization capacity, our system trains a separate global model for each anomaly scenario and relies on the reconstruction effect of the different scenario's global VAE root cause analysis models on the test sample to judge its category. Benefiting from this multi-model framework, we can also build global models for new emerging scenarios without disturbing existing ones. To jointly train a powerful global model without leaking privacy and to further improve the generalization capacity, we let the actual input to the local VAE network only contain the distance relations of the scaled symbols in the original samples, and use the decoders uploaded by all the local models to participate in the training of a certain scenario. We then generate new samples to train the global model for that scenario. Both the local models and the global models are realized through our dynamic VAE network, where the weights for different losses change dynamically during training based on the knowledge of stop gradient [78–80] and multi-task learning [70]. Furthermore, we propose a new test scheme based on Hamming distance. Benefiting from the dynamic VAE network and the Hamming test scheme, we can effectively alleviate the negative impact of noises.

Evaluation Results

Evaluation results with real-world IoT trace data collected from different scenarios demonstrate that *Score-VAE* can accurately discover the root causes of alarms trig-

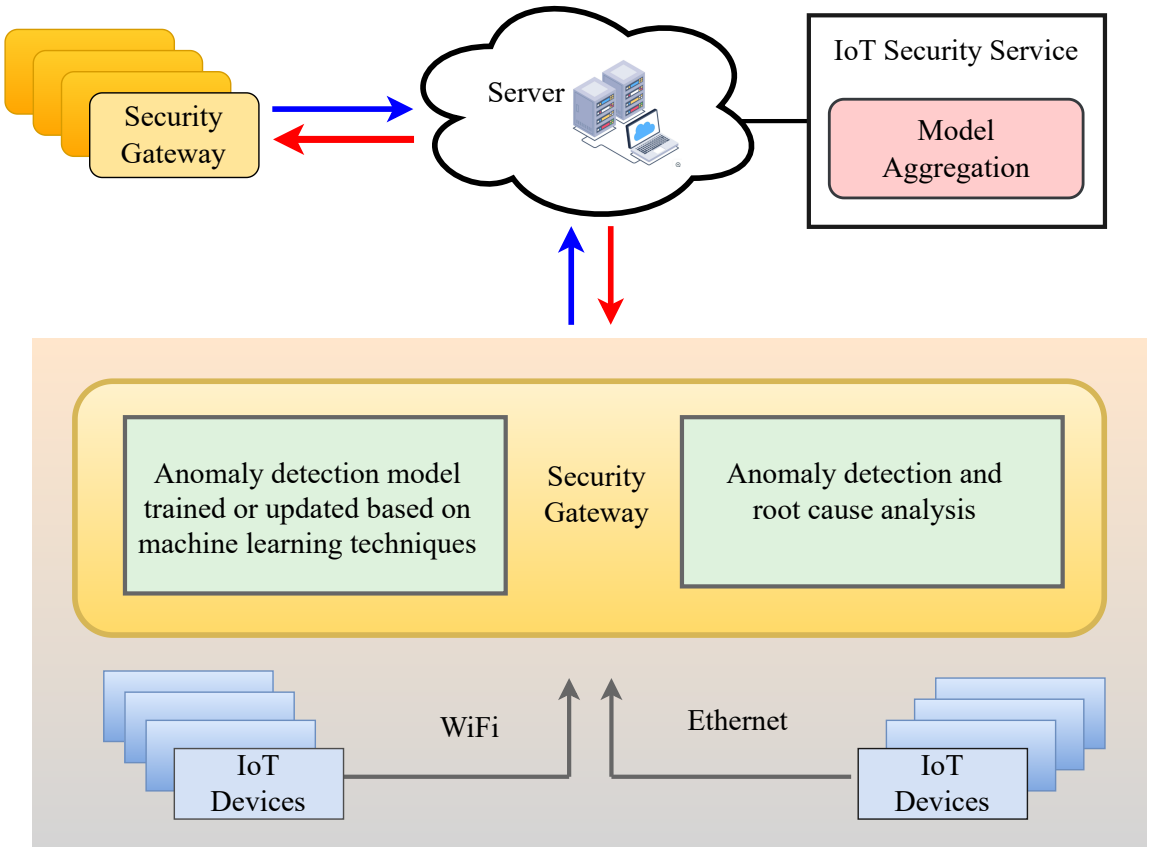


Figure 1.6: The three research problems related to IoT anomaly detection.

gered by the IoT anomaly detection systems and obtain stronger generalization ability and better performance compared to the baselines.

1.4 Dissertation Outline

Overall, this dissertation solves three research problems related to IoT anomaly detection: fast model updates, clustered FL, and root cause analysis. The remainder of the dissertation is structured as follows.

Chapter 2 proposes a novel machine unlearning method called *ViFLa*, which is deployed at network vantage points, e.g., the gateway in Fig. 1.6. The anomaly detection model trained over the *ViFLa* framework can be quickly updated whenever necessary, e.g., when an IoT device firmware update causes traffic changes.

Chapter 3 proposes a clustered federated learning-based approach to IoT anomaly

detection called *ClusterFLADS*. It takes advantage of the inappropriate global models' false predictions, along with knowledge of temperature scaling and catastrophic forgetting, to reveal distributional similarities between the training data (of different clusters) and the test data.

Chapter 4 develops a framework named *Score-VAE* for alarms' root cause analysis. According to the privacy-preserving training scheme and the Hamming test scheme, *Score-VAE* well meets the collaboration, privacy-preserving, generalization, and lifelong learning capabilities required by the root cause of IoT systems.

Chapter 2

Fast Model Update for IoT Traffic Anomaly Detection with Machine Unlearning

2.1 Introduction

The Internet of Things (IoT) technology has triggered and enriched many intelligent applications such as smart factories, smart transportation, and smart homes. Unlike traditional Internet applications, IoT systems have special features, e.g., cheap and easy to deploy, that make them popular but more vulnerable [1, 81]. Recently, deep learning-based anomaly detection systems have been developed to safeguard IoT systems [1]. As an essential requirement, the IoT traffic anomaly detection system needs to update the underlying machine learning model when people upgrade the firmware of an IoT device that causes small traffic changes or when some training data that were previously labelled as normal but identified as abnormal at a later time [60]. The task of updating an existing ML model to remove the impact of certain training data on the model is termed as *machine unlearning*.

Clearly, a naïve method for machine unlearning is to re-train the ML model from scratch using the complete set of updated training data. This method is also called *naïve unlearning*. Nevertheless, *naïve unlearning* is time-consuming and ineffective for time-sensitive applications such as an intrusion detection system. Cao and Yang [53] trained a naïve Bayes malware detector using 142,350 malware samples and found that it takes nearly a day to delete a sample and re-train the model from scratch [53].

Taking hours, let alone a day, to update an anomaly detection model in the production environment is not acceptable. As such, we focus on machine unlearning methods different from naïve unlearning.

The criteria to evaluate a machine unlearning approach should consider both efficiency and completeness. Here, efficiency refers to how fast the approach can update the model, and completeness refers to how close the performance of the updated model is to that of naïve unlearning. There is a clear tradeoff between efficiency and completeness: naïve unlearning achieves the best completeness but the worst efficiency. A natural question is: *could we achieve efficiency and completeness at the same time?* We offer a positive answer for machine unlearning in IoT traffic anomaly detection by leveraging the special features related to IoT traffic and the concept of virtual federated learning.

Several recent research efforts have been devoted to machine unlearning. For instance, Bourtoule et al. [64] proposed a Sharded, Isolated, Sliced, and Aggregated (SISA) training framework, which divides large training tasks into small sub-tasks and only re-trains the shards containing the data points that need to be unlearned. Cao and Yang [53] used statistical query (SQ) to transform the learning algorithm into summation form and only updated the summations relevant to the data points that need to be unlearned. The summation forms of simple algorithms such as naïve Bayes and k-means are provided in [53]. To speed up machine unlearning, all the above methods isolate the impact of training data within a small scope. The idea of data isolation, *in principle*, is similar to federated learning [46, 82, 82, 83], where each client trains a local model with local data, and they work together to build a global model.

We frame the data isolation principle in machine unlearning with a new concept called virtual federated learning. The main idea of virtual federated learning approach (*ViFLa*) for machine unlearning is as follows: We estimate the unlearning belief values of training samples (unlearning belief refers to the likelihood that a sample in a particular application context is to be unlearned in the future), and divide the training samples into different groups based on their unlearning belief values. Each group is considered as a *client* in the traditional federated learning, and thus a local model is trained for each client. The outputs of local models are aggregated (at a virtual server) using an attention-based aggregation method called enhanced class distribution weighted sum (ECDWS) to obtain the final prediction result. Intuitively, if the data points to be unlearned are contained in a small number of clients, we

can effectively speed up the unlearning process because only the clients affected by unlearned data points need to re-train their local models. We call this approach *virtual* federated learning approach because the clients and server are only conceptual and have no physical analogues in the real world. ***Virtual clients are used for limiting the impact of data in a small scope, and thus do not have any issue concerning data privacy, network communication, and delay***. Note that *ViFLa*, SISA [64], and SQ-based unlearning [53] all follow the same principle of data isolation but have quite different underlying mechanisms for building the final model, as further illustrated in Section 2.4.5.

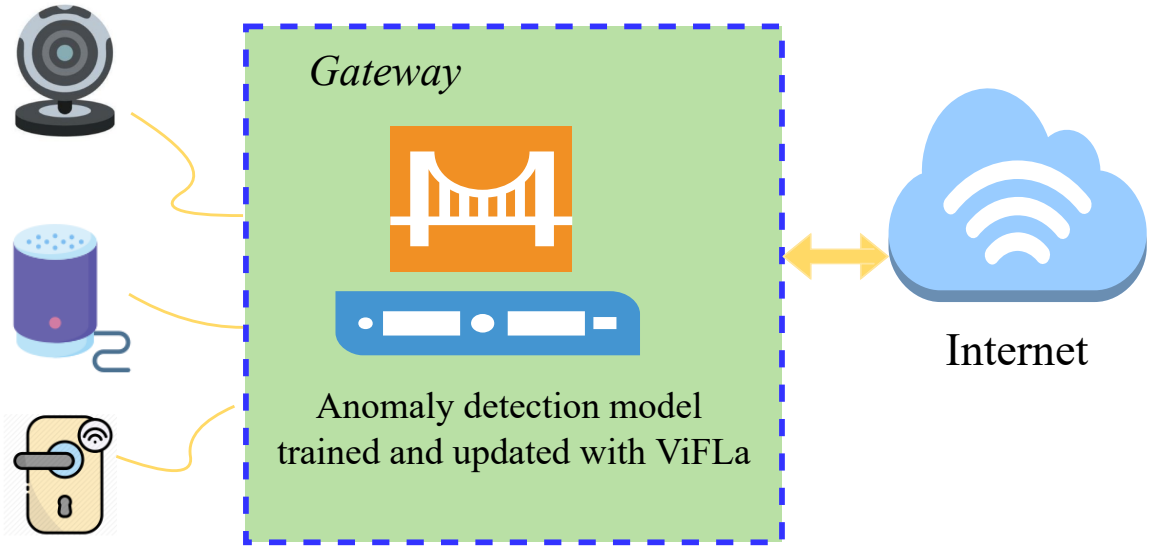


Figure 2.1: *ViFLa* at the gateway for training and updating anomaly detection model. The detailed structure of *ViFLa* is in Fig. 2.3.

We need to tackle two technical challenges in machine unlearning with *ViFLa*. First, since we intentionally divide data of high unlearning belief into a small number of clients, it is likely that the data of different virtual clients are not independent and identically distributed (non-IID). Handling non-IID data in federated learning is a challenging task [54]. Second, each client needs to quickly update its local model without re-training from scratch.

The proposed approach systematically addresses the above challenges and makes the following contributions:

- We present a novel framework called *ViFLa*, which is deployed at network vantage points (e.g., the gateway as in Fig. 2.1) for IoT traffic anomaly detection.

The anomaly detection model trained over the *ViFLa* framework can be quickly updated whenever needed, e.g., when an IoT device firmware update causes small traffic changes. Since the virtual clients and the virtual server can be located in the same physical location (e.g., the gateway), *ViFLa* does not need to consider the network communication and delay between the clients and the server (the disadvantage of traditional federated learning).

- Tackling the first challenge: Different from existing model aggregation methods in traditional federated learning, *ViFLa* adjusts the weights for the outputs of local models using the KL-attention mechanism, which improves the collaborative performance of local models by using their predicted vectors. The KL-attention mechanism is highly linked to data distribution and global intent and can achieve good performance in the presence of non-IID data.

- Tackling the second challenge: *ViFLa* includes a fast unlearning method for local models by introducing a state transition ring into SQ-learning. We formulate a new adaptive SQ-learning definition [84] for the LSTM network. The state transitions of parameters are represented by the nodes and directed edges in the ring. Benefiting from the summation [53] form and the state transition ring, model parameter updates are much faster than *naïve unlearning*.

- We evaluate the performance of *ViFLa* on IoT-23 dataset [85] and our own dataset to demonstrate the advantages of *ViFLa*. *ViFLa* can obtain a significant speed-up and similar test performance compared to naïve unlearning. A theoretical analysis shows that compared to SISA [64], *ViFLa* can reduce the computational complexity and the amount of data affected by unlearned samples.

2.2 Related work

Existing machine unlearning solutions can be roughly divided into (1) solutions that modify model parameters directly without accessing the training data, and (2) solutions that update the model with the training data.

2.2.1 Modifying Model Parameters Directly without Accessing to the Training Set

Solutions in this category modify model parameters without accessing the original training data during the unlearning process. Golatkar et al. [86] took an information-theoretical approach and proposed a "scrubbing" method. The objective of unlearning is translated to calculating optimal noise to destroy information carried in the data we wish to forget. Actually, the technique used by Golatkar et al. is equivalent to avoiding forgetting the data that we wish to retain. To be more specific, use the Fisher Information Matrix (FIM) for the samples we wish to keep, and add optimal noise to destroy information in the samples we wish to forget, i.e., add noise to destroy the weights that may have been informative about data to be forgotten but not data to be retained. Clearly, Fisher Information Matrix (FIM) and its variation are the key to letting the model remember the information in retained samples after the "scrubbing" process. The main pitfall of this approach is the high storage and computation overhead in the operations of FIM.

Loosely, "optimal brain damage" introduced by LeCun et al. [87] falls in this category. Nevertheless, the main goal is to reduce the complexity of a deep neural network by removing unimportant weights from a network. The basic idea is to use the second derivative information to measure the impact of parameters and delete those parameters that have the least effect on the training error.

2.2.2 Updating the Model Quickly with the Training Set

Solutions [64,88,89] in this category update the model quickly with the help of training data. The time for updating the model depends on the amount of data to be unlearned and the complexity of the learning algorithm. This type of solution mainly includes two approaches. One is to reduce the amount of data affected by the unlearning process. The other is to reduce the time complexity of the unlearning algorithm.

Regarding the first approach, Ginart et al. [88] proposed a divide-and-conquer k-means algorithm to divide the data into leaves and merge the results into parent nodes. Bourtoule et al. [64] proposed a SISA framework to reduce the amount of data that needs to be unlearned. The main idea is to divide large training tasks into small sub-tasks. To remove the impact of a data point from the model, this method only needs to retrain the shard containing this data point. However, the storage overhead of the slicing process in this method may be high. Aldaghri et al. [90] encoded the

training data into shards using linear encoders prior to the learning phase. However, this encoded machine unlearning approach was for simple regression models and may be hard to apply in deep learning models. Brophy et al. [91] introduced a machine unlearning method specifically for random forests. It is still unclear how to extend the technique for neural network models.

Regarding the second approach, Cao and Yang [53] used statistical query (SQ) to transform the learning algorithm into summation form [53]. To remove the impact of a data point from the model, we only need to update the summations that involve the data and then update the model with the updated summations. Since only partial summations need to be updated, the process is much faster than *naïve unlearning*. The work [53] only includes the summation form of simple algorithms, such as naïve Bayes [92–94] and k-means [40, 41]. In [95], Neel et al. utilized convex optimization and reservoir sampling to design a gradient-based machine unlearning method called descent-to-delete. Nevertheless, this work is mainly theoretical and depends on strong assumptions, e.g., the convexity of the model, which may not be true in practical IoT anomaly detection systems.

As shown in Fig. 2.2, *ViFLa* belongs to the second category, i.e., updating the model quickly with the training set, but it is different from all existing solutions in this category. To be more specific, it uses the concept of virtual federated learning to reduce the amount of data affected by unlearning. It also uses smart partition and a new model aggregation method. Compared to the existing SQ approach [53], *ViFLa* formulates a summation form of more complex models (e.g., LSTM network), and uses a state transition ring algorithm to speed up the unlearning process. It can achieve high efficiency and completeness for both IID and non-IID data.

2.3 Details of *ViFLa*

2.3.1 Overview

A high-level view of *ViFLa* is shown in Fig. 2.3, where the training, testing, and unlearning processes are denoted with black, blue, and red arrows, respectively. The training of *ViFLa* consists of two steps: smart partition and local model training. For testing, each sample is input simultaneously to all the local models and *ViFLa* uses an enhanced class distribution weighted sum (ECDWS) to aggregate the outputs of local models to obtain the final prediction result. When machine unlearning is needed,

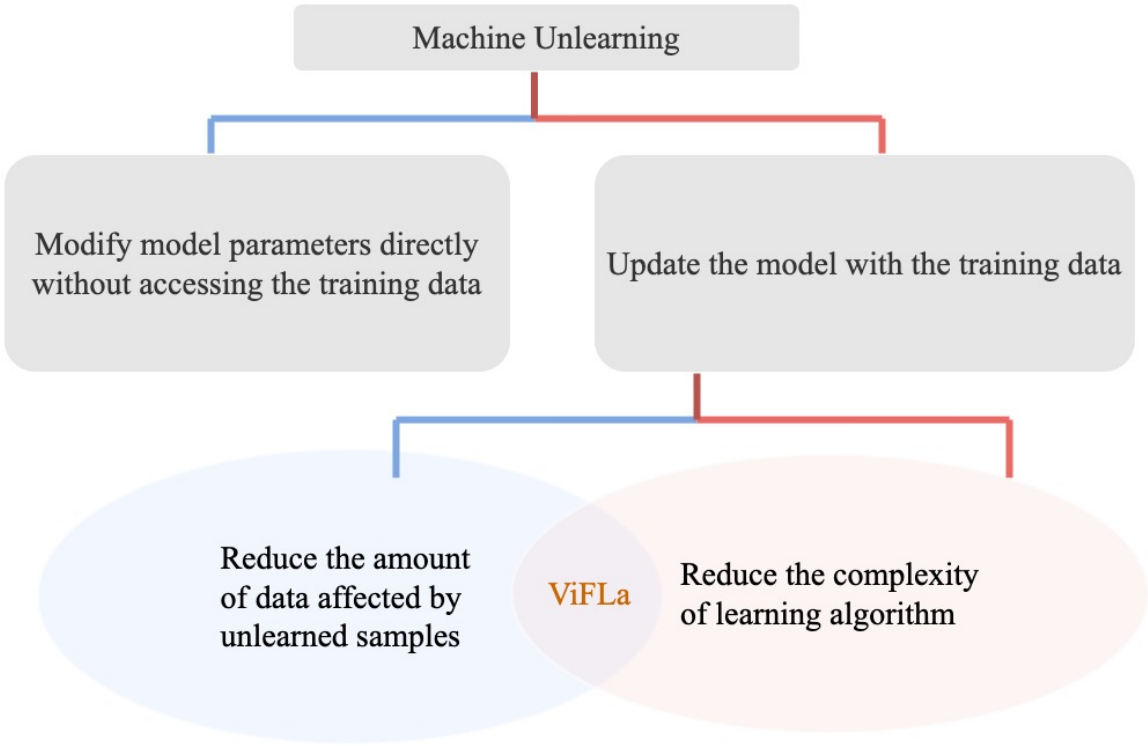


Figure 2.2: Classification of machine unlearning approach.

ViFLa performs model update with the method introduced in Section 2.3.4.

For ease of reference, the main notations used in the paper is listed in Table 2.1.

2.3.2 Training *ViFLa*

ViFLa includes mainly two steps in training:

- **Smart partition**: this step divides training samples into different groups based on their unlearning belief values. Each group is treated as a "client" as in traditional federated learning.
- **Local model training:** this step trains a local model for each virtual client. To speed up the (future) unlearning process of the local model, the local model is formulated in the summation form and the local model is trained according to the summation form.

It is worth noting that *ViFLa* does not use an explicit global model. Instead, it uses an enhanced class distribution weighted sum (ECDWS) method, which is different from existing model aggregation methods in traditional federated learning.

ECDWS only uses the prediction results of the local models and does not use any local model parameters.

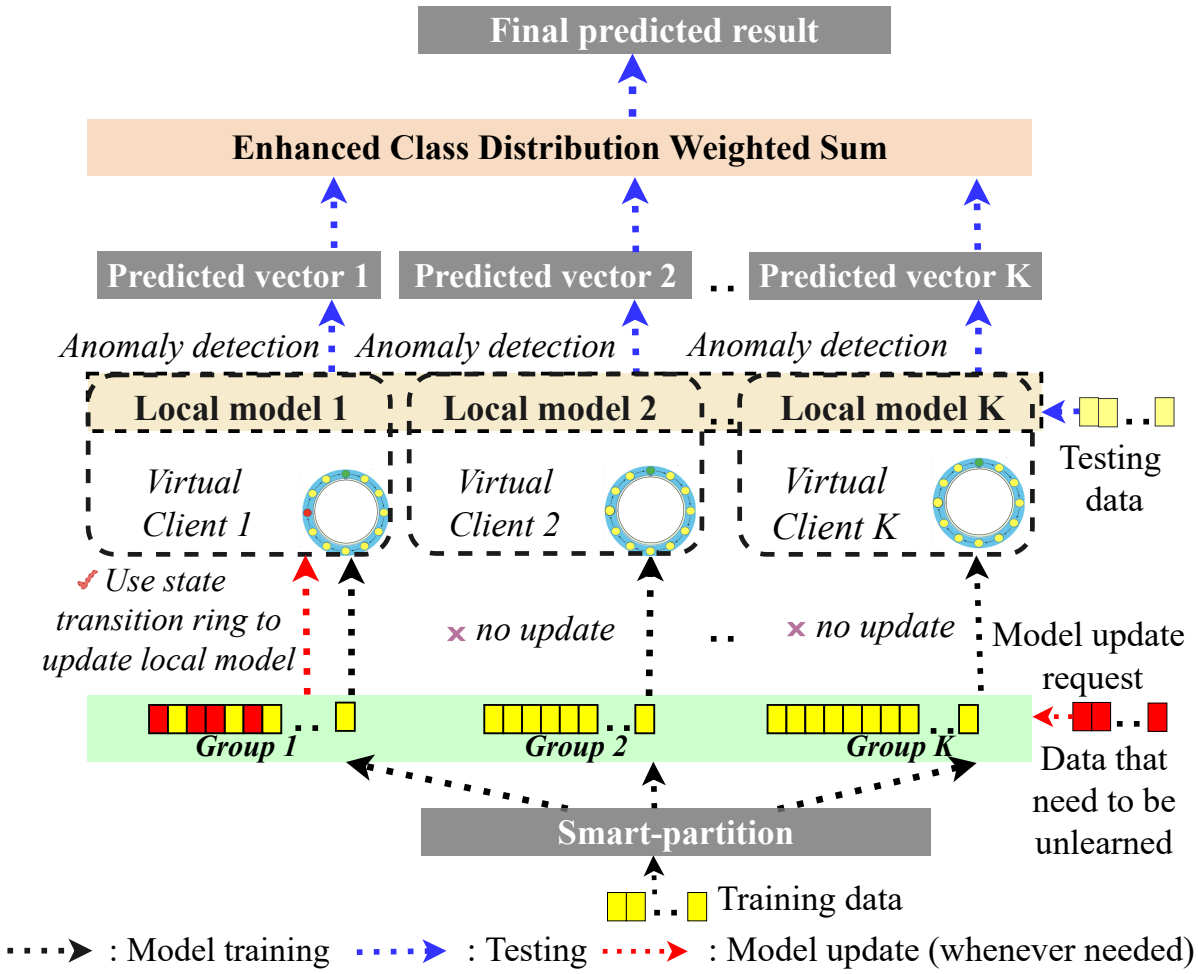


Figure 2.3: Architecture of *ViFLa*.

Smart Partition

Unlearning belief refers to the likelihood that a sample in a particular application context is to be unlearned. If we group samples of high unlearning beliefs together and build a local model with these samples, when an unlearning request arrives, the samples included in the unlearning request may be relevant to only one or very few local models. The majority of local models do not need to be re-trained.

Clearly, accurate estimation on unlearning belief is key for smart partition. Such an estimation depends on a specific application context, and there exists no one-

Table 2.1: Main notations

Notation	*Definition*
x	The feature of training sample
l	Actual class value of training sample
ϵ	A user-defined unlearning threshold
G	The number of classes
K	The number of sub-models
F	The number of features
$p(l\|x_1, x_2, ..., x_F)$	The predicted probability that the sample belongs to its actual label l
$p(l)$	The probability that the sample has actual class value l
$p(x_f)$	The predicted probability that the training samples have feature value x_f
$p(x_f\|l)$	The probability that the training samples have feature value x_f given that the actual class value is l
n_i	A node in state transition ring
e_i	An edge in state transition ring
B	The number of mini-batches
θ_{S_j}	The parameter state in node n_j
$g_{\theta_{S_j}}(batch_b)$	The gradient of $batch_b$ at state θ_{S_j}
$f_{\theta_{S_j}}(batch_b)$	The mapping result of a sample $batch_b$ of query SQ_j at state θ_{S_j}
A_{SQ_j}	The SQ answer at parameter state θ_{S_j}
$b^*_{x,l}$	Unlearning belief of sample (x, l)
$\hat{A}_{SQ_j}$	The SQ answer at parameter state θ_{S_j} during unleaning
c_i^l	Sub-model i's reliable value on class l
N_l	The number of samples of class l
$N_{i,l}$	The number of samples of class l in group i
v_i	The prediction of sub-model i's outputs
v_i^l	The probability that a test sample is predicted as class l by sub-model i
$\mathcal{R}(j)$	The similarity vector for candidate vector v_j
$\mathcal{R}(j)^{SUM}$	The sum of all the elements in the similarity vector $\mathcal{R}(j)$
$\mathcal{R}(j,i)$	The similarity of prediction vectors between sub-model j and sub-model i
$\sigma(\mathcal{R}(j,i))$	The softmax result of each element $\mathcal{R}(j,i)$ of the similarity vector
v	Final predicted vector

* The first part of notations is for smart partition; the second part is for state transition ring; the third part is for naïve class distribution weighted sum method (NCDWS) and enhanced class distribution weighted sum method (ECDWS).

fit-all solution. As an example, we use naïve Bayes classifier [92–94] to estimate unlearning belief due to its simplicity yet surprising efficacy in many complex real-world situations [14, 96].

Let $X = (X_1, X_2, \ldots, X_F)$ denote F features of training samples, where each feature takes value from its domain D_f. Let Z denotes the classes of the training samples, where Z can take one of G values $\{1, \ldots, G\}$. The details for preparing training samples from raw IoT traffic data can be found in Section 2.4.2.

Given the feature value $x = (x_1, x_2, \ldots, x_F)$ of a training sample, the predicted probability that the sample belongs to its actual class $l (1 \leq l \leq G)$ can be calculated as:

$$p(l|x_1, x_2, ..., x_F) = \frac{p(l) \prod_{f=1}^{F} p(x_f|l)}{\prod_{f=1}^{F} p(x_f)}, \tag{2.1}$$

where $p(x_f)$ denotes the predicted probability of training samples with feature value x_f, $p(l)$ denotes the probability of training samples with actual class value l, $p(x_f|l)$ denotes the probability of training samples with feature value x_f given that the actual class value is l.

We propose to estimate the unlearning belief value for the sample $(X = x, Z = l)$:

$$b^*_{x,l} = e^{-p(l|x_1, x_2, ..., x_F)}. \tag{2.2}$$

The above estimation is based on the observation that the higher the probability $p(l|x_1, x_2, ..., x_F)$, the higher the confidence that the system predicts the sample as class l, and the lower the unlearning belief. As shown in Fig. 2.4, a lower unlearning belief value implies a higher confidence in the prediction result, e.g., the probability that the system predicts the sample as class l in the context of intrusion detection of Internet of Things (IoT). An unlearning belief value greater than ϵ is considered as a high unlearning value, where ϵ is a user-defined unlearning threshold.

After we estimate the unlearning belief for each training sample, we group the training samples according to their unlearning belief. To be more specific, given the number of groups K, we order the training samples based on their unlearning belief values and then evenly divide the ordered samples into the K groups. In this way, we try to contain the data points that are to be unlearned in a small number of groups. After that, each group is treated as a client as in the federated learning that trains a local model using the data in the group. Refer to Section 2.4.5 for the explanation on why smart partition works in practice.

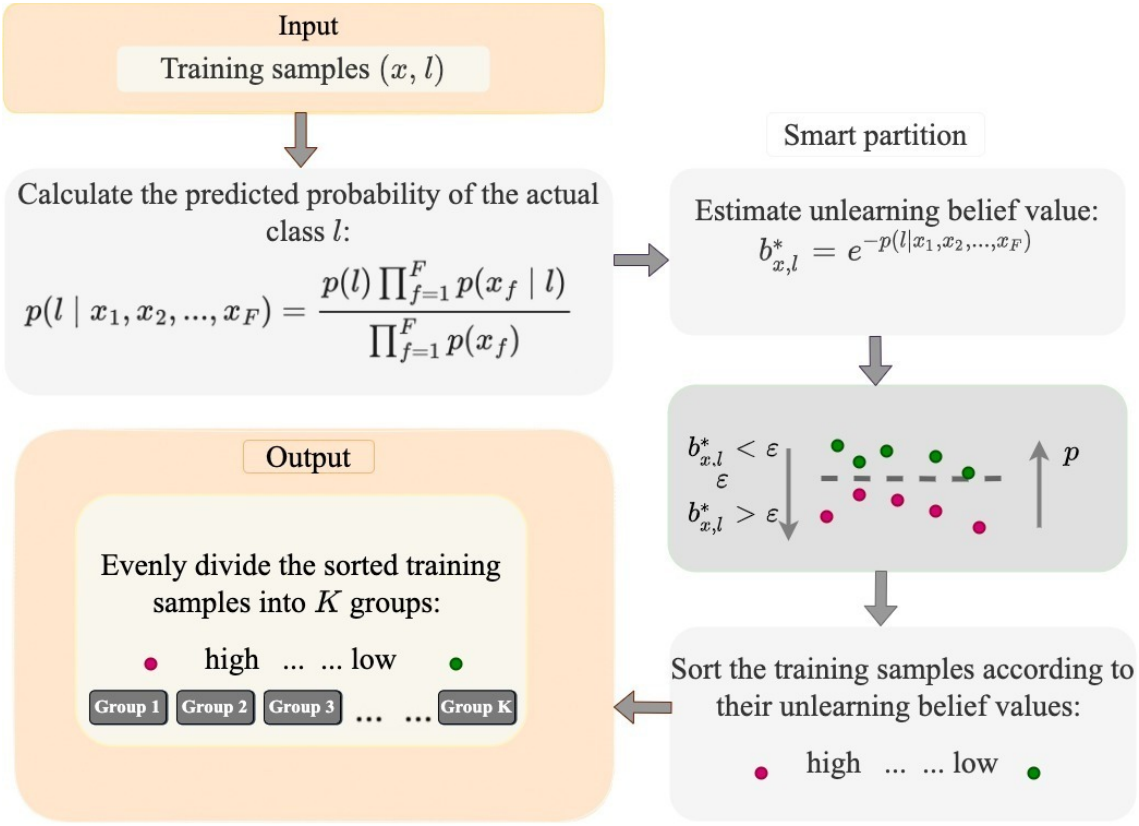


Figure 2.4: Workflow of smart partition. The range of the unlearning belief is $0.37 \leq b^*_{x,l} \leq 1$ since $0 \leq p(l|x_1, x_2, ..., x_F) \leq 1$. ϵ is a user-defined threshold.

Local Model Training

Each group is considered as a *virtual client* in the FL framework. We use the LSTM network [97] as an example to illustrate the training method. To speed up future unlearning, we adopt statistical query (SQ) learning [53] to transform the LSTM network into summation form, whose details are given in Appendix. Each LSTM local model is trained with mini-batch gradient descent, and the training of local models is independent of each other. In addition, we introduce a new state transition ring mechanism as shown in Fig. 2.5. The state transition ring involves B nodes $(n_0, n_1, \ldots, n_{B-1})$ and B directed edges $(e_0, e_1, \ldots, e_{B-1})$, where B is the total number of mini-batches. The state in node n_j is represented as θ_{S_j}. We use node n_0 as the root node and store the initial parameter state θ_{S_0} in this node.

Statistical query (SQ) means the learning algorithm can only query statistics about the training samples. We provide query SQ_j for each state θ_{S_j}. The query con-

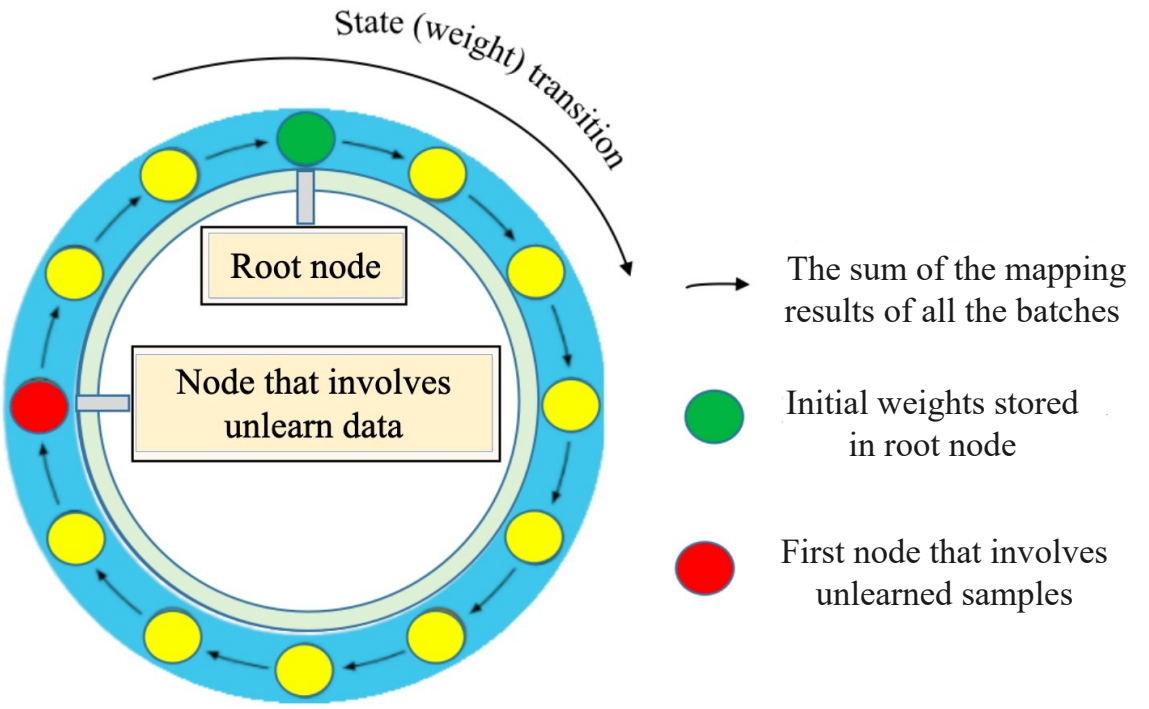


Figure 2.5: State transition ring.

sists of a mapping function and a set of training batches $Q = \{batch_0, batch_1, \ldots, batch_{B-1}\}$. The mapping result of a sample $batch_b \in Q$ of query SQ_j at state θ_{S_j} is calculated as,

$$f_{\theta_{S_j}}(batch_b) = \begin{cases} g_{\theta_{S_j}}(batch_b) & \text{if } b = j \\ 0 & otherwise \end{cases} \tag{2.3}$$

where $g_{\theta_{S_j}}(batch_b)$ is the gradient of $batch_b$ at state θ_{S_j}.

We then calculate the summation-form answers of these statistical queries (SQ). The answer of a statistical query SQ_j is the sum of the mapping results of all the batches in the training set, denoted as $A_{SQ_j} = \sum_{b=0}^{B-1} f_{\theta_{S_j}}(batch_b)$. Using the answer A_{SQ_j}, the parameter state θ_{S_j} (stored in node n_j) is transferred to the state $\theta_{S_{j+1}}$ (stored in the node n_{j+1}). After updating all the states in the ring, we set the latest state θ_{S_B} as the new initial state θ_{S_0} and store it in the root node n_0. We repeat this process until convergence is determined by "early stopping" [98], i.e., the accuracy on the validation set has reached the expected value, and the accuracy will not exceed this value in the next few iterations. The pseudo-code for local model training is shown in Algorithm 1.

Remark. The state transition ring is not a new machine learning algorithm but just a summation form [53] of traditional ML algorithms that make the machine unlearning process faster. Specifically, it degenerates the converged state in the learning process to the previous state by updating the summation form. This previous state serves as the initial parameter state to learn a new convergent state. We can thus speed up the unlearning process since the new initial state is close to a convergent state. It is also worth noting that an epoch in the traditional form of ML algorithms means one iteration over all training data. When one or several epochs are trained, we update the checkpoint of the model and only one checkpoint (state) is saved. In contrast, the state transition ring keeps a separate state for each batch and stores it in a node. Each yellow node in the ring corresponds to a batch and a model state. During the learning process, the state stored in a node is only updated when the batch corresponding to its previous node is trained.

Algorithm 1: Local Model Training in *ViFLa*

Input: Initial parameter state θ_{S_0} stored in root node n_0, mini-batches $Q = \{batch_0, \ldots, batch_{B-1}\}$

Output: state $\theta_{S_1}, \ldots, \theta_{S_{B-1}}, \theta_{S_0}$ in the last epoch

repeat
  for *SQ* $j \leftarrow 0$ **to** $B-1$ **do**
    for *Batch* $b \leftarrow 0$ **to** $B-1$ **do**
      if $j = b$ **then**
        $f_{\theta_{S_j}}(batch_b) = g_{\theta_{S_j}}(batch_b)$;
        $A_{SQ_j} = A_{SQ_j} + f_{\theta_{S_j}}(batch_b)$;
      else
      end
    end
    $e = (j+1) \mod B$;
    Transfer state θ_{S_j} stored in node n_j to state θ_{S_e} stored in node n_e by using the summation form SQ answer A_{SQ_j};
  end
until *Convergence*;
return *Final state* $\theta_{S_1}, \ldots, \theta_{S_{B-1}}, \theta_{S_0}$ *stored in nodes* $n_1, \ldots, n_{B-1}, n_0$, *respectively;*

2.3.3 Testing *ViFLa*

Each test sample will be input to all local models simultaneously. After that, *ViFLa* uses an enhanced class distribution weighted sum (ECDWS) to aggregate the outputs of local models to obtain the final prediction result. Before presenting ECDWS, we first introduce a naïve class distribution weighted sum method (NCDWS), and then explain how ECDWS overcomes the main pitfall of NCDWS.

Naïve Class Distribution Weighted Sum (NCDWS)

The motivation of NCDWS is to alleviate the impact of non-iid data in different local models since the aggregation performance is highly linked to the data distribution. In this method, we analyze the class distributions in different local models *in advance*, and determine the aggregation parameters based on the class distribution. For a test sample, each sub-model i outputs its prediction,

$$v_i = \begin{bmatrix} v_i^1 & v_i^2 & \dots & v_i^G \end{bmatrix}^T, \tag{2.4}$$

where v_i^l $(1 \leq i \leq K, 1 \leq l \leq G)$ denotes the probability that the sample is predicted as class l. The predicted vectors in different local models will be used as input vectors of NCDWS. One simple solution is to use the weighted average over all the predicted vectors. However, this method may suffer from non-IID data since the distribution of training data largely impacts the test performance. To address the problem, we define a class-based reliable value for each sub-model. The reliable value is used to represent the reliability of the prediction result of each sub-model for different classes, denoted as

$$\begin{Bmatrix} c_1^1 & c_2^1 & \dots & c_K^1 \\ c_1^2 & c_2^2 & \dots & c_K^2 \\ \dots & \dots & \dots & \dots \\ c_1^G & c_2^G & \dots & c_K^G \end{Bmatrix}$$

where c_i^l represents sub-model i's reliable value on class l. We use $c_i^l = \frac{N_{i,l}}{N_l}$ to approximate sub-model i's reliable value on class l since a sub-model observes better performance on a class if it includes more training samples of this class. $N_{i,l}$ is the amount of samples with class l in sub-model i and N_l is the total amount of samples with class l. The detail of NCDWS is illustrated in Algorithm 2.

NCDWS achieves good performance by considering class distribution. However,

Algorithm 2: Naïve Class Distribution Weighted Sum (NCDWS)

Input: Predicted vector v_i $(1 \leq i \leq K)$ of the sub-models, $N_l(1 \leq l \leq G)$, $N_{i,l}(1 \leq i \leq K, 1 \leq l \leq G)$

Output: Final predicted vector v

```
Initialing v^l=0;
for class l ← 1 to G do
    for sub-model i ← 1 to K do
        v_i^l = c_i^l * v_i^l ;
        v^l = v^l + v_i^l ;
    end
end
return Final predicted vector v=(v^1,v^2,...,v^G);
```

it is not practical since it needs to know the distribution of training data in different local models in advance. This assumption is too strong and too demanding in practice since (1) it needs to test the distribution of training data and (2) this distribution may change after model unlearning. To overcome this problem, we introduce an enhanced class distribution weighted method (ECDWS). Unlike NCDWS, ECDWS does not need to know any class distribution information in advance.

Enhanced class distribution weighted sum

The core of ECDWS is the KL-attention recommendation mechanism, which we propose based on the concepts of KL-divergence [61] and self-attention [62,63]. The main idea of the KL-attention recommendation is to give higher weights to similar outputs of local models. For a test sample, each sub-model i outputs its prediction,

$$v_i = \begin{bmatrix} v_i^1 & v_i^2 & \dots & v_i^G \end{bmatrix}^T, \tag{2.5}$$

where v_i^l $(1 \leq i \leq K, 1 \leq l \leq G)$ denotes the probability that the sample is predicted as class l. To capture the global intent of predicted vectors and ignore unimportant information, we consider each sub-model's predicted vector as a candidate item in a KL-attention recommendation system. We calculate the similarity between two candidate vectors v_j and v_i $(1 \leq j, i \leq K)$. The similarity function $\mathcal{R}(j, i)$ is defined based on KL divergence:

$$\mathcal{R}(j, i) = e^{-\sum_{l=1}^{G} v_j^l \log \frac{v_j^l}{v_i^l}}. \tag{2.6}$$

We obtain a similarity vector $\mathcal{R}(j)$ for each candidate vector v_j, where $\mathcal{R}(j) = (\mathcal{R}(j,1), \mathcal{R}(j,2), \ldots, \mathcal{R}(j,K))$. Denote $\mathcal{R}(j)^{SUM}$ as the sum of all the elements in the similarity vector $\mathcal{R}(j)$, i.e., $\mathcal{R}(j)^{SUM} = \sum_{i=1}^{K} \mathcal{R}(j,i)$. We apply the softmax function to each element $\mathcal{R}(j,i)$ of the similarity vector, i.e., $\mathcal{R}(j)$=$(\sigma(\mathcal{R}(j,1)), \sigma(\mathcal{R}(j,2)), ..., \sigma(\mathcal{R}(j,K)))$, where $\sigma(\mathcal{R}(j,i)) = \frac{\mathcal{R}(j,i)}{\mathcal{R}(j)^{SUM}}$.

We then represent the candidate vector v_j as a weighted sum of all the candidate vectors:

$$v_j = \sum_{i=1}^{K} \sigma(\mathcal{R}(j,i)) v_i. \tag{2.7}$$

That is, the weight is determined by the similarity of candidate vectors. We repeat this process to update all the candidate vectors until convergence. The proof of the convergence of this process is given in Appendix. The detail of ECDWS is given in Algorithm 3.

Algorithm 3: Enhanced Class Distribution Weighted Sum (ECDWS)

Input: Predicted vector v_i ($\forall 1 \leq i \leq K$) of the sub-models
Output: Final predicted vector v

Initializing $R(j)^{SUM} = 0$;
repeat
  for *candidate vector* $j \leftarrow 1$ **to** K **do**
    for *candidate vector* $i \leftarrow 1$ **to** K **do**
      $\mathcal{R}(j,i) = e^{-\sum_l v_j^l \log \frac{v_j^l}{v_i^l}}$; $\mathcal{R}(j)^{SUM}$=$\mathcal{R}(j)^{SUM}$+$\mathcal{R}(j,i)$;
    end
    for *candidate vector* $i \leftarrow 1$ **to** K **do**
      $\sigma(\mathcal{R}(j,i)) = \frac{\mathcal{R}(j,i)}{\mathcal{R}(j)^{SUM}}$;
    end
    $v_j = \sum_i \sigma(\mathcal{R}(j,i)) v_i$;
  end
until *Convergence*;
$v = \frac{\sum_{j=1}^{K} v_j}{K}$;
return *Final predicted vector* v=$(v^1, v^2, ..., v^G)$;

Summary of Test Procedure

We input the test sample into the local models simultaneously and obtain a predicted vector from each of the sub-models. The i-th value in the vector denotes the

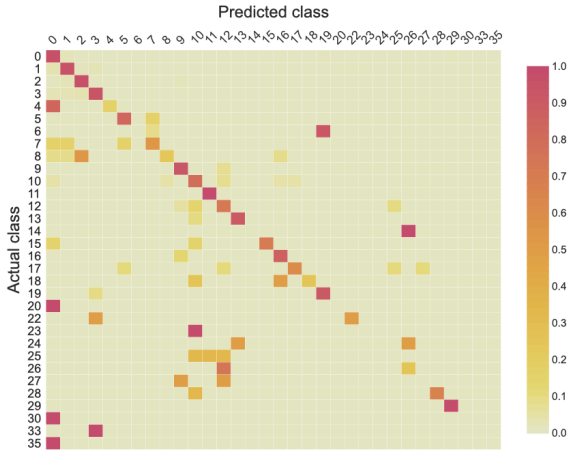
probability that the sample belongs to the i-th class. Then we apply the ECDWS algorithm to aggregate these predicted vectors to obtain the final vector. The sample is predicted as class i, where the i-th value in the final vector is the largest. If there is a tie (i.e., multiple classes have the same highest value), randomly assign a class from the tie.

2.3.4 Unlearning with *ViFLa*

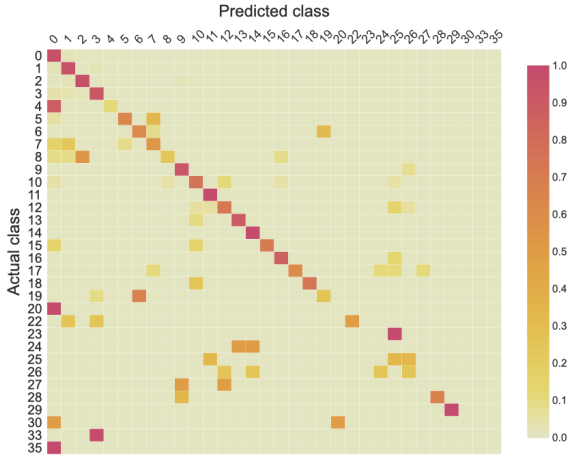
The unlearning process in *ViFLa* is straightforward. To unlearn some data, we update the summation form SQ answers A_{SQ_j} $(\forall 0 \leq j \leq B-1)$ in the learning process by removing the contribution of the unlearned batch $batch_b$ $(b \in U)$, where U denotes the set of the index of all the unlearned batches. A batch is considered as unlearned batch if its mapping result at any state in the state transition ring is affected by the unlearned samples. Then the convergent state θ_{S_B} stored in node n_0 in the learning process will degenerate to a previous state. We use this previous state as our new initial parameter state and learn a new convergent state. Different from the learning process, the unlearning process is very fast since the new initial state is close to a convergent state.

The main steps of the unlearning process are as follows:

- Step 1: Obtain the summation form SQ answer $A_{SQ_j}(\forall 0 \leq j \leq B-1)$ in the learning process.
- Step 2: Calculate the new SQ answers, denoted as $\hat{A}_{SQ_j}(\forall 0 \leq j \leq B-1)$, by removing the mapping results of the unlearned batches from the SQ answers obtained in the learning process, i.e., $\hat{A}_{SQ_j}=A_{SQ_j} - \sum_{b \in U} f_{\theta_{S_j}}(batch_b)$ $(\forall 0 \leq j \leq B-1)$.
- Step 3: Use the new summation form SQ answers $\hat{A}_{SQ_j}(\forall 0 \leq j \leq B-1)$ obtained in step 2 to update the parameter state. The convergent parameter state θ_{S_B} will degenerate to a previous state.
- Step 4: Set this previous state as new initial state.
- Step 5: Repeat the training process (Section 2.3.2) until convergence.



(a) The confusion matrix of test data after removing the contribution of 5% training samples of the model by naïve unlearning.



(b) The confusion matrix of test data after removing the contribution of 5% training samples of the model by *ViFLa*.

Figure 2.6: The confusion matrix of test data after removing the contribution of 5% training samples of the model by using different methods. The Kappa index between the two confusion matrices is 0.938. We also test the case of removing the contribution of 20% training samples of the model by naïve unlearning and *ViFLa* (figures omitted for brevity). The Kappa index between the two confusion matrices is 0.921.

2.4 *ViFLa* in Action: Machine Unlearning for IoT Anomaly Detection

2.4.1 How and When Should *ViFLa* Be Used?

Note that *ViFLa* itself is *not* an anomaly detection model. Instead, *ViFLa* provides a framework to train the detection model and more importantly to quickly update the trained model. It must work with an underlying anomaly detection model for traffic anomaly detection. Therefore, *ViFLa* should be deployed at the same place where the anomaly detection model is trained and used, e.g., the security gateways of an IoT system. *ViFLa* can be applied to an IoT network wherever multiple IoT devices are connected to the network.

The model update is a much-needed feature for anomaly detection in IoT. For instance, when some normal data samples used for model training are later identified as anomalies or when an IoT device upgrades its firmware and causes changes in traffic patterns, the detection model should forget the impact of obsolete data and accommodate the contribution of new data. We in the following evaluate the benefit of *ViFLa* with the underlying LSTM anomaly detection model (refer to Appendix) and real-world trace data.

2.4.2 Data Preprocessing

We use IoT-23 dataset [85] as well as our own DCS-932LB data. IoT-23 includes malicious and benign communication sequences of different IoT devices. We use benign packets of a Philips HUE smart LED lamp (21,664 benign packets) and Malware-Capture-34-1 (233,865 packets). DCS-932LB includes 28,867 benign packets and 18,559 Mirai malware packets of a smart camera. We use IoT-23 dataset as it is a commonly-used public dataset. Unfortunately, this dataset does not provide data for some experiments, e.g., IoT traffic data before and after device firmware update. Therefore, we build our own testbed [26] and collect a dataset to complete the experiments.

Anomaly detection can be built based on the communication patterns of IoT devices. Using the 7 features proposed in [1], we extract the characteristic patterns from the sequence of benign data packets, and then determine whether the characteristic pattern of a device is consistent with the patterns learned from benign sequences.

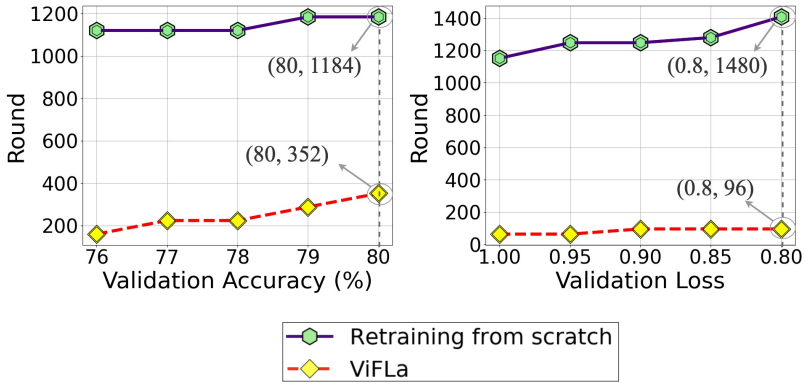
The seven features include traffic direction (incoming/outgoing), bin index of the local port number, bin index of the remote port number, bin index of packet length, TCP flags, protocol type, and bin index of packet inter-arrival time. Using the above 7 features, We extract a feature tuple $(a_1, \ldots, a_7)$ for each packet and then map each packet to a *packet type* based on its feature tuple. The total number of packet types is 38, i.e., the value of the feature tuple has 38 different combinations. As a result, the data packet sequence in the dataset will be converted to a sequence of packet types.

After that, we convert the packet type sequence to the actual input of *ViFLa* using a sliding window of size F. For any $m(> F)$, the packet type sequence $(h_{m-F}, h_{m-F+1}, \ldots, h_{m-1})$ of F preceding packets $PK_{m-F}, PK_{m-F+1}, \ldots, PK_{m-1}$ is used as the feature values $(x_1, x_2, \ldots, x_F)$ of the input sample (refer to Section 2.3.2), and the type of packet PK_m is used as the actual class value l of the input sample. These feature-class pairs $((x_1, x_2, \ldots, x_F), l)$ can be used as training/testing samples of *ViFLa*. We train a three-layer LSTM [50] model in the *ViFLa* framework for effective machine unlearning. Given any $m(> F)$ and the types of the F preceding packets $PK_{m-F}, PK_{m-F+1}, \ldots, PK_{m-1}$, the detection model can estimate the probability distribution that data packet PK_m belongs to different packet types. We set $F = 20$.

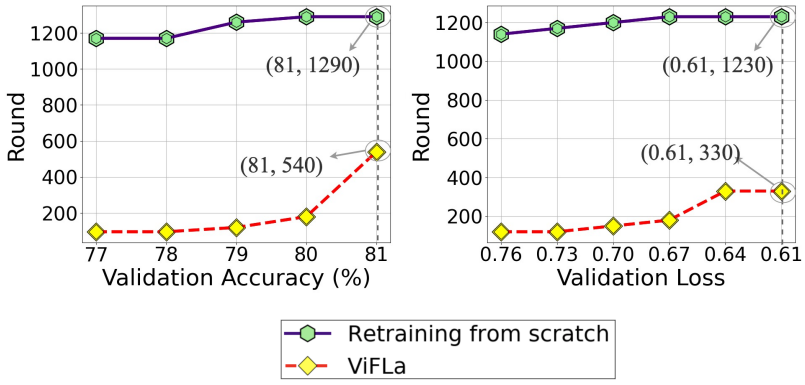
2.4.3 Completeness

To evaluate the completeness, we compare the results of *ViFLa* and the results of naïve unlearning. We use the confusion matrix to visualize the results, where each row represents the samples in an actual class, each column represents the samples in a predicted class, and the value at (i, j) denotes the probability that class i is predicted as class j.

As shown in Fig. 2.6, the confusion matrix obtained with *ViFLa* after removing the contribution of some samples and the confusion matrix obtained with a model retrained from scratch are very close. To quantitatively measure the difference between two confusion matrices, we also calculate Cohen's kappa coefficient [99] between the two confusion matrices, which is a well-known metric to assess the agreement between two classifiers. The kappa score is above 0.9. Note that a kappa score above 0.8 is generally considered good agreement [99]. The evaluation results demonstrate that ViFLa can achieve nearly identical performance of retraining from scratch.



(a) The speed comparison of two unlearning methods in training a new convergence when an IoT device (camera model: DCS-932LB) upgrades its firmware version from V2.16.08 to V2.17.01.



(b) The speed comparison of two unlearning methods in training a new convergence when an IoT device (camera model: DCS-932LB) upgrades its firmware version from V2.17.01 to V2.18.01.

Figure 2.7: The speed comparison of two unlearning methods in training a new convergence when an IoT device (camera model: DCS-932LB) upgrades its firmware version.

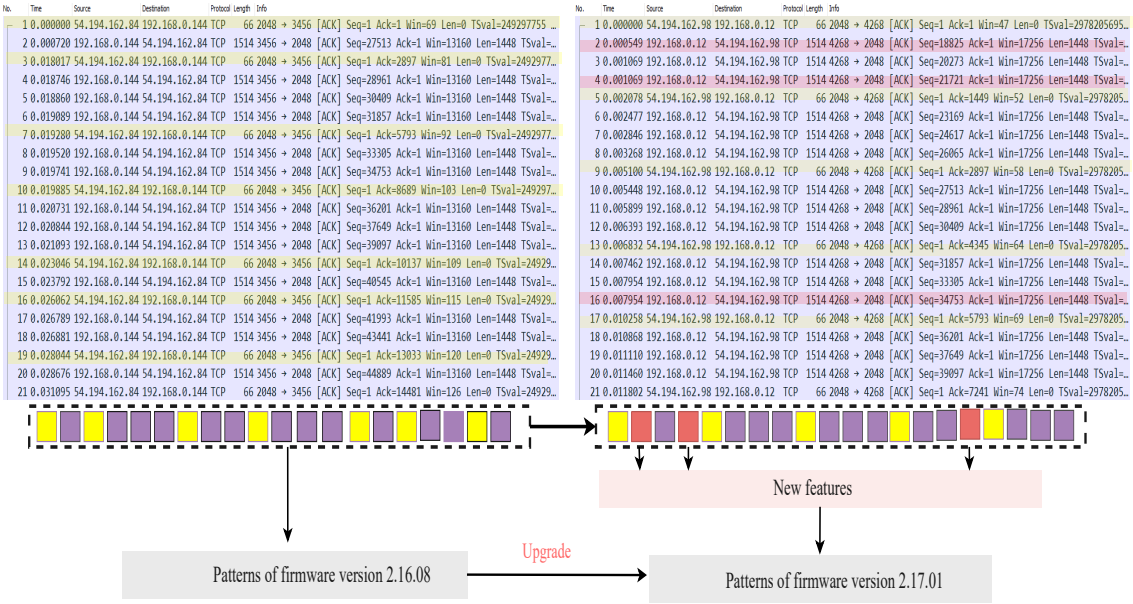


Figure 2.8: The traffic difference when the firmware of an IoT device (camera model: DCS-932LB) is upgraded. After the firmware update, the user captures the new traffic (right) of this device with Wireshark and submits a model update request to *ViFLa*, which automatically identifies the difference in the feature and updates the detection model accordingly. Different colors illustrate different packet types (Refer to Section 2.4.2 for the detail of determining the type of a packet). The colored boxes at the bottom of the packet window mean that for every packet, the sequence of 20 preceding packet types are used as the feature.

2.4.4 Efficiency

In the first test, we emulate the scenario where an IoT device upgrades firmware and thus accordingly needs to quickly upgrade the corresponding detection model. The IoT device we tested is a D-Link smart camera (model DCS-932LB). We upgraded its firmware from V2.16.08 to V2.17.01, and then from V2.17.01 to V2.18.01. We recorded its traffic before and after each upgrade. Following the feature extraction method of [1], we extract characteristic patterns from the data packet sequence of the old firmware version and the new firmware version of the IoT device, respectively. To quickly update the detection model after the firmware upgrade, we can use *ViFLa* to re-train the model. Note that we do not need to manually compare the data samples in the two firmware versions. Instead, we only need to replace the entire data sample of the old version with the entire data sample of the new version, and then use the parameters in the old version of the model as the initial weights to retrain a model for the new version. *ViFLa* can automatically identify the difference (based on the

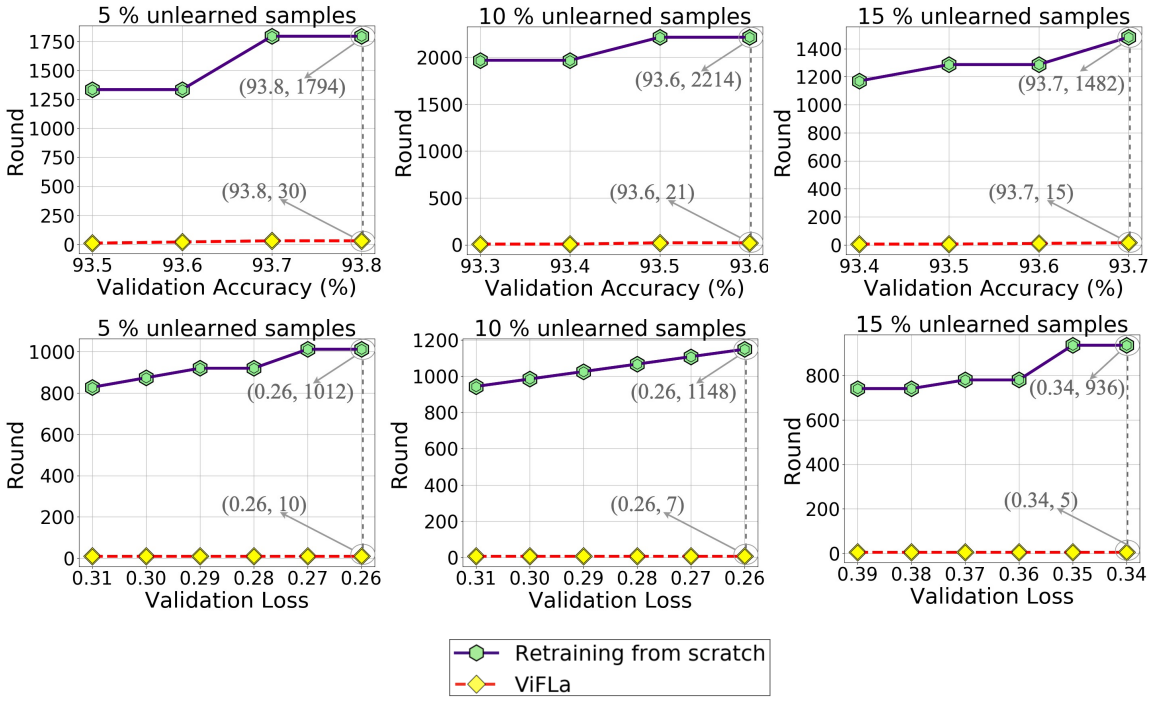


Figure 2.9: The speed comparison of two unlearning methods in training a new convergence state for remaining training data.

initial weights and the state transition ring) and update the model accordingly. As a comparison, we also retrain the model from scratch using all the new traffic. As shown in Fig. 2.7, *ViFLa* achieves a significant speedup compared to retraining from scratch when we upgrade the device. Each round of training means that we have completed the training of a batch of samples (the batch size is 128).

Fig. 2.8 illustrates the traffic difference before/after the camera's firmware is upgraded, from which we can see that firmware upgrades only causes small changes in traffic patterns of the IoT device. The characteristic patterns in the new version are similar to the characteristic patterns learned from the old version. Benefiting from this and with the help of the state transition ring, the convergence state of the old version will be close to the convergence state of the new version. This explains why *ViFLa* speeds up retraining from scratch.

In the second test, we randomly select a fraction of training samples with probability following their unlearning belief values. To unlearning the selected samples, we re-train the model via *ViFLa* and via naïve unlearning, respectively. The training rounds required for different validation accuracy and average validation loss of

Table 2.2: Ablation study of components in *ViFLa*

Smart partition	State transition ring	ECDWS	Test accuracy improvement*		Unlearning speed improvement*	
			iid	non-iid	iid	non-iid
✓	✓	✓	**2% - 3%↑**	**27% - 28%↑**	**≈100x**	**≈120x**
✗	✓	✓	2% - 3%↑	27% - 28%↑	≈25x	≈30x
✓	✗	✓	2% - 3%↑	27% - 28%↑	≈4x	≈4x

* The improvement is over a modification of *ViFLa* that uses random data partition and weighted average local accuracy (WALAcc), and disables state transition ring.

remaining data in *ViFLa* and the training rounds using naïve unlearning is shown in Fig. 2.9. After unlearning training samples with the highest 5% belief values, we can see that *ViFLa* needs 30 rounds to obtain 93.8% validation accuracy of the remaining data, while retraining from scratch requires 1794 rounds. In other words, *ViFLa* can achieve up to 60X speedup compared to naïve unlearning.

Note that we compare our proposed method with naïve unlearning for two main considerations. First, most of the existing methods have already reported their improvements over naïve unlearning. Readers can compare different methods more objectively by seeing how much improvement each method achieves using naïve unlearning as the baseline. Second, due to the lack of experimental details, it is difficult to fairly compare our method with the existing methods directly. For instance, [95] presents a general method that may have different implementation details.

2.4.5 Ablation Study: Compared to Other Methods

ViFLa is quite different from existing machine unlearning methods, such as SISA [64], [60] and [53], in that *ViFLa* includes new mechanisms, such as smart partition, ECDWS, and state transition ring. In addition, the underlying machine learning models of *ViFLa* is different from that in [53, 64]. Due to this reason, it would be difficult to obtain fair comparison between *ViFLa* and existing solutions. Nevertheless, we can remove some mechanisms in *ViFLa* to gain insight on the advantages of *ViFLa* over others, because the performance of *ViFLa* with ablated function roughly

Table 2.3: Performance of different aggregation methods before and after unlearning mislabelled samples.

Aggregation approach	Accuracy of iid data (%)		Accuracy of non-iid data (%)	
	learning	unlearning	learning	unlearning
WALAcc	86.97	87.32	59.99	59.8
NCDWS	89.54	89.92	83.69	83.99
ECDWS*	89.79	90.02	87.47	87.65

* ECDWS achieves better performance without even knowing the distribution of training data.

reflects that of existing solutions. In the ablation experiment, we also use IoT-23 data.

Smart Partition

To evaluate the benefit of smart partition, we randomly selected some training samples and mislabeled their classes (i.e., modified the actual class to a different class). Then we use k-fold ($k = 4$) cross-validation [100–104] to calculate the unlearning belief value for each sample. Each sample is used once in the test set and used to train the naïve Bayes model 3 times. As a result, we can predict the probability of each sample belonging to its labeled class. Then, we average the predicted probability of all the mislabeled samples, and further estimate their average unlearning belief value (the threshold $\epsilon = 0.9$). We found that mislabeled samples would gain larger unlearning belief values. We then group (the group number is 4) training samples according to their unlearning belief. The test result is then compared with that obtained from randomly group training samples. The benefit of smart partition is reflected at the first and third rows of Table 2.2 under different combinations. The smart partition-based method only needs to update one single group to forget the mislabelled data, while the random group method needs to update all four groups.

Remark 1. *The good performance of smart partition comes from the capability of the naïve Bayes model in quickly learning the patterns in training samples. This does not necessarily mean that the naïve Bayes model is a good candidate for IoT anomaly*

detection. The purpose of smart partition is to quickly estimate the chance that a sample might need to be unlearned in the future, and thus achieving highly accurate estimation, while helpful, is not the goal.

State Transition Ring

To evaluate the benefit of a state transition ring for different LSTM structures, we test the unlearning performance with and without a state transition ring. The benefit of the state transition ring is reflected in the first and second rows of Table 2.2 under different combinations.

NCDWS vs. ECDWS

Two types of baselines might be considered. One is a parameter-based aggregation method, such as FedAvg [46] that learns a global model by aggregating parameters of local models. The other is weighted average local accuracy (**WALAcc**) that aggregates the predicted accuracy of sub-models based on the number of training samples used to train the sub-models. WALAcc improves a method in [105] by putting weights on local accuracy. We omit FedAvg because (1) the final (global) model is coupled with all local models, making FedAvg not suitable for machine unlearning, and (2) FedAvg performs worse than WALAcc in our test.

We set the number of sub-models as 4, and test the performance of NCDWS and ECDWS. The class distribution of the data samples generated from the IoT-23 dataset benign traffic is uniform. To evaluate the performance of different aggregation methods on iid data, we evenly distribute the training data to the four partitions. To evaluate the performance of non-iid data, we distribute some classes of data samples only to one or two partitions. From the results in Table 2.3, we can see that both NCDWS and ECDWS achieve significant improvements over WALAcc in accuracy for non-iid data. In addition, ECDWS shows clear advantages over NCDWS because ECDWS can achieve similar accuracy without even knowing the distribution of training data.

The above accuracy results only refer to the probability that a sample is correctly predicted. They do not directly translate to the anomaly detection result, which is disclosed below.

2.4.6 Performance of Anomaly Detection

A common strategy to test anomaly detection accuracy is to treat a sample as abnormal if its accuracy is lower than a threshold, and then trigger an anomaly alarm if a given percentage of samples in a sliding window are abnormal. We use false positive rate (FPR) and true positive rate (TPR) to evaluate anomaly detection results. The 20% of benign traffic generated by the Philips HUE smart LED lamp is used to test the FPR, and malware traffic is used to test the TPR.

Since *ViFLa* is for model update and must work with an underlying detection method for anomaly detection, we used LSTM as the detection model for the test. The output of each sample in *ViFLa* is a predicted vector $v = v^1, \ldots, v^G$, where v^l denotes the probability that the sample is predicted as class $l(1 \leq l \leq G)$ based on its feature value $x_1, x_2, \ldots, x_F$. Then we take the predicted probability of the actual class of the sample. We choose an anomaly detection threshold of 0.01, and a sample is considered as abnormal if the predicted probability of its actual class l is lower than the detection threshold. If 50% of samples in a sliding window of size of 30 samples are abnormal, an anomaly alarm is triggered. As a result, we got the performance of 100% TPR and no false positives. This excellent performance is no surprise because the detection model is type-based (i.e., different IoT device types use their corresponding detection model), which has shown excellent detection accuracy as reported in previous work [1].

2.5 Efficiency Analysis

While it is hard to have a fair comparison between *ViFLa* and existing methods as discussed in Section 2.4.5, we can perform a theoretical analysis on the efficiency of *ViFLa* to offer more insights on the difference between *ViFLa* and existing methods. For this, we use SISA as the baseline since its components show a similarity to those in *ViFLa*. As a rough analogy, its sharding method corresponds to our smart partition (i.e., a shard in SISA corresponds to a virtual client in *ViFLa*), and its slicing method corresponds to our state transition ring. The total samples of all the groups are represented as N.

First, we compare the time cost of sharding in SISA with the time cost of smart partition in *ViFLa*. In sharding, samples are randomly divided into different shards. Thus, the probability that a sample to be unlearned falls on shard k is $\frac{1}{K}$, where K

is the number of shards. The expected cost of an unlearning sample is the average number of points to be retrained caused by this unlearning sample, and the expected total cost is the sum of the expected costs of all unlearned samples [106]. Then, the expected total cost of all the unlearning samples is

$$E_1 = \sum_{i=1}^{M} \sum_{j=0}^{i-1} \binom{i-1}{j} (\frac{1}{K})^j (1 - \frac{1}{K})^{i-1-j} (\frac{N}{K} - j - 1) \tag{2.8}$$

where M is the total number of samples to be unlearned, and $\frac{N}{K}$ is the number of samples in each shard. To help understand (2.8), for any given ith unlearning sample, j means the number of previous unlearning samples that fall in the same shard as i. When i is fixed, a greater j means more samples land on the same group.

In contrast, the smart partition in *ViFLa* groups samples of high unlearning beliefs together and builds a local model with these samples. When an unlearning request arrives, the samples included in the request may be relevant to only one or very few local models. The majority of local models do not need to be re-trained. We sort the unlearned samples by their belief values from high to low and divide them into different groups. Therefore, an unlearned sample has a higher probability of falling on the kth group than the $(k+1)$th group $(1 \leq k < K)$, where K is the number of virtual clients. Assume that an unlearned sample falls on the group $(1, 2, \ldots, K)$ with probability $(P_1, P_2, \ldots, P_K)$, respectively, where $\sum_{k=1}^{K} P_k = 1$. Then the expected total cost of all the M unlearned samples is

$$E_2 = \sum_{i=1}^{M} \sum_{j=0}^{i-1} \binom{i-1}{j} \sum_{k=1}^{K} P_k (P_k)^j (1 - P_k)^{i-1-j} (\frac{N}{K} - j - 1). \tag{2.9}$$

While it is difficult to rigorously prove that $E_2 \leq E_1$, we run numerical simulation with randomly generated M, N, K, and $P_1, P_2, \ldots, P_K$ values a million times and found that $E_2 < E_1$ in all the tests; $E_2 = E_1$ only when $P_1 = P_2 = \ldots = P_K$.

Second, we compare the time cost of slicing in SISA and the time cost of state transition ring in *ViFLa*. The total cost of slicing in SISA is $e^{'} \frac{N}{K} (\frac{2}{3} + \frac{1}{3R})$, where $e^{'}$ is the number of epochs without slicing and R is the number of slices in the retrained model (refer to [106] for more details). When R is large enough, we obtain the lower bound $\frac{2}{3} \frac{N}{K} e^{'}$, where N is the total number of samples and $\frac{N}{K}$ is the number of samples in each shard. In the state transition ring, the total cost is $\hat{e} \frac{N}{BK} = \frac{\hat{e}}{B} \frac{N}{K}$, where $\hat{e}$ is the number of iterations in state transition ring. In our experiment, the value (normally

< 5) of $\frac{\hat{e}}{B}$ is much smaller than $\frac{2}{3}e'$ since the new initial state is close to the convergent state.

Third, we compare the storage cost between SISA and *ViFLa*. SISA assumes that there are R slices in each shard k. First, SISA trains the model using the first slice $D_{k,1}$ and saves the checkpoint as $M_{k,1}$. Then it trains the model using the first two slices $D_{k,1} \cup D_{k,2}$ and saves the checkpoint as $M_{k,2}$. Hence, it needs to store R model states (checkpoints) for each shard. In its evaluation [106], the slice size is set to 1 since this value needs to be small enough for better unlearning performance. Nevertheless, a very small slice size will make the number of slices R in each shard close to the number of samples $\frac{N}{K}$ in each shard. So SISA needs to store $\frac{N}{K}$ model states for each shard. Overall, the total number of model states that need to be stored in SISA is N since there are K shards.

In contrast, for each group k in *ViFLa*, we just need to store a state for each node and there are B (B is the number of batches in each group) nodes in the state transition ring. Therefore, our method needs to store B model states for each group. Since we have K groups, the total number of states that need to be stored in *ViFLa* is BK. Clearly, $BK < N$ because $N = BKT$ where $T(> 1)$ is the batch size. In conclusion, *ViFLa* incurs a smaller storage cost than SISA.

2.6 Conclusion

Machine unlearning, a technique that quickly updates a machine learning model and removes the impact of a small portion of training data on the model, is much needed in IoT traffic anomaly detection. The requests for machine unlearning may come from different scenarios, e.g., some training data may be mislabeled due to unknown attacks or an IoT device upgrades its firmware and thus changes partial data that have been used in the model training. We proposed a new solution, *ViFLa*, that leverages the concept of virtual federated learning and consists of three new mechanisms, smart partition, enhanced class distribution weighted sum (ECDWS), and state transition ring, to achieve efficiency and completeness of machine unlearning. Using real-world trace data, we thoroughly evaluate the performance of *ViFLa*, covering not only the effectiveness of its individual components but also its benefit in different application scenarios. Overall, *ViFLa* can achieve similar accuracy of re-training from scratch with significant speedup. We also performed a theoretical analysis of the efficiency of *ViFLa* using SISA as a baseline. Compared to the baseline, *ViFLa* can reduce the

computational complexity and the amount of data affected by unlearned samples.

Chapter 3

Taking Advantage of Mistakes: Clustered Federated Learning for IoT Anomaly Detection

3.1 Introduction

IoT anomaly detection has attracted significant attention due to the booming market [107] and the lack of built-in security controls [108] of IoT devices. In order to utilize large-scale data from IoT systems without revealing user privacy [109], the federated learning (FL) based IoT anomaly detection system has been proposed [1]. Fig. 3.1 shows the architecture, where each client refers to an autonomously managed IoT system, e.g., a small office/home office (SOHO), and the server refers to a computing entity trusted by all the clients. Note that to improve accuracy, the IoT anomaly detection is usually type-based [1], in which the type of IoT devices are identified [81] and the FL procedure is performed for the same type of devices. The effectiveness of this scheme is based on the assumptions that the traffic patterns in IoT devices can be modeled and the same type of IoT devices have similar and regular traffic patterns [1].

Nevertheless, the data from different clients (e.g., IoT devices) may not be independent and identically distributed (i.e., non-IID). As a result, the global model may perform poorly, as the local models can negatively affect each other when forced to aggregate [54, 58, 59]. Consequently, a single global model cannot satisfy the needs of all clients when dealing with non-IID data. *clustered federated learning*

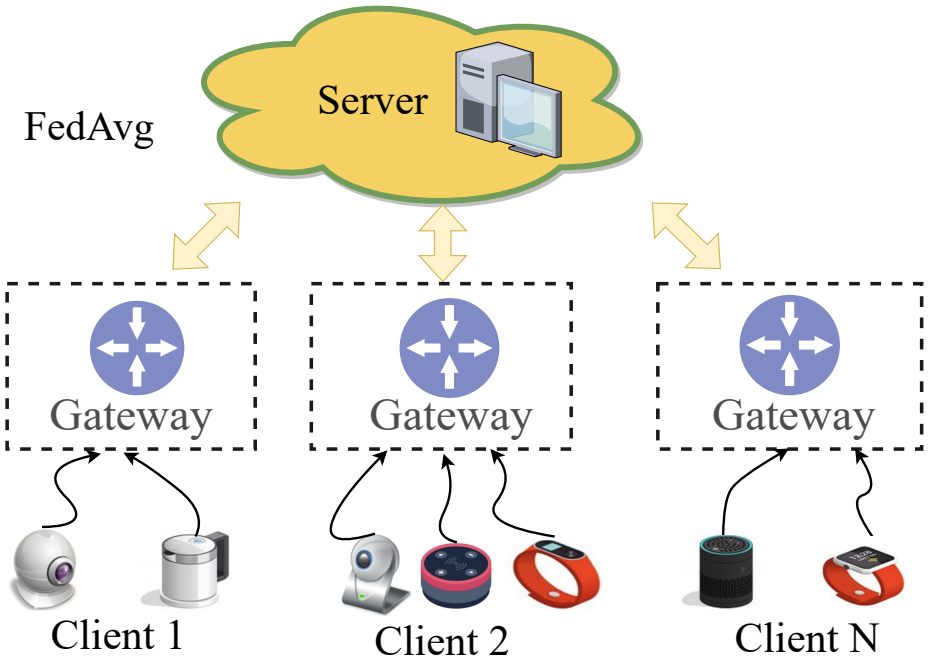


Figure 3.1: The federated learning-based IoT anomaly detection system.

(CFL) [59, 66–69] has been offered as a solution to this problem. Instead of building one global model [54–57], multiple global models that can adapt to different clusters are trained by CFL by grouping clients with similar data distributions into the same cluster. It shows that CFL can effectively solve the negative impact caused by aggregating local models trained on non-IID data.

3.1.1 Observations & Motivations

When applying existing CFL solutions for IoT anomaly detection, however, we have the following findings and problems.

- The non-IID problem in the IoT network exists not only in the spatial domain (i.e., data from different clients are non-IID) but also in the temporal domain (i.e, the training and test data collected in sequence are non-IID to each other). Based on our experiments in a real-world IoT testbed (refer to Section 3.5.1 for details), the test data and training data from the same environment can also be non-IID, due to i) the variation of network configuration and condition and ii) duplicate TCP packet transmissions caused by link failure and/or network congestion.

- IoT anomaly detection systems usually have high requirements for the time delay. For example, firmware updates of some IoT devices are quite frequent, in which the global model for anomaly detection also needs to be re-trained promptly. A long time delay in such updates may lead to the temporary unavailability of services, which can cause severe impacts on clients.

In consequence, existing CFL solutions cannot adapt well to the IoT anomaly detection task. The non-IID problem of IoT systems in the temporal domain could degrade the accuracy of CFL, since the test data (of each cluster) may be quite different from the training data where the global model is obtained (in that cluster). Thus, the first question we need to answer is: *how can we increase the precision of IoT anomaly detection by addressing the spatial-temporal non-IID issue under the existing CFL scheme?* Additionally, the current clustering procedure in existing CFL solutions is in need of multiple iterations [59,67,70], which is time-consuming and may cause an unacceptable time delay in IoT anomaly detection and corresponding service. Thus, we also need to answer: *how can we re-design the clustering procedure in the current CFL to improve its time efficiency while ensuring the clustering performance?*

3.1.2 Challenges and Contributions

To answer the above two questions for more accurate and efficient anomaly detection in IoT systems/applications, we are faced with two major challenges. First, with newly collected test data from an IoT system, the distributional difference between it and the training data cannot be easily measured. This difference may change over time in IoT systems: for one period, the test data may be very similar to the training data, while for another period, they may not look like each other at all due to network congestion or configuration changes. Second, model filtering (i.e., weeding out useless noisy parameters) can be used to slim the cluster models and speed up the clustering process, while how to discard noisy parameters and leave useful ones is non-trivial due to the uninterpretability of model parameters (obtained from neural networks).

In this work, we contribute the following in order to address the aforementioned challenges.

- We present *ClusterFLADS*, a new clustered federated learning approach to IoT anomaly detection. To address the non-IID problem in both spatial and temporal domains, we take advantage of the false predictions of the inappropriate global models, together with knowledge of temperature scaling [71, 72] and

catastrophic forgetting [73, 74] to reveal distributional similarities between the training data (of different clusters) and the test data.

- We design an efficient feature extraction scheme by exploiting the difference in the role each layer of a neural network plays in the learning process. By carefully selecting model parameters and conducting dimensionality reduction, our approach can effectively improve the clustering speed and thus reduce the anomaly detection delay.
- We extensively evaluate the performance of *ClusterFLADS* using real-world IoT data. Compared to the baselines, our experimental results disclose that *ClusterFLADS* can accurately and efficiently cluster diverse clients and achieve better anomaly detection results over various data distributions.

The rest of the paper is organized as follows. We present the architecture and training procedure of *ClusterFLADS* in Section 3.3, the re-designed testing procedure of *ClusterFLADS* in Section 3.4, and performance evaluations in Section 3.5, respectively. Section 3.2 introduces the related work. The paper is concluded in Section 3.6.

3.2 Related work

As shown in Fig. 3.2, the clustered federated learning methods can be roughly divided into two categories: i) those without knowledge sharing among clusters and ii) those with knowledge sharing among clusters.

Solutions without Knowledge Sharing among Clusters

Solutions in this category do not share knowledge (e.g., model parameters) between clusters. According to the number of clustering iterations, solutions in this category will be further split into non-iterative clustering and iterative clustering. Liu et al. [68] proposed a non-iterative clustering framework called PFA that enables privacy-preserving federated adaptation. They exploit neural networks' sparsity to obtain privacy-protected representations and use them to divide the clients into different clusters. Nevertheless, this framework is primarily designed for computer vision and uses convolutional neural networks (CNNs). Sattler et al. [67] designed a novel framework called clustered federated learning (CFL). They recursively partition

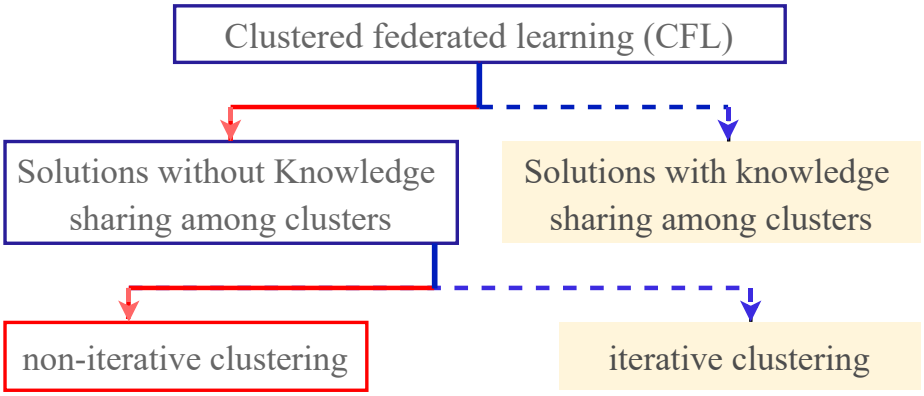


Figure 3.2: Classification of clustered federated learning approaches.

clients into various clusters in a top-down fashion: clients use their local data and the model update values from the cluster to update their local models. The server computes the cosine similarity of client-transmitted updates and divides clients into different clusters by minimizing the maximum similarity between clients of different clusters. Model updates for each cluster are obtained by averaging the updates of all clients belonging to that cluster. Such an iterative clustering method, however, is time-consuming and does not allow clients to learn knowledge in different clusters.

Solutions with Knowledge Sharing among Clusters

Solutions in this category share knowledge between clusters. Ghosh et al. [59] designed a new clustered federated learning framework called Iterative Federated Clustering Algorithm (IFCA). The IFCA algorithm alternates between minimizing a loss function and estimating cluster identities in a distributed setting. They also utilize the knowledge-sharing strategy in multitasking [70] for cluster model training. A distributed setup reduces the computational cost of the server but increases the communication overhead. Furthermore, the multi-tasking sharing strategy helps to train a more robust model for each task, mainly by sharing the first few layers of all models. However, task-specific knowledge is mostly contained in the last layer [59, 110–113]. Although the training effect has improved, the sharing of cluster-specific knowledge still needs to be improved.

This work belongs to non-iterative clustering in the first category. In contrast to previous work, we are the first to apply clustered federated learning to tackle the problem of IoT anomaly detection. Technologically, our approach is able to i) address

the non-IID problems in both the spatial domain and the temporal domain (rather than the IID data assumption between training and test data from the same client as in previous work) and ii) determine the global model during testing in a dynamic way (rather than using a fixed one). Furthermore, the strategic feature extraction of our approach makes the clustering process more efficient than those based on complete models [66].

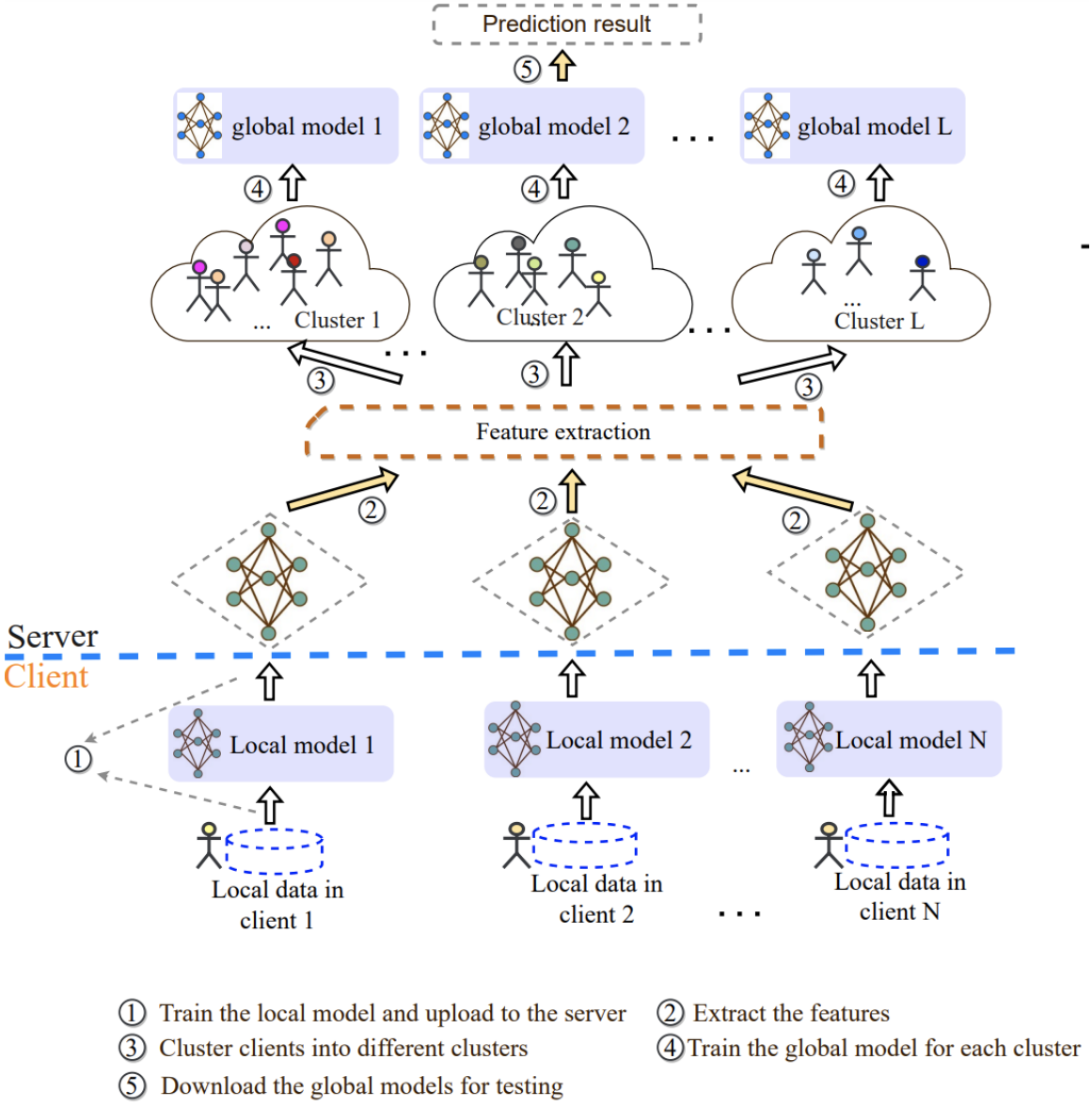


Figure 3.3: The architecture and workflow of clustered federated learning, in which five major steps are involved. Note that the yellow arrows indicate the improved or redesigned steps in this work.

3.3 CFL Architecture and ClusterFLADS Training

In this section, we first make an overview of the existing CFL approach and its typical workflow. Then, we elaborate on the training process of CFL, in which feature extraction is refined to improve the clustering efficiency.

3.3.1 Architecture and Workflow

An overview of the CFL architecture and workflow is illustrated in Fig. 3.3, in which five major steps are involved.

1. Each client trains an initial local anomaly detection model and uploads it to the server (① in Fig. 3.3).
2. The server extracts features from the uploaded local models (② in Fig. 3.3).
3. The server uses the clustering method to divide the diverse clients into different clusters based on the extracted features (③ in Fig. 3.3).
4. The server train a global anomaly detection model for each cluster (④ in Fig. 3.3).
5. The test data are collected from each client and tested using the corresponding global model (⑤ in Fig. 3.3).

For *ClusterFLADS* proposed in this work, we refine and redesign the CFL procedures to tackle the aforementioned problems in Section 3.1 and boost its performance of IoT anomaly detection. Specifically, we first refine the feature extraction process in the original training phase (refer to Section 3.3.2), and then completely redesign the testing scheme (refer to Section 3.4). For ease of reference, the main notations used in this paper are listed in Table 3.1.

3.3.2 Training Procedure of *ClusterFLADS*

We adopt a three-layer LSTM [50] for both global and local detection models. The detailed hyperparameters of the LSTM are given later in Section 3.5.1. Overall, the training procedure of *ClusterFLADS* involves four steps.

Table 3.1: Main notations

Notation	*Definition*
H	The number of packets in the sequence
N	The number of local clients
M	The number of neurons in the last layer of the model
U	The last layer of the model
$\theta_j^U(i,1)$	parameters in model j that related to the connection between the unit i in the last layer U and the unit 1 in the output layer
Z	The dimension of extracted client-specific features
f_j	The extracted client-specific features of client j
L	The number of clusters
$\theta_{global,l}$	The global model for cluster l
D_j	The privacy data of local client j
E	The number of local epochs in each communication round
$A_{global,l}$	The validation set's accuracy of sub-model $\theta_{global,l}$
$T_{global,l}$	The validation set's temperature scaling value of sub-model $\theta_{global,l}$
B	The number of batches in testing set
R	The batch size
$\hat{c}_{b,r}^l$	The global sub-model l's predicted label of the rth sample in testing batch b
$(x_{b,r}, \hat{c}_{b,r}^l)$	The feature-label pair reconstructed by the global sub-model l for the rth sample in the test batch b
$\theta_{update,l}^b$	The updated model of the global sub-model $\theta_{global,l}$
$A_{update,l}$	The validation set's accuracy in the update model $\theta_{update,l}^b$
F_b^l	The forgotten value of the test batch b on the global sub-model $\theta_{global,l}$
β	The comparison threshold
tf_b^l	The calibrated forgetting value of the batch b on global sub-model $\theta_{global,l}$
ϵ	The anomaly threshold
γ	The trigger threshold

*The upper part of notations is for training; the bottom part is for testing.

Step 1

Each client trains an initial local anomaly detection model and uploads it to the server. We take FedAvg [46, 114] as the aggregation method and perform $\hat{T}$ rounds of training.[1] In each round, the aggregated global model is denoted as $\theta_{global}^{(\hat{T})}$ and is sent back to all clients. Then each client j $(1 \leq j \leq N)$ updates $\theta_{global}^{(\hat{T})}$ with its private data and uploads the updated model to the server. After $\hat{T}$ rounds of training, the uploaded models from the clients are treated as initial local anomaly detection models, which are used to extract client-specific features.

Step 2

The server extracts client-specific features from the uploaded local model parameters. For the model parameters of client j, the server selects the parameters for the last layer θ_j^U, denoted as:

$$\begin{Bmatrix} \theta_j^U(1,1) & \theta_j^U(2,1) & ... & \theta_j^U(M,1) \\ \theta_j^U(1,2) & \theta_j^U(2,2) & ... & \theta_j^U(M,2) \\ ... & ... & ... & ... \\ \theta_j^U(1,\hat{R}) & \theta_j^U(2,\hat{R}) & ... & \theta_j^U(M,\hat{R}) \end{Bmatrix}$$

where M is the number of neurons in the last layer U, $\hat{R}$ is the number of output classes, and $\theta_j^U(i,r)$ $(1 \leq j \leq N, 1 \leq i \leq M, 1 \leq r \leq \hat{R})$ denotes the parameters in client j's local model related to the connection between the unit i in the last layer U and the unit r in the output layer. We then perform principal component analysis (PCA) [115] on the filtered residual parameters to reduce feature dimension and noise. For each client j, we get client-specific features:

$$f_j = \begin{bmatrix} f_{j,1} & f_{j,2} & \dots & f_{j,Z} \end{bmatrix}^T, \tag{3.1}$$

where $f_{j,z}$ $(1 \leq j \leq N, 1 \leq z \leq Z)$ is the zth features of client j and Z is the number of features after PCA dimensionality reduction [115]. These features are fed into the k-means [40] method for clustering.

[1] Our later experiments show that a small $\hat{T}$ is sufficient for client-specific feature extraction

Step 3

The server uses k-means to divide local clients into L clusters based on their client-specific features obtained in Step 2. The value of L is determined by the Elbow method [2]. By doing this, clients with similar data distribution are clustered together.

Step 4

For each cluster, the server uses FedAvg to train a global anomaly detection model. The convergence state is determined by the validation set. We stop training when the validation accuracy does not improve much. After this, we obtain a list of L global models,

$$\theta_{global} = \Big[\theta_{global,1} \quad \theta_{global,2} \quad \dots \quad \theta_{global,L})\Big]^T, \tag{3.2}$$

where $\theta_{global,l}$ $(1 \leq l \leq L)$ denotes the global model for cluster l. Finally, the server sends the list of L global models to all clients.

Note that in *Stop 4*, compared to the original CFL training process, we filter out all but the last layer parameters. Such an operation is based on the following considerations. The first few layers of the model usually contain more general information, while the latter layers contain more task-specific knowledge. The PCA [115] method can further extract the key information and reduce noise. Benefiting from the combination of parameter filtering and PCA [115], we successfully avoid the negative impact of permutation invariance of hidden layers [105]. In our later evaluation (in Section 3.5.2), we provide more evidence and analysis to explain why our feature selection can work well.

3.4 Testing Design of *ClusterFLADS*

In this section, we aim to redesign the testing scheme of the original CFL for adaption of IoT anomaly detection. Specifically, we first give the main ideas and rationale of the new design and then elaborate on each step of the testing procedure.

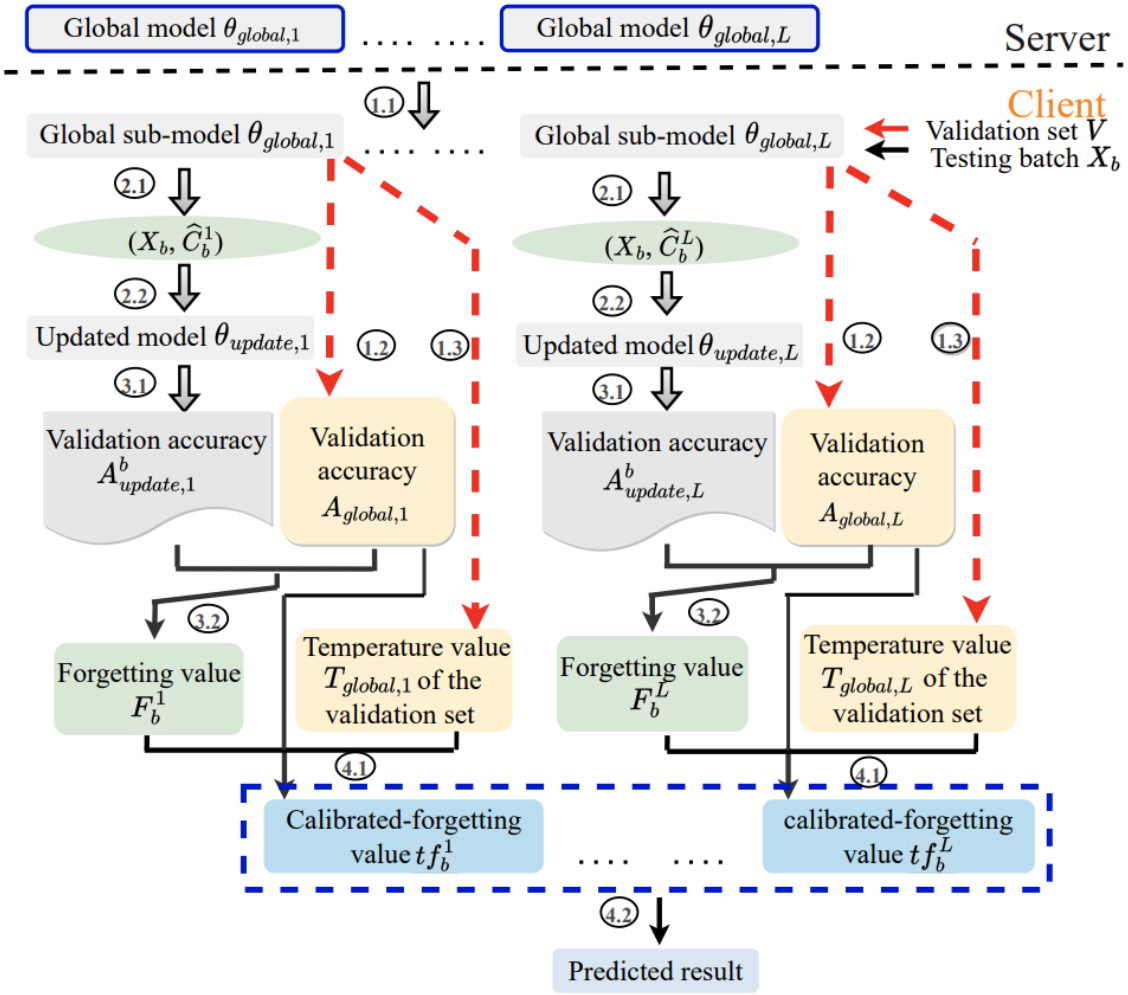


Figure 3.4: The testing procedure of *ClusterFLADS*, which includes four steps: 1) obtain the accuracy and temperature scaling values of the validation set on different sub-models; 2) build dummy feature-label pairs to fine-tune the sub-models and obtain the updated models; 3) compute forgetting values of different sub-models; 4) calculate the calibrated forgetting value of different sub-models and get the final prediction result.

3.4.1 Design Rationale

Key idea 1: use false predictions to reveal the degree of the distribution difference between test and training data

Here we leverage *catastrophic forgetting* [73, 74], which is one phenomenon showing that the neural network forgets what it has learned from old tasks while learning a new task. This phenomenon usually occurs in situations where the training data for several tasks are non-IID. Specifically in our anomaly detection task, given a batch of test samples, each global model makes its prediction. We thus can construct dummy pairs with the features of the test samples and their labels predicted by each global model. Then the formed dummy pairs are used to update their corresponding global models. The global models that did not show catastrophic forgetting after updating (e.g., validation accuracy reduction is smaller than a threshold) are selected as the optimal model for anomaly detection.

Key idea 2: calibrate forgetting value with temperature scaling

In order to solve the situation where catastrophic forgetting occurs in multiple models, we leverage *temperature scaling* to calibrate the forgetting value. Temperature scaling is a technique designed for calibration of prediction confidence [71]. A calibrated confidence means that the predicted class's probability closely reflects its ground truth correctness likelihood. The output logits in the neural network model are passed through the softmax function to obtain the class probability distribution. Temperature scaling fixes the model parameters and learns a temperature value T using a validation set. Then it simply divides the logits by the learned scalar parameter T. Then, if the model is less-confident, we can set a smaller temperature scaling value ($T < 1$) to calibrate the model; otherwise, we set a larger one ($T > 1$) for calibration. A temperature scaling value close to 1 implies that only a minor calibration is necessary for the confidence value.

In summary, if the global model does not observe catastrophic forgetting, the dummy pairs that the global model generates are likely to follow a distribution similar to the global model's training data distribution. Thus, global models in which catastrophic forgetting [73, 74] was not observed had a better predictive ability on the test data. If more than one such global model exists, we choose the global model with a higher validation accuracy and a smaller temperature scaling value. This is because a larger validation accuracy means that a change in precision has less effect on the

overall accuracy, and a smaller temperature scaling value implies that the problem of catastrophic forgetting is less likely caused by model overconfidence, i.e., the model has a better match between the predicted class's probability and its ground truth correctness likelihood.

3.4.2 Testing Procedure

Following the above analysis, we redesign the testing procedure with four steps.

Step 1 (1.1,1.2 and 1.3 in Fig. 3.4)

First, we test the accuracy $A_{global,l}$ $(1 \leq l \leq L)$ and temperature scaling values $T_{global,l}$ $(1 \leq l \leq L)$ of the validation set on all sub-models $\theta_{global,l}$ $(1 \leq l \leq L)$. Details of the method for calculating accuracy and temperature values are described in Alg. 5 (Line 1-Line 7).

Step 2 (2.1 and 2.2 in Fig. 3.4)

In order to obtain more accurate test performance, we divide the test samples into small batches and make predictions in batch units. Assume that we have a batch of testing data,

$$X_b = \begin{bmatrix} X_{b,1} & X_{b,2} & \dots & X_{b,R} \end{bmatrix}^T, \tag{3.3}$$

where b $(1 \leq b \leq B)$ is the batch index and R is the batch size.

Each sub-model will output its predicted labels,

$$\hat{C}_b^l = \begin{bmatrix} \hat{c}_{b,1}^l & \hat{c}_{b,2}^l & \dots & \hat{c}_{b,R}^l \end{bmatrix}^T, \tag{3.4}$$

where $\hat{c}_{b,r}^l$ $(1 \leq b \leq B,\ 1 \leq r \leq R)$ denotes the sub-model $\theta_{global,l}$'s predicted label of the rth sample in X_b.

Based on the features and predicted labels we construct our dummy feature-label pairs,

$$S_b^l = \begin{bmatrix} (x_{b,1}, \hat{c}_{b,1}^l) & (x_{b,2}, \hat{c}_{b,2}^l) & \dots & (x_{b,R}, \hat{c}_{b,R}^l) \end{bmatrix}^T, \tag{3.5}$$

We use the dummy feature-label pair S_b^l to fine-tune its corresponding sub-model $\theta_{global,l}$. After that, the sub-model $\theta_{global,l}$ $(1 \leq l \leq L)$ is updated to $\theta_{update,l}^b$ $(1 \leq l \leq L)$.

Step 3 (3.1 and 3.2 in Fig. 3.4)

For each updated model $\theta^b_{update,l}$, we obtain its validation accuracy $A^b_{update,l}$ ($1 \leq l \leq L$). Then we compute the forgetting value of batch X_b on each sub-model $\theta_{global,l}$ using the following equation,

$$F^l_b = \begin{cases} 1 & \text{if } A_{global,l} - A^b_{update,l} < \beta \\ 0 & otherwise \end{cases} \tag{3.6}$$

where $1 \leq l \leq L$ and β is the comparison threshold.

We denote the forgetting value of all sub-models as,

$$F_b = \begin{bmatrix} F^1_b & F^2_b & \dots & F^L_b \end{bmatrix}^T, \tag{3.7}$$

where F^l_b ($1 \leq l \leq L$) denotes the forgetting value of the test batch b on the global sub-model $\theta_{global,l}$.

The forgetting value is used to measure the degree of catastrophic forgetting of a global model after learning dummy sample pairs. If the drop in accuracy is greater than a threshold β, we consider catastrophic forgetting has occurred and set the value of F^l_b to 0, otherwise 1.

Step 4 (4.1 and 4.2 in Fig. 3.4)

We use the temperature scaling value and the validation set's accuracy obtained in step 1 to calibrate the forgetting values. The calibrated forgetting values of batch X_b on sub-model $\theta_{global,l}$ are denoted as follows,

$$tf^l_b = \frac{F^l_b}{\sqrt{T_{global,l}}} * A_{global,l} \tag{3.8}$$

The batch X_b's calibrated forgetting values on all the sub-models are represented as,

$$tf_b = \begin{bmatrix} tf^1_b & tf^2_b & \dots & tf^L_b \end{bmatrix}^T, \tag{3.9}$$

where tf^l_b denotes the calibrated forgetting value of the batch X_b on global sub-model $\theta_{global,l}$.

Finally, the calibrated forgetting value tf^l_b is used as the score of the batch X_b on sub-model $\theta_{global,l}$. The client accepts the prediction result of the sub-model θ_{global,l^*}

that has the largest score as the final prediction result of the batch, where

$$l^* = \arg\max_{l} \ tf_b^l. \tag{3.10}$$

3.5 Performance Evaluation

We conduct a series of experiments to verify the performance of our approach. Our experiments aim to answer the following questions. **RQ1**: can our feature extraction and clustering method successfully cluster clients with similar data distributions together? **RQ2**: how does our test scheme select the optimal outcome from a set of global sub-model predictions and what's the overall performance? **RQ3**: what is the overall performance of *ClusterFLADS* compared to existing solutions?

3.5.1 Data and Model Details

Data preparing

Although there are many existing IoT system datasets [85], we cannot find a complete dataset that simultaneously covers all kinds of traffic, e.g., no attacks and no congestion, congestion but no attacks, attacks, and so on. Hence, we have built our own IoT network testbed to collect data, which mainly consists of the firewall, routers, malware (Mirai) control center, monitoring machine, and IoT devices (smart cameras). The overall structure of the testbed is shown in Fig. 3.5. We have captured traffic data with Wireshark in the IoT network when the devices work normally or are attacked by Mirai [25]. Our data includes both benign and malicious data of smart cameras in different scenarios. Specifically, we collect the following datasets:

- *Set M_1:* malicious data. This dataset contains the data from the cameras when we compromise the camera and launch Mirai attacks on the compromised cameras. This dataset includes 64101 packets.
- *Set B_1:* normal benign data (following the distribution $\hat{D}_1$). This data set includes benign data when the cameras are in normal working mode and the network is not experiencing congestion. This dataset includes 20728 packets.
- *Set B_2:* benign data with network congestion issues (following distribution $\hat{D}_2$). This dataset includes benign data from the cameras where the network is congested. To emulate the network congestion scenario, we use other local machines

streaming internet video. Frequent packet re-transmissions and losses are observed in this situation, indicating network congestion. This dataset includes 20102 packets.

- *Set B_3:* benign data during network configuration changes (following distribution $\hat{D}_3$). This dataset includes the benign data from the cameras in the presence of network configuration changes. To emulate this scenario, we changed the maximum TCP segment size (MSS) to a different value. This dataset includes 34971 packets.

We divide each benign set B_i ($1 \leq i \leq 3$) into training set $Train_i$ (80%), validation set $Validation_i$ (10%) and test set $Test_i$ (10%). Then we distribute the training sets to different simulated clients to simulate a federated anomaly detection system. The malicious set M_1 is only used for testing. Note that the clients are simulated using the collected data.

Data preprocessing

We borrow the same data preprocessing method as in DIoT [1]. Specifically, according to the seven characteristics (refer to Table 3.2 for details), we convert the packet sequences $g_1, g_2, \ldots, g_H$ ($1 \leq h \leq H$), H is the number of packets in the sequence) to the symbol sequences $y_1, y_2, \ldots, y_H$ ($1 \leq h \leq H$). Skipping the first 20 symbols, we treat each symbol y_h ($21 \leq h \leq H$) as the actual label a_h and its preceding 20 symbols as the features $d_h = (d_{h,1}, d_{h,2}, \ldots, d_{h,20})$ to establish the feature-label pairs (d_h, a_h). These feature-label pairs will be fed into the detection model as training and test samples.

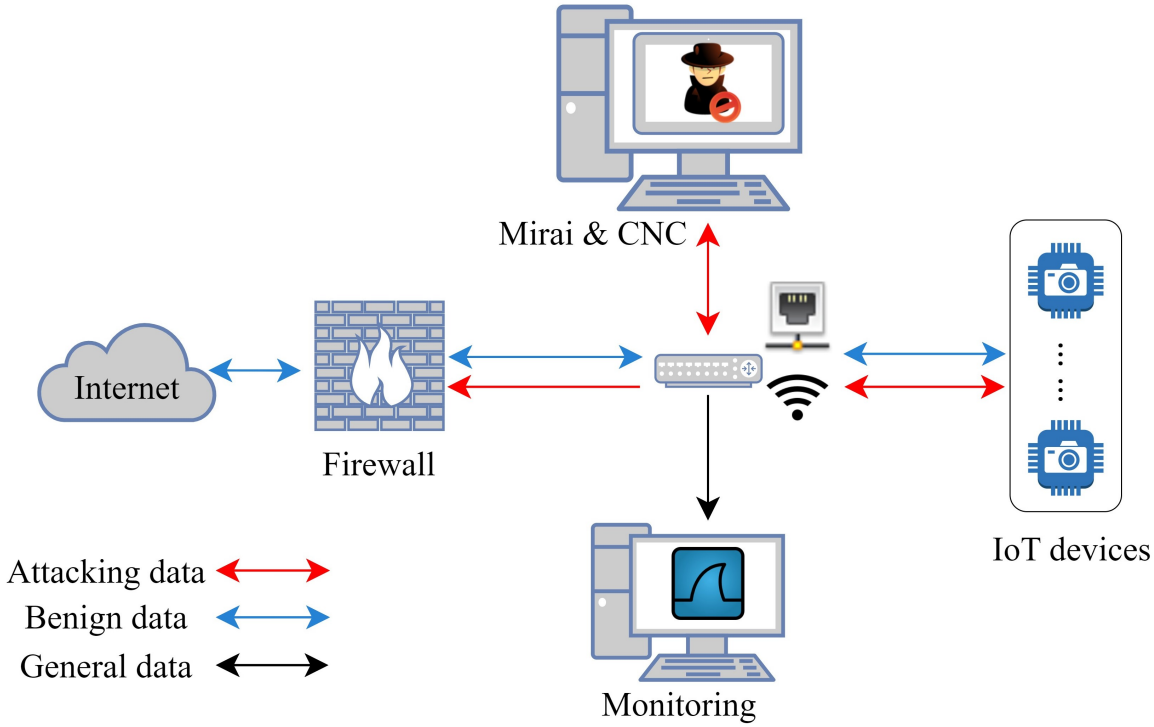


Figure 3.5: The overall structure of lab testbed.

Table 3.2: Characteristics of packets used in symbol mapping [1]

ID	*Characteristics*	*Value*
u_1	direction	0 = outgoing, 1 = incoming
u_2	type of local port	port type's bin index
u_3	type of remote port	port type's bin index
u_4	length of the packet	packet length's bin index
u_5	TCP flags	values for TCP flags
u_6	protocols	encapsulated protocol types
u_7	IAT bin	bin index of packet inter-arrival time (IAT)

Hyper-parameters of the Three-layer LSTM

We adopt a three-layer LSTM as the detection model. The learning rate is set at 0.001, the batch size is 128, and Adam [116] is the optimizer. The anomaly threshold ϵ is set to 0.01 and the trigger threshold γ is set to 0.1.

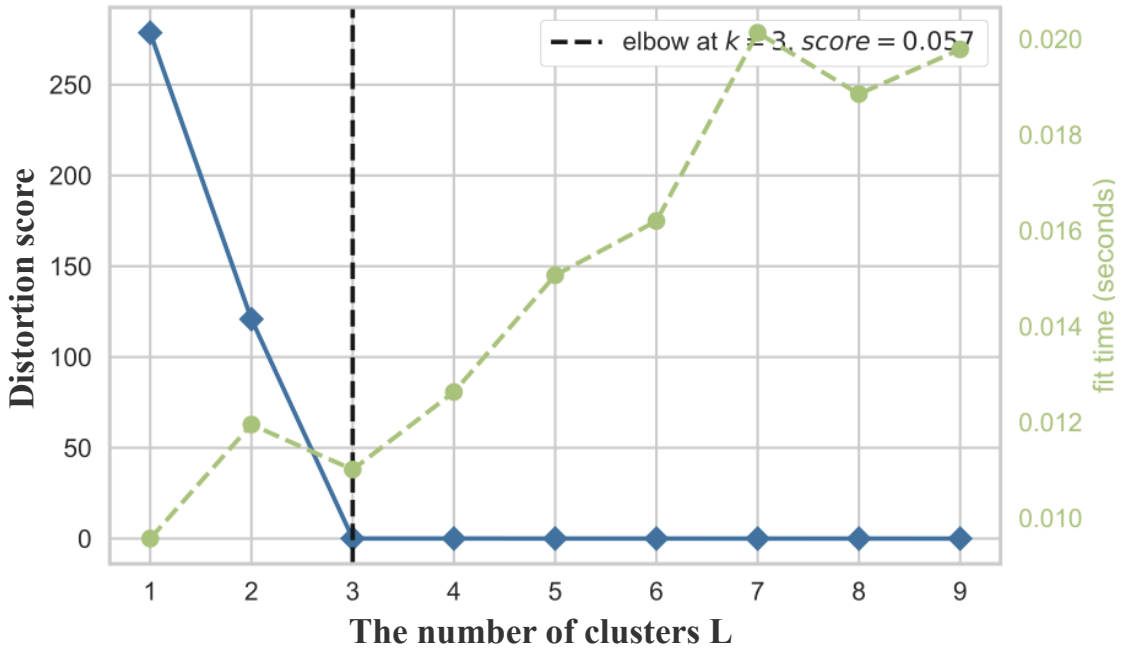


Figure 3.6: The Elbow method [2] is used to calculate the number of clusters with k-means. We plot the distribution score under a range of L values. The distribution score is the sum of the square distance of each point to its cluster center. The optimal L is the Elbow of the plot.

3.5.2 Answer to RQ1

To answer RQ1, we assign $Train_1$ to the first 10 clients, $Train_2$ to the middle 5 clients, and $Train_3$ to the last 10 clients, as shown in the first and second columns in Table 3.3. Each communication round will perform E local epochs and E is set to 5. We set $\hat{T}$ to 2, Z to 2, the batch size to 128, and the learning rate to 0.001. Note that the meaning of notations is defined in Table 3.1.

Experimental results.

The Elbow method [2] is used to calculate the number of clusters with k-means. Fig. 3.6 plots the distribution scores under the different number of clusters, where the distribution score is the sum of the square distance of each point to its cluster center. We can see that the elbow position of the plot is 3, which is consistent with the number of distributions in our data. The clustering results are shown in Table 3.3. The first 10 clients (following data distribution $\hat{D}_1$) are clustered to cluster 1, middle 5 clients (following data distribution $\hat{D}_2$) are clustered to cluster 2, and the last 10

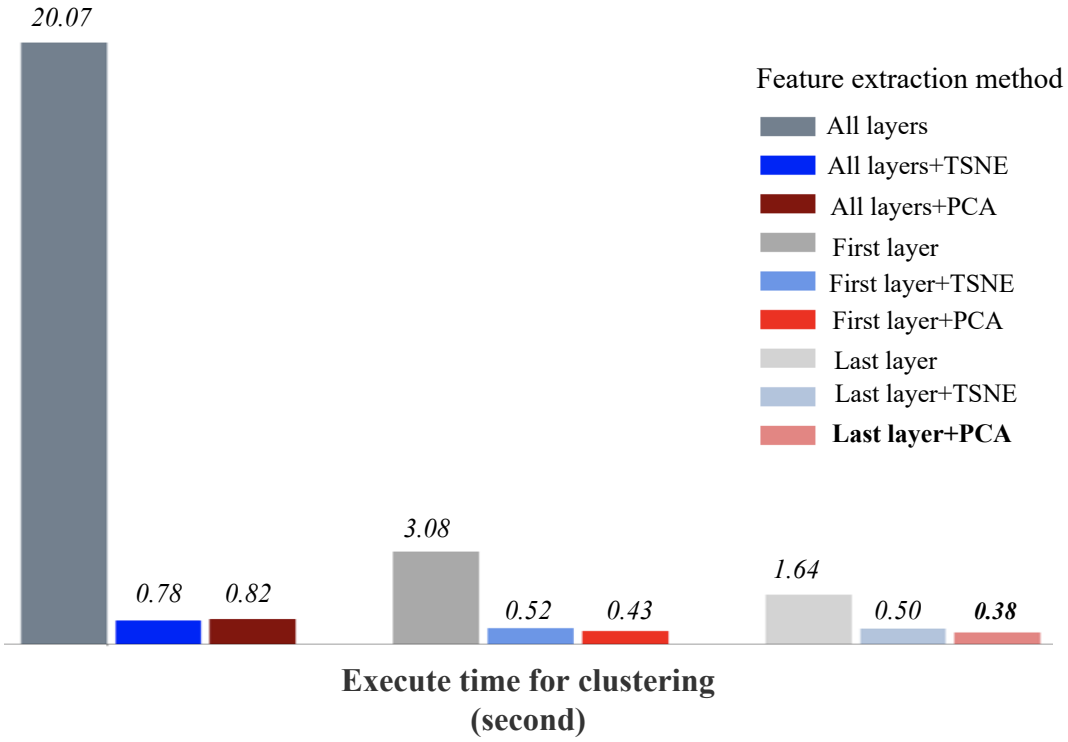


Figure 3.7: The clustering time under different feature extraction methods.

clients (following data distribution $\hat{D}_3$) are clustered to cluster 3. The above test results show that our clustering method successfully clusters clients with similar data distributions together.

Table 3.3: Clustering results for different clients.

Client ID	*Data distribution*	*Cluster ID*
$1 \sim 10$	$\hat{D}_1$	1
$11 \sim 15$	$\hat{D}_2$	2
$16 \sim 25$	$\hat{D}_3$	3

Table 3.4: Overall Performance of clustering under different feature extraction methods.

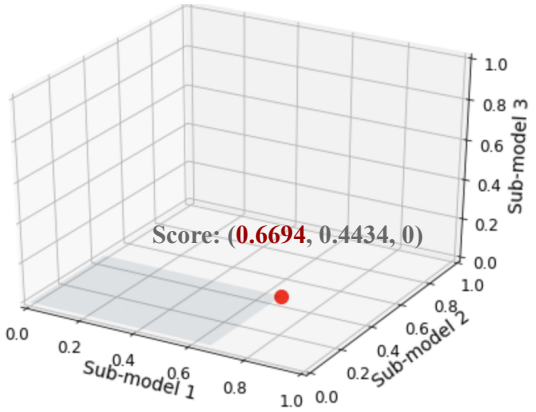
Evaluation metrics	Last layer+TSNE	All layers+PCA	First layer+PCA	Last layer+PCA
Silhouette score [117]	0.578	0.985	0.972	**0.988**
Calinski-Harabasz score [118]	42.025	27045.462	4910.029	**53803.984**
Davies-Bouldin score [119]	0.553	0.022	0.045	**0.016**

*A higher Silhouette score and Calinski-Harabasz score indicate better clustering results. Better clustering outcomes are indicated by a lower Davies-Bouldin score.

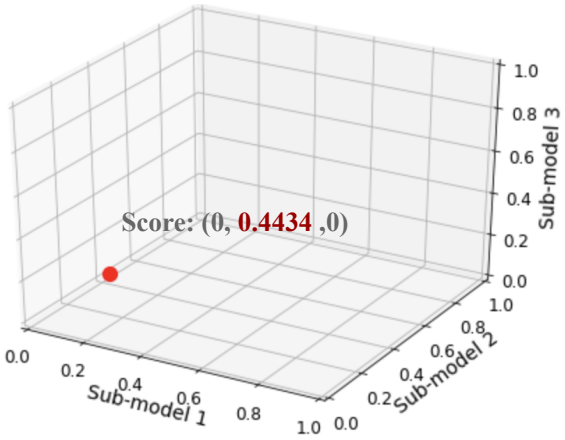
The efficiency of our feature extraction scheme. To demonstrate the effectiveness of our feature extraction scheme, we do a number of experiments. First, we test the running time of clustering under different feature extraction methods. For the configuration of the server, we use a machine with macOS Catalina Version 10.15.7, 16 GB 3733 MHz LPDDR4X RAM, and 2 GHz Quad-Core Intel Core i5 CPU. We consider the case of selecting the first layer, the last layer, and all layers of the model parameters.

We consider two dimensionality reduction methods: Principal component analysis (PCA) [115] and T-distributed Stochastic Neighbor Embedding (TSNE) [120]. Large datasets with high-dimensional features can be handled using the linear dimensionality reduction technique known as PCA. It uses singular value decomposition to convert many correlated variables into a small number of uncorrelated principal components. TSNE [120] is a nonlinear dimensionality reduction method. By minimizing the Kullback-Leibler divergence between the low-dimensional joint probability and the high-dimensional data joint probability, TSNE [120] can perform dimensionality reduction. Specifically, it transforms the similarity of low-dimensional space data points and high-dimensional space data points into joint probability by using Gaussian distribution and long-tailed distribution, respectively, so that a small distance in high-dimensional space will be a large distance after mapping.

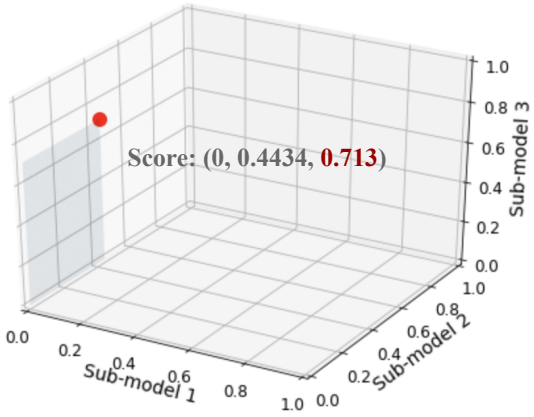
The evaluation results are shown in Fig 3.7. The results show that our feature extraction method dramatically reduces the running time for clustering, benefiting from the efficient combination of PCA and the last layer. This time advantage becomes more pronounced when the number of clients and neural network layers increases, and



(a) The average calibrated-forgetting values of test set 1.



(b) The average calibrated-forgetting values of test set 2.



(c) The average calibrated-forgetting values of test set 3.

Figure 3.8: The average calibrated-forgetting values of different test sets.

the distribution of data becomes more diverse. This explains why we choose some layers of the model instead of all layers to extract features and clusters.

We also compare the clustering performance under different feature extraction methods using the three most popular evaluation metrics:

- Silhouette score [117]. The average of the silhouette coefficients over all samples is known as the silhouette score. The Silhouette Coefficients for each sample are calculated using the formula $(b-a)/max(a,b)$, where a represents the average distance between this sample and other samples in its intra-cluster, and b represents the average distance between the sample and other samples in its closest cluster. Better clustering outcomes are indicated by a higher Silhouette score.

- Calinski-Harabasz score [118]. The ratio of the sum of the between-cluster and within-cluster dispersion is used to determine this score. A larger Calinski-Harabasz score indicates better clustering results.

- Davies-Bouldin score [119]. Each cluster computes its similarity to its most similar cluster. The metric then uses the average of this similarity across all clusters as the score. The similarity is determined by the ratio of the intra-cluster distance to the inter-cluster distance. The smaller the Davies-Bouldin score, the better the clustering results.

As shown in Table 3.4, our feature extraction method (PCA + last layer) can obtain the best clustering performance.

Further analysis. We first demonstrate that clients with smaller distances in model parameters are likely to have more similar data distributions. Suppose we have two clients. We use θ_{FedAvg} to denote the global model trained using FedAvg, θ_{sgd} to denote the model trained locally using data from all clients, θ_{local_1} to denote the local model of client 1, and θ_{local_2} to denote the local model of client 2. The sizes of the training data in client 1 and client 2 are n_1 and n_2, respectively. According to [54], if all clients have the same initialization weights, then the Earth Moved Distance (EMD), which measures the distance between two distributions, is the root cause of weight divergence $||\theta_{FedAvg} - \theta_{sgd}||/||\theta_{sgd}||$. We can rewrite the weight divergence as:

$$
\begin{aligned}
&||\frac{n_1 * \theta_{local_1} + n_2 * \theta_{local_2}}{n_1 + n_2} - \theta_{sgd}||/||\theta_{sgd}|| \\
=&||\frac{n1 * (\theta_{local_1} - \theta_{local_2}) + (n_2 + n_1)(\theta_{local_2} - \theta_{sgd})}{n_1 + n_2}||/||\theta_{sgd}|| \\
\leq&\frac{n_1 * ||\theta_{local_1} - \theta_{local_2}||}{n_1 + n_2}/||\theta_{sgd}|| + ||\theta_{local_2} - \theta_{sgd}||/||\theta_{sgd}||
\end{aligned} \tag{3.11}
$$

The right side of the inequality is determined by $||\theta_{local_1} - \theta_{local_2}||$ if θ_{local_2} and θ_{sgd} are fixed. Therefore, if two clients have the same initialization weights, a larger distance between their local model parameters implies a large distance in their data distribution. In our algorithm, the global model aggregated in the first round of communication will be used as the initial model for each local client j ($1 \leq j \leq N$) in the second round of communication. Therefore, clients with similar data distributions are more likely to have smaller distances in model parameters.

The reason why clustering works better with only the last layer of the model to extract features is mainly that the last layer learns more task-specific features [59, 110–113].

3.5.3 Answer to RQ2

As mentioned in Section 3.3.2, we train a global model for each cluster. The global model of each cluster will be considered a sub-model of our final global anomaly detection system. Our test scheme can select the optimal outcomes for testing data from a set of sub-models' predictions. To evaluate the performance of our test scheme, we use three test sets:

- $Test_1$ that follows the distribution $\hat{D}_1$
- $Test_2$ that follows the distribution $\hat{D}_2$
- $Test_3$ that follows the distribution $\hat{D}_3$

We set the test batch size to 60, the learning rate to 0.1, and the number of epochs to fine-tune the global model to 15. The comparison threshold is set to 0.1.

Experimental results. The overall performance is shown in Table 3.5) and Fig 3.8. The samples in $Test_1$ follow the same distribution $\hat{D}_1$ as cluster 1. We can see that all samples in $Test_1$ (the first row of Table 3.5) have correctly selected the global model of cluster 1 as their optimal sub-model and the average calibrated forgetting score is 0.6694 (see Fig. 3.8 (a)). The samples in $Test_2$ follow the same

distribution $\hat{D}_2$ as cluster 2. We can see that all samples in $Test_2$ (refer to the second row of Table 3.5) have selected the global model of cluster 2 as their optimal sub-model and their average calibrated forgetting score is 0.4434 (see Fig. 3.8 (b)). The samples in $Test_3$ follow the same distribution $\hat{D}_3$ as cluster 3. We can see that all samples in $Test_3$ (refer to the third row of Table 3.5) have selected the global model of cluster 3 as their optimal sub-model and their average calibrated forgetting score is 0.713 (see Fig. 3.8 (c)). The above evaluation results show that our test scheme performed extremely well in automatically selecting the optimal sub-model for the test samples.

Table 3.5: Overall Performance of the test scheme.

		Probability distribution of identified clusters (%)		
Test traffic	Actual cluster	Cluster 1	Cluster 2	Cluster 3
Test set 1	1	100	0	0
Test set 2	2	0	100	0
Test set 3	3	0	0	100

Figure 3.9: Test performance of anomaly detection in different scenarios.

Discussion. Experimental results show that sub-models that observe (1) no catastrophic forgetting after updates, (2) higher validation accuracy, and (3) lower temperature scaling values are more likely to be optimal sub-models for the test batch X_b. Our further explanations regarding these criteria are as follows.

1. According to [54] and Section 3.5.2, we know that a greater test accuracy reduction (compared to θ_{sgd}) of the global model θ_{FedAvg} can be explained by a larger lower bound of the weight distance $||\theta_{local_1} - \theta_{local_2}||$. We use the sub-model $\theta_{global,l}$ as the initial model (the training data $train^l$ of $\theta_{global,l}$ follow the distribution $\mathcal{D}_l$). Then we use S_b^l to update $\theta_{global,l}$ to $\theta_{update,l}^b$, and use the training data $train^l$ from cluster l to update $\theta_{global,l}$ to $\theta_{global,l}^{'}$. Since $\theta_{global,l}$ has converged on the training data $train^l$, $\theta_{global,l}^{'}$ should approximate $\theta_{global,l}$ well. If two clients have the same initial model $\theta_{global,l}$, the more severe the non-IID problem between the data S_b^l and $train^l$, the greater the accuracy reduction in the global model $\theta_{FedAvg} = (n_1 * \theta_{update,l}^b + n_2 * \theta_{global,l})/(n_1 + n_2)$. Since validation set $V_l \sim \mathcal{D}_l$, we have validation accuracy $A_{update,l}^b \leq A_{FedAvg} \leq A_{global,l}$. Hence, the lower bound of $A_{sgd} - A_{update,l}^b$ is larger if $A_{sgd} - A_{FedAvg}$ is larger. Therefore, a more skewed non-IID between the test batch and the cluster's training data implies a larger accuracy reduction of the updated model $\theta_{update,l}^b$. The global sub-model that observes a small accuracy reduction after updating is more likely to be the optimal one for the test batch X_b.

2. A higher validation accuracy means that changes in unit accuracy (a change in unit accuracy means the accuracy is increased or decreased by 1 percent) have less impact on the overall accuracy.

3. A larger temperature scaling value means that when a model has difficulty forgetting, it is mainly due to its own overconfidence rather than the fitness of the test samples.

3.5.4 Answer to RQ3

To answer this question, we introduce two baselines:

1. Baseline 1: this baseline uses the method described in DIoT [1]. It is an IoT anomaly detection method [1] built on the traditional federated learning architecture [46, 82, 82, 83].

2. Baseline 2: this baseline applies the algorithm introduced in IFCA [59]. It should be emphasized that IFCA mainly uses convolutional neural networks and is evaluated on the MNIST and CIFAR datasets, so we only refer to its key algorithm, using our own LSTM network and hyper-parameters. The cluster number is set to 3 and the learning rate is set to 0.001. Similar to the implementation of IFCA [59], we allow all clusters to share the weights for the first three layers and execute the algorithm to learn only the last two fully-connected layers.

We consider three testing scenarios:

1. IID-data scenario: the training data between different clients, and the same client's test data and training data are all IIDs.

2. Non-IID scenario 1: the training data among different clients are non-IID, but the same client's test data and training data are IID.

3. Non-IID scenario 2: the training data among different clients are non-IID and the same client's test data and training data are non-IID.

Experimental results. The evaluation results are shown in Fig. 3.9. The figures in the first row show the evaluation results of IID data, the figures in the second row show the evaluation results of non-IID data in the spatial domain (i.e., non-IID data among different clients), and the figures in the third row show the evaluation results of non-IID data in the temporal domain (i.e, the same client's test and training data are non-IID).

We compared the false positive rate (FPR), true positive rate (TPR), and overall accuracy in our method with baseline methods under different test scenarios. The true positive rate (TPR) means the probability of successfully detecting the malicious samples, and the false positive rate (FPR) means the probability of falsely detecting the benign samples as the malicious samples. The percentage of successfully identified samples to all testing samples is represented by the accuracy rate. All methods achieve good performance in the IID scenario (i.e, 100% true positive rate (TPR), 0% false positive rate (FPR), and 100% accuracy).

For all non-IID scenarios, baseline 1 observed an improved false positive rate (FPR). This is due to the negative impact of local models on each other during aggregation.

Baseline 2 solves the non-IID problem in non-iid scenario 1 (spatial domain) but ignores the non-IID problem in non-IID scenario 2 (temporal domain). Although all clusters share the first few layers, task-specific knowledge is mostly contained in the last few layers. Therefore, each cluster mainly learns knowledge belonging to a specific distribution and lacks knowledge of other clusters. Since each client selects its own cluster's global model for testing, very high false positives are observed when its test and training data are non-IID.

By effectively combining our clustering, training, and testing methods, *ClusterFLADS* achieves excellent performance, i.e., 100% true positive rate and no false positives, in all the test scenarios.

3.6 Conclusion and Future Work

Motivated by the observation that FL-based IoT traffic anomaly detection suffers from data distribution shifts in both temporal and spatial domains, we present a new clustered federated learning approach to IoT anomaly detection. *ClusterFLADS* takes advantage of the false predictions of the inappropriate global models, together with knowledge of temperature scaling and catastrophic forgetting to reveal distributional similarities between the test data and the training data (of different clusters). The global models that did not show catastrophic forgetting after updating (e.g., validation accuracy reduction is smaller than a threshold) had a better predictive ability on the test data. If more than one such global model exists, we choose the global model with a higher validation accuracy and a smaller temperature scaling value.

We design an efficient feature extraction scheme by exploiting the difference in the role each layer of a neural network plays in the learning process. By carefully selecting model parameters and conducting dimensionality reduction, our approach can effectively improve the clustering speed and thus reduce the anomaly detection delay.

To evaluate the performance of *ClusterFLADS*, we collected real-world IoT trace data and used the data to simulate different scenarios. The extensive experiment showed that *ClusterFLADS* could cluster clients accurately and quickly. It outperformed two existing baseline solutions and achieved excellent test performance (100% true positive rate and no false positives) under different data distributions.

While *ClusterFLADS* was motivated to address the non-IID problem in FL-based IoT anomaly detection systems, it should be generic enough to be applicable in other

RL application domains. Nevertheless, we may need to adapt the *ClusterFLADS* detection model and feature extraction method to make it work for a particular application. We leave this as our future work.

Algorithm 4: Training of *ClusterFLADS*

Input: Privacy data D_j and number of training samples q_j of each client j ($\forall 1 \leq j \leq N$), total number of training samples q

Output: Global model $\theta_{global,l}$ ($\forall 1 \leq l \leq L$) of the cluster l

Initialize global model parameters $\theta_{global}^{(1)}$;
for *communication round* $t \leftarrow 1$ **to** $\hat{T}$ **do**
  for *client* $j \leftarrow 1$ **to** N **do**
    $\theta_j^{(t)}$ = ClientExecute($\theta_{global}^{(t)}$, j);
  end
  if $t < \hat{T}$ **then**
    $\theta_{global}^{(t+1)} = \sum_{j=1}^{N} \frac{q_j}{q} \theta_j^{(t)}$
  else
    for *client* $j \leftarrow 1$ **to** N **do**
      f_j = FeatureExtraction($\theta_j^{(t)}$)
    end
end
Execute k-means to cluster features $f_1, \ldots, f_N$ into L clusters ;
Calculate the number of clients $\hat{l}$ in each cluster l and the number of training samples q^l in cluster l ;
Initialize $\theta_{global,l}$;
for *cluster* $l \leftarrow 1$ **to** L **do**
  repeat
    for *client* $o \leftarrow 1$ **to** $\hat{l}$ **do**
      $\theta_{l,o}$=ClientExecute($\theta_{global,l}$,o);
    end
    $\theta_{global,l} = \sum_{o=1}^{\hat{l}} \frac{q_o^l}{q^l} \theta_{l,o}$
  until *Convergence*;
end
return *Global model* $\theta_{global,l}$ *($\forall 1 \leq l \leq L$) of the clusters;*
ClientExecute (w, j)**:**
  for *local epoch* $e \leftarrow 1$ **to** E **do**
    for *batch* $b \leftarrow 1$ **to** B **do**
      $w = w - \eta \frac{\partial l(w;b)}{\partial w}$
    end
  end
  return w to the server;
FeatureExtraction (θ_j)**:**
  Extract the parameters θ_j^U;
  f_j=PCA(θ_j^U);
  return f_j to the server;

Algorithm 5: Testing of *ClusterFLADS*

Input: Global sub-models $\theta_{global,l}$ ($\forall 1 \leq l \leq L$), testing batch X_b, validation set $\mathbf{V}$
Output: l^*

for *cluster* $l \leftarrow 1$ **to** L **do**
    $A_{global,l}$ = DetectionAcc($\theta_{global,l}$,$\mathbf{V}$) ;
    $T_{global,l}$ = Temperature($\theta_{global,l}$,$\mathbf{V}$) ;
    Feed X_b to $\theta_{global,l}$ and obtain S_b^l ;
    Use S_b^l to update $\theta_{global,l}$ to $\theta^b_{update,l}$;
    $A^b_{update,l}$ = DetectionAcc($\theta^b_{update,l}$,$\mathbf{V}$) ;
end
Compute F_b based on Equation 3.6 ;
Compute tf_b based on Equation 3.8 ;
Compute l^* based on Equation 3.10 ;
return l^* ;
DetectionAcc (*Model, S*)**:**
    len=$S.length()$;
    acc=count=0
    for *sample* $(x, c) \in$ *Set* S **do**
        Predicted label $\hat{c} = Model(x)$;
        if $\hat{c} = c$ **then**
            count++ ;
    end
    $acc = \frac{count}{len}$;
    return *acc*;
Temperature (*Model, S*)**:**
    initialize temperature ;
    optimizer2 = torch.optim.LBFGS([temperature]) ;
    for *epoch* $e \leftarrow 1$ **to** E **do**
        for *batch* $b \leftarrow 1$ **to** B **do**
            optimizer2.step(closure());
        end
    end
    return temperature ;
closure ()**:**
    clear the gradients ;
    TScale = TScaling(logits,temperature) ;
    loss = CrossEntropyLoss(TScale,labels) ;
    loss.backward() ;
    return loss ;
TScaling (*logits, temperature*)**:**
    TScale = $\frac{logits}{temperature}$;
    return TScale ;

Chapter 4

Root Cause Analysis for Federated Learning-based IoT Anomaly Detection

4.1 Introduction

Due to the lack of built-in security controls [108, 121] in IoT devices and the "rush to market" mentality of IoT device manufacturers [1, 122], IoT anomaly detection has attracted a lot of attention. To leverage data on large-scale IoT devices of local users without compromising privacy, federated learning (FL) [46, 82, 82, 83] has been applied for IoT anomaly detection [1, 123]. As shown in Fig. 4.1, a global model is trained for each type of IoT devices since the same type of IoT devices adopt similar configurations (e.g., default factory settings) and thus yield similar traffic patterns [1, 81].

Nevertheless, the traffic from different clients in a real IoT system may be non-independent and identically distributed (i.e., non-IID), leading to quite different IoT traffic patterns even for the same type of devices with the same configuration. This is due to the fact that the performance of local/global models in federated architectures varies over time and location changes (we call these spatial-temporal dynamics in network environments), e.g., duplicate TCP packets triggered by TCP retransmissions due to link failures or network congestions. Consequently, a large number of FP events (false alarms) could raise (please refer to our experimental results shown in Fig. 4.2 for more details) and overwhelm the operator of the anomaly detection system. Hence,

it becomes incredibly critical to tell apart the root causes for the raised alarms (either true or false).

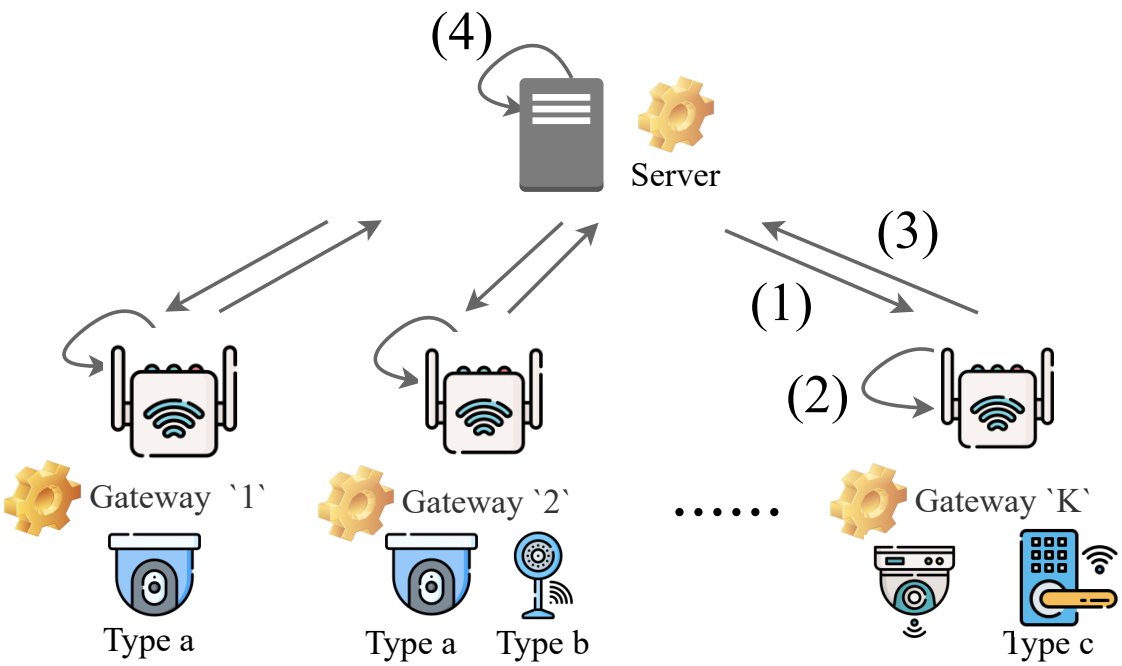
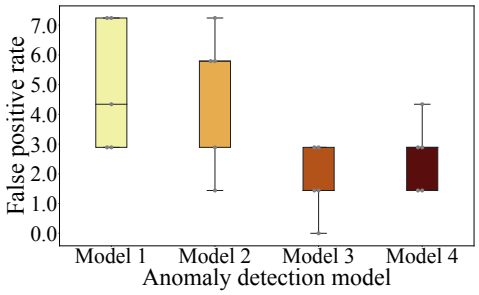


Figure 4.1: The architecture of FL-based anomaly detection. The training process consists of the following four steps: 1. Each client downloads the initial global detection model. 2. The client updates the global model with local data. 3. The client sends the updated model to the server. 4. The server aggregates the local models from the clients.

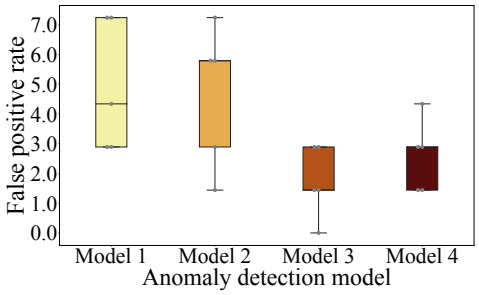
4.1.1 Motivation

Root cause analysis has been widely applied to reason the raised failures or abnormal events. (Refer to Sec. 4.2 for a thorough review of existing methods). When adopting them for FL-based anomaly detection in IoT applications, however, we are faced with the following problems.

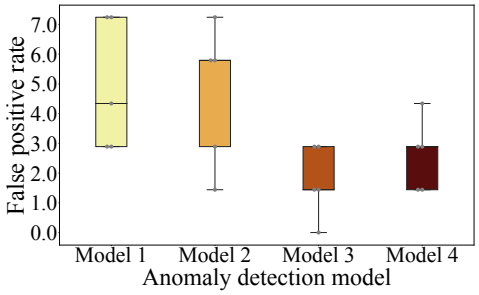
- Root cause analysis applied to IoT systems require better generalization capabilities. Due to the unstable network environment of the IoT system, abnormal data caused by the same reasons will also fluctuate greatly. Additionally, cunning and variable attack can take new forms we haven't seen before. In this case, we hope that the system can see the essence through the phenomenon,



(a)



(b)



(c)

Figure 4.2: Boxplot of FPR (%) of different anomaly detection models when (a) data from different clients are IID, (b) the training data and testing data from the same client are IID, but data from different clients are non-IID. This emulates the situation where different clients may have different network configurations or network conditions, (c) the training data from different clients are IID, but the training data and test data from the same client are non-IID. This happens when network conditions change significantly over time and such changes may or may not be triggered by attacks. We consider four different models: a 3-layer GRU model (marked as "model 1", also the same model used in DÏoT [1]), a 3-layer LSTM network (marked as "model 2"), an encoder-decoder GRU network (marked as "model 3") and an encoder-decoder LSTM network (marked as "model 4").

and correctly find the most likely cause for the anomaly instead of becoming random.

- IoT systems have higher requirements for collaboration capabilities. Due to the diversity and complexity of IoT anomaly scenarios, it is difficult for local users to train a powerful root cause analysis model alone. Nevertheless, the existing root cause analysis solutions [124–127] are mainly trained and applied locally, largely ignoring the collaboration capabilities required by the IoT system.

- IoT root cause analysis systems need to be capable of lifelong learning to adapt to new emerging anomaly scenarios. IoT technology is changing rapidly, and the known anomaly scenarios may only be the tip of the iceberg. Therefore, it is difficult for us to train a root cause analysis model that can cover all kinds of anomaly scenarios. Therefore, the system should have the ability to continuously learn new knowledge and self-renew.

In consequence, existing solutions cannot well solve the task of root cause analysis in IoT systems. Unstable network environments and new forms of attack can significantly degrade test performance for root cause analysis. Therefore, the first question we need to answer is: *how to improve the generalization ability?* Additionally, blind emphasis on collaboration may jeopardize user privacy. Therefore, the second question we need to answer is: *how can the local users collaborate while ensuring privacy?* Third, constantly emerging new anomaly scenarios can render existing systems obsolete. Therefore, the third question we need to answer is: *how can the system continuously learn new knowledge and self-renew to adapt to new emerging anomaly scenarios?*

4.1.2 Challenges and Contributions

To design a root cause analysis solution that can solve the above three questions in an IoT system, we have several challenges. First, the possible fluctuations and new forms of the data are unknown, so it is difficult for the model to learn the essential pattern through the phenomenon from the existing training data belonging to each scenario. Actually, classifiers often rely on specific rules learned from the feature-label pairs of the training data to determine the most likely category for the test samples. However, this method may make the model pay too much attention to details and even learn the noise of individual instances [128], which is not conducive

to improving the generalization ability of the model. Second, we can add noise to the data samples [129,130] so that local users can share their data to train the global model without revealing their privacy. However, the randomness of noise and the uncertainty of adding noise methods can destroy useful patterns contained in raw data and reduce test performance. Third, the process of learning new knowledge and self-renewal often renders the model temporarily unusable [131], leading to great inconvenience and safety hazards in people's daily life.

To address the above challenges, we propose a new framework named *Score-VAE* to analyze the root cause of anomalies in IoT systems. *Score-VAE* designed a new training scheme and testing scheme that can fit the requirements of generalization, lifelong learning, collaboration, and privacy protection in root cause analysis of IoT systems. Specifically, we summarize the major contributions of this work as follows.

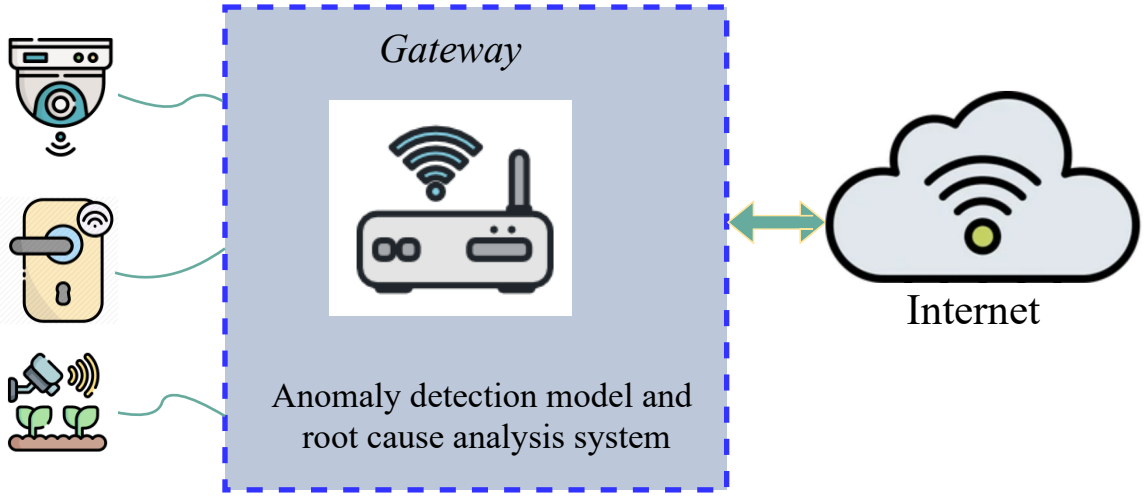


Figure 4.3: *Score-VAE* at the gateway for root cause analysis.

- We introduce *Score-VAE*, a new root cause analysis framework of anomalies in IoT systems, which is deployed at the same place as the anomaly detection model, e.g., the gateway as in Fig. 4.3. To improve the generalization capacity, our system trains a separate global model for each anomaly scenario and relies on the reconstruction effect of the different scenario's global VAE root cause analysis models on the test sample to judge its category, rather than use a single model to learn rules directly from the mapping relationship of feature label pairs. Benefiting from this multi-model framework, we can also build global models for new emerging scenarios without disturbing existing ones.

- To collaboratively train a powerful global model without leaking privacy and further improve the generalization capacity, we introduce a new training procedure such that the actual input to the local VAE network only contains the distance relations of the scaled symbols in the original samples, and then use the decoders uploaded by all the local models participating in the training of a certain scenario and noise to generate new samples to train the global model for that scenario. Both the local models and the global models are realized through our dynamic VAE network, where the weights for different losses change dynamically during training based on the knowledge of stop gradient [78, 79] and multi-task learning [70]. Furthermore, we propose a new test scheme based on Hamming distance. Benefiting from the dynamic VAE network and the Hamming test scheme, the negative impact of noise can be addressed.
- We extensively evaluate the performance of *Score-VAE* using real-world IoT data. The experimental results show that *Score-VAE* can well meet the requirements of collaboration and privacy protection, thus adapting to the federated learning architecture. Compared to the baselines, *Score-VAE* also shows stronger generalization ability and thus better test accuracy.

4.2 Related Work

Existing work of root cause analysis can be roughly divided into two categories: i) identifying root causes with expertise and ii) identifying root causes with machine learning (ML).

4.2.1 Identify Root Causes with Expertise

In this category, expert knowledge is used to formulate specific rules [75–77] for determining the root cause of failures. Mourtzis [75] designed a knowledge-based social networking app for employees to collaboratively diagnose problems during the development of large projects. Mamoutova [77] utilize expert knowledge to analyze failures of storage systems.

Solutions in this category can achieve excellent performance with specialized knowledge of fault classification or identification. However, in our case, we do not have enough expertise to establish effective rules. Different from traditional Internet applications, the Internet of Things is an emerging industry that has developed

rapidly in recent years. Short development history, rapid replacement, and uneven products make it particularly difficult to acquire the expertise to identify different IoT anomaly scenarios.

4.2.2 Identify Root Causes with ML

Bayesian networks, decision trees, and neural networks are the three most commonly used ML techniques in identifying the root cause.

Bayesian Networks

Bayesian networks have been widely applied to represent the relationship between alarms and causes. Chockalingam et al. [132] proposed a new framework for analyzing the root causes of problems observed in cyber-physical systems. However, conditional probability tables in directed acyclic graphs are difficult to obtain efficiently. Wang et al. [133] formulated a strategy for examining the underlying causes of a unique kind of alert in thermal power plants. The Bayesian network's prior and posterior probabilities were updated using data samples, and the set of parent nodes with the highest posterior conditional probability was used to identify the root cause. However, it requires prior knowledge to be reliable.

Decision Trees

Chen et al. [125] introduced a decision tree-based approach for failure diagnosing. They trained the decision tree by using the request traces and run-time information from time periods when user-visible failures were presented. Yang [134] devised an online random forest to diagnose sensor failures. Detzner et al. [135] employed an interactive decision tree for root-cause analysis of product detection in manufacturing companies. While decision trees are good at interpreting recognized causes, they are inefficient on time series data, and errors may increase fast when the number of categories is large.

Neural Networks

Roelofs et al. [124] introduced a method called Anomaly Root Cause Analysis. Their method could find those features contributing more to wind turbine anomalies by

applying an optimization algorithm to an autoencoder network. However, this approach assumes that the anomaly detection system has no false positives, which may not be possible in reality. Velásquez et al. [126] designed a new proposal that adopts RL based neural network classifier to identify the faults in power transformers. Nevertheless, the implementation of the method is too complex. Additionally, existing works [124, 126, 127, 136, 137] mainly focus on identifying the root causes of anomalous events that have already been confirmed as failures or attacks and trained locally, ignore the generalization, life-long learning, collaboration, and privacy protection capacities required by root cause analysis of IoT systems.

This work belongs to the second category and is realized through the neural network technique. Different from existing work, our method involves anomalous scenarios not only malicious attacks but also false positives caused by environmental changes. Furthermore, instead of locally training and applying a classifier like previous work, our privacy-preserving training scheme and hamming test scheme help users jointly train a global root cause analysis model with better generalization and lifelong learning capacities.

Table 4.1: Packet characteristics used in symbol mapping [1]

ID	*Characteristic*	*Value*
g_1	direction	0 = outgoing, 1 = incoming
g_2	the type of local port	port type's bin index
g_3	the type of remote port	port type's bin index
g_4	the length of packet	packet length's bin index
g_5	TCP flags	TCP flag values
g_6	protocols	encapsulated protocol types
g_7	IAT bin	bin index of packet inter-arrival time (IAT)

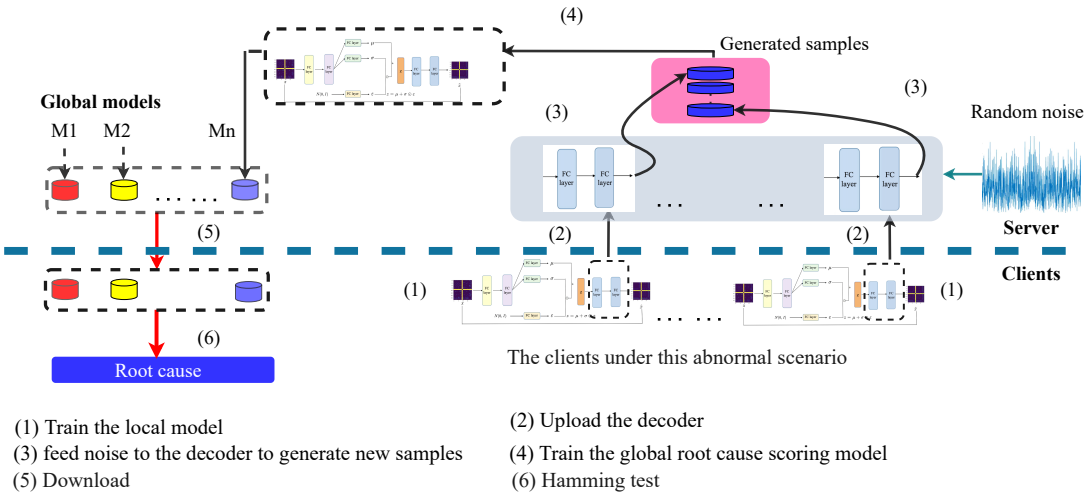


Figure 4.4: Architecture of *Score-VAE*: the black arrows denote the flow of training global VAE root cause analysis model, and the red arrows denote the flow of testing.

4.3 Overview and architecture of *Score-VAE*

4.3.1 Design Rationale

As indicated in Section 4.1, FL-based anomaly detection models could observe a large number of FP alarms in complex IoT scenarios. The workload of the operator could be much reduced if the false positives and true positives could be distinguished correctly. Intuitively, we can either (1) develop new methods to enhance existing FL-based anomaly detection or (2) design a separate root cause analysis system as an add-on service that automatically discovers the causes of an alarm. It is a long journey in the former direction [138, 139], and it is unlikely we will have a breakthrough in the short term. We thus adopt the latter method.

Designing a root cause analysis system as an add-on service to complement the functionality of the existing FL-based anomaly detection system is also based on the following rationale:

- First, due to the imbalance of benign and attack data or the lack of (unknown) attack data, it is more convenient to train an anomaly detection model with benign data only [140]. This follows the commonly-accepted assumption that device manufacturers are not malicious, i.e., IoT devices may be vulnerable but should be trusted when first released by a manufacturer. As such, there is a time

period long enough to learn benign models of IoT devices. Due to this reason, many IoT anomaly detection systems such as [1] are trained with benign data only.

- Second, re-training an existing anomaly detection model requires substantial overhead in both time and cost. Even though machine unlearning methods [53, 91, 106] may be used to speed up the model update, it is still very time-consuming and undesirable to update the global anomaly detection model for better accuracy whenever a new scene (e.g., network congestion) is observed in local gateways.

- Finally, root cause analysis and global anomaly detection can use different modeling techniques, and the model for root cause analysis can be trained independently from that for anomaly detection. In addition, we can train *Score-VAE* incrementally based on new scenarios observed in local gateways, offering us great flexibility for extending the *Score-VAE* model. Whenever a new scene is observed, we can collect the corresponding data and use them to update the root cause analysis without touching the global anomaly detection. This benefit is further illustrated in the evaluation Section 4.7.2.

Root cause analysis has been widely used for reasoning a failure or abnormal event. However, existing root cause analysis solutions highly rely on rules learned from the mapping relationships of feature-label pairs for anomalous scenario classification, lack generalization ability, and are mostly locally trained. Therefore, designing a root cause analysis system, which has better generalization ability, and can adapt to the FL architecture, will address these gaps and help FL-based anomaly detection systems significantly reduce operator workload.

4.3.2 Architecture and Workflow

A high-level view of *Score-VAE* architecture and its workflow is shown in Fig. 4.4. The black arrows denote the flow of training the global VAE root cause analysis model, and the red arrows denote the flow of testing.

The procedures for training the models of *Score-VAE* are elaborated in Section 4.4. For each scenario, we generate new training samples for the global model by feeding noise into the decoders of the local models uploaded by the clients participating in

training for that scenario. Both the local models and the global models are realized through our dynamic VAE networks.

The procedures for the test scheme of *Score-VAE* are elaborated in Section 4.5. *Score-VAE* works with existing anomaly detection models in the following way: the test samples (i.e., incoming traffic) first pass through the anomaly detection model, which raises alarms whenever anomalies are detected. Anomaly samples are passed to the downloaded *global root cause model* for Hamming testing, and then find the root cause of the anomaly.

4.4 Training procedure of *Score-VAE*

In this section, we introduce the training procedure of *Score-VAE*. We introduce a joint noise privacy-preserve training scheme (we use the term "joint noise" because with the help of noise, clients can jointly train a powerful model while ensuring privacy) that can collaboratively train a powerful global variational autoencoder (VAE) [80, 141–143] root cause analysis model while preserving clients' privacy. Our training scheme consists of five steps [144] [145] [129] [143].

- *Step-0:* perform two rounds of preprocessing on the training data.
- *Step-1 (refer to (1) in Fig. 4.4):* each client trains the local VAE generative model by using the dynamic VAE network and the client's private data.
- *Step-2 (refer to (2) in Fig. 4.4):* all clients upload their decoder of the local VAE generative model trained in the first step to the server.
- *Step-3 (refer to (3) in Fig. 4.4):* the server uses random noise $g \sim \mathcal{N}(0, 1)$ to generate new data points from all the received local VAE generative models [80, 129, 141]. The newly generated data can be treated as the server's public data.
- *Step-4 (refer to (4) in Fig. 4.4):* the server trains its global VAE root cause scoring model for each anomaly scenario by using public data belonging to that scenario. The training method of the global VAE root cause scoring model is the same as the training method in *Step-2*.

The details of the two rounds of preprocessing are displayed in Section 4.4.1. And in Section 4.4.2, we explain the details of dynamic VAE networks that are used to train the local and global root cause analysis models.

4.4.1 Two-round data processing

First round of data processing

As introduced in Section 4.7.1, we built an IoT testbed with smart cameras (model DLink DCS-932LB) to emulate various IoT scenarios and collect data. We then map each packet to a symbol based on the seven characteristics of the packet listed in Table 4.1. Specifically, the mapping assigns each unique combination of packet characteristics $(g_1, \ldots, g_7)$ a dedicated symbol representing the "type" of the particular packet [1]. Note that the symbols are indexed, and it is the index numbers that are used for model training. Thus, the sequence of packets can be translated into a sequence of symbols. We then adopt a moving window that covers 20 packets[1] For each packet, we use its symbol as the labeled class and the symbol sequence in its proceeding 20 packets as its feature. In this way, we can translate each packet into a sample.

Second round of data processing

For ease of training and to protect privacy, we conduct the second round of preprocessing on the data samples. First, we scale each sample's feature values to a narrow range (A, B) using min-max scaling [146], where A and B are two hyperparameters, whose settings can be found in Section 4.7. Specifically, denote a sample $D_j = (f_j^0, f_j^1, \ldots, f_j^{19})$, where $f_j^i (i = 0, \ldots, 19)$ are the 20 feature values. Denote $f_{max}^i = max\{f_0^i, f_1^i, \ldots\}$ over all samples, and $f_{min}^i = min\{f_0^i, f_1^i, \ldots\}$ over all samples. Then each sample $D_j = (f_j^0, f_j^1, \ldots, f_j^{19})$ is scaled to $\hat{D}_j = (\hat{f}_j^0, \hat{f}_j^1, \ldots, \hat{f}_j^{19})$, where $\hat{f}_j^i = \frac{f_j^i - f_{min}^i}{f_{max}^i - f_{min}^i} * (B - A) + A$. After that, we convert the input features $(\hat{f}_j^0, \hat{f}_j^1, ..., \hat{f}_j^{19})$ of the transformed data sample $\hat{D}_j$ into a matrix X_j, denoted as

$$X_j = \begin{Bmatrix} d_j(0,0) & d_j(1,0) & ... & d_j(19,0) \\ d_j(0,1) & d_j(1,1) & ... & d_j(19,1) \\ ... & ... & ... & ... \\ d_j(0,19) & d_j(1,19) & ... & d_j(19,19) \end{Bmatrix}$$

where $d_j(m,n)$ $(0 \leq m, n \leq 19)$ represents the absolute distance between $\hat{f}_j^m$ and $\hat{f}_j^n$ in the transformed data sample $\hat{D}_j$. Then the matrices will be used to train *Score-*

[1]The window size of 20 packets for feature representation is empirical and was validated to be effective for anomaly detection in [1].

VAE. The above processing steps make the actual inputs of the local VAE network only contain the distance relations of the scaled symbols in the original samples. By further combining with our joint noise training scheme, the privacy of users is greatly protected.

For ease of understanding and intuitively illustration our testing process (as shown Fig. 4.6), we in the rest of the paper treat matrix X_j as an image of size 20×20 pixels, where $d_j(m, n)$ $(0 \leq m, n \leq 19)$ denote the pixel values. In other words, each sample is represented by a small image.

4.4.2 Dynamic VAE Network

As mentioned earlier, the local models and the global root cause analysis models are realized through the dynamic VAE network. We call this a dynamic VAE network because the weights for different losses change dynamically during training.

Architecture and workflow

The architecture of the dynamic VAE network is shown in Fig. 4.5. VAE [141,144,145] network consists of three parts: encoding, reparameterization, and decoding. Given a matrix sample X_j, the encoder approximates the posterior distribution $p(z_j|X_j)$ of the latent variable. The posterior distribution is assumed to be a normal distribution $\mathcal{N}(\mu_j, {\sigma_j}^2)$ with mean μ_j and standard deviation σ_j. Then we use reparameterization, which samples latent variable from the posterior distribution $p(z_j|X_j)$ and feed it into the decoder to obtain the reconstructed sample $X_j^{'}$.

Loss function

It is worth noting that we use a dynamic loss function in the VAE network to replace the traditional loss function. The detail of the training process of our dynamic VAE network is illustrated in Alg. 6. Specifically, for each training sample j in batch b, we calculate its reconstruction loss $\mathcal{L}_j^{rec}$, kl loss $\mathcal{L}_j^{kl}$, and distribution loss $\mathcal{L}_j^{dis}$. The reconstruction loss $\mathcal{L}_j^{rec}$ is the L_1 norm distance $|X_j - X_j^{'}|$ of the reconstruction sample $X_j^{'}$ and the original sample X_j. The kl loss $\mathcal{L}_j^{kl}$ represents the KL divergence $\frac{1}{2} * (2log(\sigma_j) + 1 - \sigma_j^2 - \mu_j^2)$ of the posterior distribution $p(z_j|X_j)$ of the latent variable and the standard normal distribution $\mathcal{N}(0, 1)$. The distribution loss is determined by the probability that the original sample X_j belongs to the posterior distribution

$p(z_j|X_j)$ of the latent variable, denoted as

$$\mathcal{L}_j^{dis} = -log(p(X_j|\mu_j, \sigma_j^2)) = -log(\frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(X_j-\mu_j)^2}{2\sigma_j^2}}).$$

Algorithm details

We calculate the total reconstruction loss, *kl* loss, and distribution loss for each batch b of J samples, i.e., $\mathcal{L}^{rec}(b) = \sum_{j=0}^{J-1} \mathcal{L}_j^{rec}$, $\mathcal{L}^{kl}(b) = \sum_{j=0}^{J-1} \mathcal{L}_j^{kl}$ and $\mathcal{L}^{dis}(b) = \sum_{j=0}^{J-1} \mathcal{L}_j^{dis}$. Similar to [147], we adopt dynamic loss $\mathcal{L}^{SUM}(b)$, which is the weighted sum of the reconstruction loss, the *kl* loss, and the distribution loss, denoted as $\mathcal{L}^{SUM}(b) = w_{rec,kl} * (\mathcal{L}^{kl}(b) + \mathcal{L}^{rec}(b)) + C_1 * w_{dis} * \mathcal{L}^{dis}(b) + C_2 * log(v_{rec,kl} * v_{dis})$ (we add a constraint term $C_2 * log(v_{rec,kl} * v_{dis})$ to avoid the weights becoming too small, C_1 and C_2 is the hyper-parameters used to adjust the order of magnitude of different terms), where $v_{dis} = 2 * {w_{dis}}^{-1/2}$. The weights of different losses can be automatically tuned with model parameters through gradient descent as shown in Alg. 6. The reconstruction loss and the *kl* loss share the same weight $w_{rec,kl}$ because they are losses in the traditional VAE network [141]. Then we check whether the weight of the loss is out of the defined range (Line 9 to Line 11 in Alg. 6). However, this process may make the loss calculation process discrete, and we cannot calculate the gradient. To solve the problem, we use an intermediate vector and stop gradient method as shown from Line 12 to Line 14 in Alg. 6. Specifically, We use the PyTorch *.detach()* method to stop gradient [79, 80]. During forward-propagation, the *.detach()* method does not make any changes. We compute the loss of $v_{dis} = a_{dis} + (v_{dis} - a_{dis})$. During back-propagation, the part with *.detach()* does not compute the gradient, so we only compute the gradient of $v_{dis} = a_{dis}$. Note that a_{dis} is continuous and its gradient can be calculated.

Discussion of the efficiency of distribution loss

The purpose of introducing the distribution loss is to maximize the probability that X_j belongs to the posterior distribution $q(X_j^{'}|z_j)$ of the reconstruction sample $X_j^{'}$.

The goal of reconstruction loss is to minimize the distance $|X_j - X_j^{'}|$, where $X_j^{'}$ is the reconstruction sample and X_j is the original sample. The goal of the Kullback–Leibler loss is to make sure the distribution of the latent variable z_j follows the standard normal distribution. We denote the distribution loss as $L_j^{dis} = -log(P(X_j|z_j)$. So minimizing the distribution loss is equal to minimizing the dis-

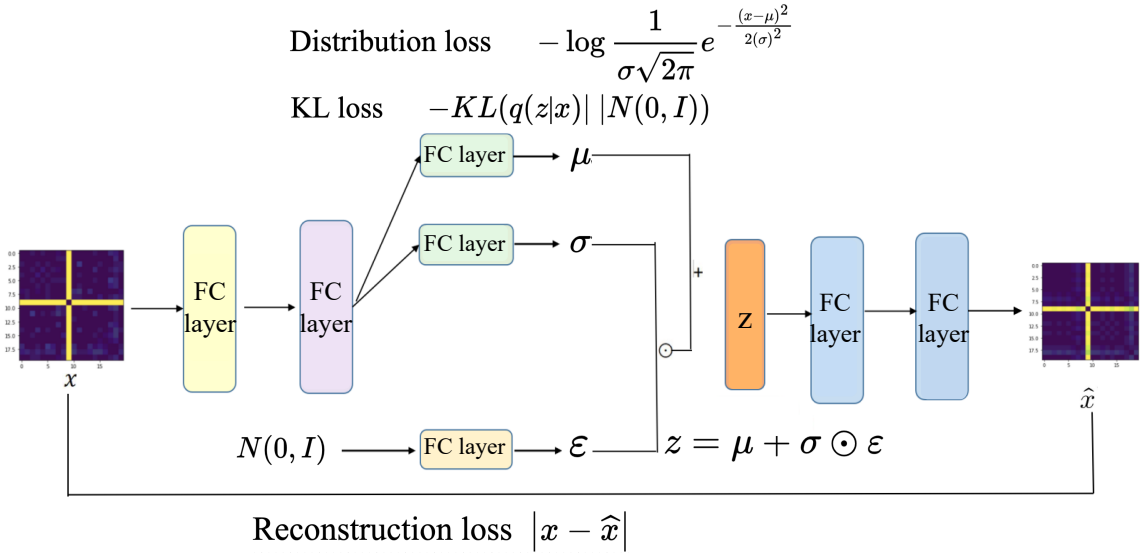


Figure 4.5: The architecture of dynamic VAE network in *Score-VAE*.

tance of $q(X_j^{'}$ and $p(X_j)$, where $q(X_j^{'})$ is the distribution of reconstruction data point $X_j^{'}$ and $p(X_j)$ is the distribution of original data point X_j.

4.5 Test Procedure of *Score-VAE*

Hamming test: During the testing period, *Score-VAE* works together with the existing anomaly detection model. The test samples first pass through the anomaly detection model, which raises alarms whenever anomalies are detected. Then the anomalies are passed to the global root cause analysis model for root cause analysis.

Fig 4.6 uses an example to illustrate the flow of our test scheme, which includes 5 steps.

- *Step-0:* convert the anomaly test data into image samples using the preprocessing methods described in Section 4.4.1.
- *Step-1* (refer to (1) in Fig. 4.6): pass the test image samples to the root cause analysis system.
- *Step-2* (refer to (2) in Fig. 4.6): reconstruct test image samples with the global root cause analysis model for each scenario and output reconstructed images.

- *Step-3* (refer to (3) in Fig. 4.6): convert each reconstructed image into a binary image. The basic idea is to calculate the average of all pixel values in an image. Set a pixel to 1 if it is greater than the mean, and 0 otherwise.

- *Step-4* (refer to (4) in Fig. 4.6): calculate the Hamming distance between the reconstructed binary image of each scenario and the binary image of the original test sample. The Hamming distance [148] represents the number of pixels of two images at which their corresponding values are different.

- *Step-5* (refer to (5) in Fig. 4.6): compare the Hamming distance between the reconstructed binary image and the original binary image. The scenario that reconstructs the binary image with the smallest Hamming distance will be treated as the root cause of the anomaly sample.

In each group, if the proportion of samples caused by root cause c is greater than the root cause threshold γ, the root cause c is considered to be one of the root causes for that group. Finally, the global VAE root cause scoring system outputs its root cause score for the group,

$$s = \begin{bmatrix} s^1 & s^2 & \dots & s^C \end{bmatrix}^T, \tag{4.1}$$

where s^c $(1 \leq c \leq C)$ denotes the score that the anomaly is caused by the root cause c. The score s^c is set to 1 if c is one of the anomaly root causes of the group, otherwise set to 0.

Algorithm 6: Dynamic VAE network

Input: Training set $\mathbf{X}$, $(a_{rec,kl}, a_{dis})$, $[h^{min}_{rec,kl}, h^{max}_{rec,kl}]$, $[h^{min}_{dis}, h^{max}_{dis}]$, mini-batch size J

Output: Optimal parameters θ^* for the VAE model

Randomly initialize the parameters of the VAE model

repeat

  for *Batch* $b \leftarrow 1$ **to** B **do**

    $\mathcal{L}^{rec}(b) = \sum_{j=0}^{J-1} |X_j(b) - X_j^{'}(b)|$

    $\mathcal{L}^{kl}(b) = \sum_{j=0}^{J-1} \frac{1}{2} * (2log(\sigma_j(b)) + 1 - \sigma_j^2(b) - \mu_j^2(b))$

    $\mathcal{L}^{dis}(b) = \sum_{j=0}^{J-1} -log(\frac{1}{\sigma_j(b)\sqrt{2\pi}} e^{-\frac{(X_j(b)-\mu_j(b))^2}{2\sigma_j^2(b)}})$

    $v_{rec,kl} = a_{rec,kl}$

    $v_{dis} = a_{dis}$

    // Check loss weight range

    $v_{rec,kl}$ = Compare($a_{rec,kl}$, $h^{min}_{rec,kl}$, $h^{max}_{rec,kl}$)

    v_{dis} = Compare(a_{dis}, h^{min}_{dis}, h^{max}_{dis})

    // Use .detach() to stop gradient during backward

    $v_{rec,kl} = a_{rec,kl} + (v_{rec,kl} - a_{rec,kl}).detach()$

    $v_{dis} = a_{dis} + (v_{dis} - a_{dis}).detach()$

    $w_{rec,kl} = \frac{1}{2 * v^2_{rec,kl}}$

    $w_{dis} = \frac{1}{2 * v^2_{dis}}$

    $\mathcal{L}^{SUM}(b) = w_{rec,kl}(\mathcal{L}^{rec}(b) + \mathcal{L}^{kl}(b)) + C_1 * w_{dis} * \mathcal{L}^{dis}(b) + C_2 * log(v_{rec,kl} * v_{dis})$

    Backward $\mathcal{L}^{SUM}(b)$ to update the model parameters and loss weights

  end

until *Convergence*;

return *Optimal model parameters* θ^**;*

Function `Compare`(a, min, max)**:**

  result;

  if $a \leq min$ **then**

    result = min;

  else if $a \geq max$ **then**

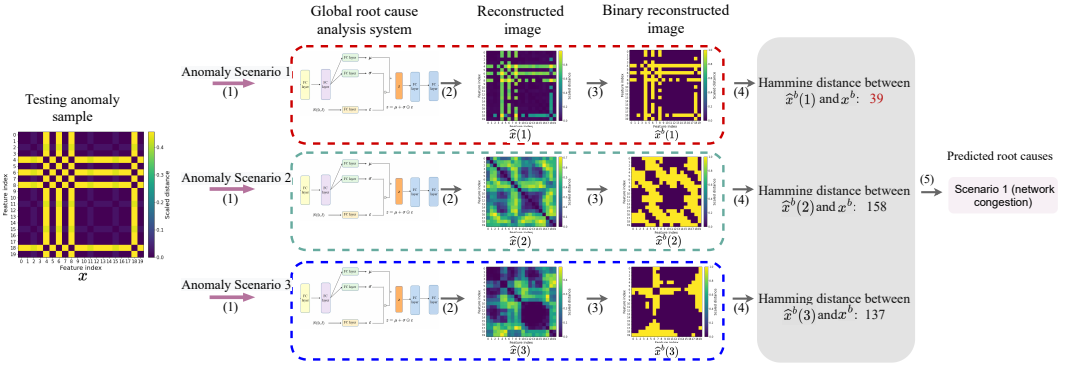
    result = max;

  else

    result = a;

  end

  return result;



(1) Pass the test image samples to the root cause analysis system.

(2) Reconstruct test image samples with the global root cause analysis model for each scenario and output reconstructed images.

(3) Convert each reconstructed image into a binary image.

(4) Calculate the Hamming distance between the reconstructed binary image of each scenario and the binary image of original test sample.

Figure 4.6: Hamming distance between the original binary image and the reconstructed binary image in the network congestion test example.

4.6 Further discussion: what are the main advantages of *Score-VAE* for root cause analysis?

Better generalization capability

Score-VAE have better generalization capabilities. since it relies on the reconstruction effect of the different scenario's global VAE root cause scoring models on the test sample to judge its category, rather than use a single model to learn rules directly from the mapping relationship of feature label pairs.

Lifelong-learning

Additionally, this training scheme makes the global models of different scenarios in the root cause analysis system independent of each other. Therefore, the process of training a global model for a newly observed scene does not render existing global models temporarily unusable.

Privacy-preserve

The privacy of local users can be preserved for the following reasons: 1. the processing steps make the actual inputs of the local VAE network only contain the distance relations of the scaled symbols in the original samples. 2. only the decoder parts are uploaded to the server. 3. the data samples used to train the global model are generated from random noise.

The negative impacts brought by the randomness of noise and the uncertainty of adding noise methods can be addressed due to: 1. we employ noise that has a similar distribution to the latent variable. 2. the generative ability of the VAE network provides an opportunity to build a bridge between random noise and training data. 3. the dynamic VAE network enables the trained model to reconstruct not only the sample pixel values, but also its distribution. 4. the hamming distance can reduce the distance of matching pixels and enlarge the distance of unmatched pixels by converting the reconstructed image to a binary image. Then only the incorrect data bits flipped in the binary image will be counted.

4.7 Experimental Evaluation

We perform a series of experiments to illustrate the benefits of *Score-VAE* by answering the following questions. **RQ1**: can *Score-VAE* correctly identify the causes that trigger the alarms from the anomaly detection model? **RQ2**: how does each component of *Score-VAE* contribute to the overall performance? **RQ3**: how does the performance of *Score-VAE* compare to the baseline?

4.7.1 Data preparation

The false alarms can be caused by attacks or the temporal-spatial dynamics of the network environment. However, we did not find any existing IoT system datasets that simultaneously contained all kinds of traffic. Hence, we built an IoT testbed with smart cameras (model DLink DCS-932LB) to emulate various IoT traffic scenarios. The cameras are connected to the Internet via a gateway (as shown in Fig. 4.1), and we use sniffer software to capture the data package going through the gateway via port mirroring.

We collected both malicious data and benign data. The malicious data include data flows under different Mirai attack [19] stages: loading, scanning, and DDoS attack. Benign data includes data flow under non-attack conditions, including different network configurations (e.g., different MTU values) and different network traffic loads (e.g., temporary network congestion). The details are as follows:

- Set_1*:* Mirai data includes malicious traffic during the Mirai loading phase. Specifically, Mirai infects a smart camera, turning it into a remotely controlled "zombie." This set includes 64101 samples.

- Set_2: Mirai data includes malicious traffic during the Mirai scanning stage. A bot sends a small amount of data to declare its connection with the command-and-control (CnC[2]) server and the CnC server sends the bots some commands. This set includes 18494 samples.

- Set_3: Mirai data includes malicious traffic during the distributed denial of service (DDoS) attack stage. A bot attacks a selected target by generating a large amount of fake data. This set includes 20310 samples.

- Set_4: Normal data includes benign traffic when the cameras are in normal working mode and the local network traffic load from other machines is light. This set includes 20728 samples.

- Set_5: Network configuration change data includes benign traffic when network configuration changes. We changed the TCP maximum segment size (MSS) to different values to test this scenario. Note that TCP may dynamically change MSS to suit the path MTU (PMTU). This set includes 34971 samples.

- Set_6: Network congestion data includes traffic in the presence of network congestion. Specifically, we captured live video data from the cameras while other local machines are streaming Internet videos. In this case, we observed frequent packet re-transmissions and losses, which manifest the network congestion. This set includes 20102 samples.

We divide each data set Set_i ($1 \leq i \leq 6$) into training set $Train_i$ (80%), validation set $Validation_i$ (10%) and testing set $Test_i$ (10%).

[2]An attacker-controlled computer called the CnC server is used to issue commands to systems infected with the Mirai malware. Since we have control of the Mirai bots, we configure the IP address of the CnC server to our controlled machine to avoid attacking any machine beyond our lab.

Table 4.2: Overall performance of *Score-VAE*

Test traffic	No.	Actual root cause	Probability distribution of identified root causes (%)				Test accuracy (%)
			Network congestion	Network configuration change	Mirai attack	Others	
Malicious	1	Mirai (loading)	1.49	0	98.51	0	98.51
	2	Mirai (scanning)	0	0	100	0	100
	3	Mirai (DDoS attack)	0	0	100	0	100
Benign	4	Network configuration change	0	99.05	0	0.95	99.05
	5	Network congestion	100	0	0	0	100

4.7.2 Answer to RQ1

We consider three anomaly scenarios: network configuration changes (scenario 1), Mirai attacks (scenario 2), and network congestion (scenario 3). Based on the training method introduced in Section 4.4, we use $Train_5$ to train the global model of scenario 1, $Train_2$ to train the global model of scenario 2, and $Train_6$ to train the global model of scenario 3. The data of $Test_1 \sim Test_6$ are used for testing.

We use the following hyper-parameters. The batch size is 100 and learning rate is $0.5 * e^{-3}$. The values of C_1 and C_2 are set to 0.2 and 1000 respectively. The min-max scaling range value for the Mirai attack scenario is set to $(0, 1)$. The min-max scaling range value for the network congestion scenario is set to $(-2, 2)$. The min-max scaling range value for the network configuration changes scenario is set to $(-4, 4)$. The root cause threshold γ is set to 0.2.

Experimental results. The overall performance of *Score-VAE* is shown in Table 4.2. The results show that ***Score-VAE* can successfully identify the root causes of alarms triggered by global anomaly detection, with an accuracy higher than 98%.**

4.7.3 Answer to RQ2

Score-VAE includes three critical components: joint noise, dynamic VAE, and Hamming test. To demonstrate the contribution of each component, we perform an ablation study. The *baseline* is the one that disables joint noise and uses the traditional

Table 4.3: Ablation study of components in *Score-VAE*

Joint noise	Dynamic VAE	Hamming test	Test accuracy improvement*					Privacy protection*
			Mirai (loading)	Mirai (scanning)	Mirai (DDoS attack)	Network configuration change	Network congestion	
✓	✓	✓	**1.49%↓**	–	–	**99.05%↑**	**100%↑**	**Yes**
✗	✓	✓	10.10%↓	–	–	–	90.47%↑	No (same as the *baseline*)
✓	✗	✓	100%↓	58.07%↓	–	100%↑	3.44%↑	Yes
✓	✓	✗	–	–	–	–	–	Yes

* The improvement is over the *baseline* that disables joint noise and uses traditional VAE network and L_1 loss. "–" means the same as the *baseline*.

VAE network and L_1 loss. Table. 4.3 shows the results.

Joint Noise

If we disable the joint noise, the system will lack adequate protection for privacy, as explained in Section 4.4.

Dynamic VAE

The second row of Table 4.3 indicates that dynamic VAE, working with the Hamming test, can significantly improve the accuracy of identifying network congestion. The main reason is that the dynamic loss function consists of traditional losses (reconstruction loss and Kl loss) and distribution loss. The distribution loss can help the VAE model learn the latent variables better and ensure that the posterior distribution of reconstructed data is close to the distribution of original data. In addition, the weights learned by the model can better capture the importance of different losses than average weights or manually adjusted weights.

Hamming test

Comparing the fourth row and the first row of Table 4.3, we find that the Hamming test can improve the accuracy of identifying network configuration and network congestion. The main reason is that Hamming distance can reduce the distance of

matching pixels and enlarge the distance of unmatched pixels by converting the reconstructed image to a binary image. Then only the incorrect data bits flipped in the binary image will be counted. Compared with other loss calculation methods such as L_1 loss and kl divergence, Hamming distance reduces the noise error value brought by the preprocessing and reconstruction process.

In summary, the ablation study shows that combining these three components can significantly improve the accuracy of identifying non-malicious root causes (e.g., network configuration change or network congestion), while achieving similar accuracy in identifying malicious root causes (only the accuracy for detecting Mirai loading is slightly reduced).

4.7.4 Answer to RQ3

As mentioned in Section 4.2, there are many potential supervised learning models suitable for root cause analysis problems (for example, a simple neural network/tree-based model). To better illustrate the necessity of our design score-VAE, we comprehensively compare *Score-VAE* with two baselines from two perspectives.

1. Baseline 1: this baseline uses a decision tree classifier [149] for root cause analysis.

2. Baseline 2: this baseline uses a 3-layer LSTM neural network [50] for root cause analysis.

Experimental results. The evaluation results are shown in Table. 4.4. From the perspective of test performance, *Score-VAE* can achieve the best performance (99.14% test accuracy). This may be due to our joint noise training scheme and Hamming test method enhancing the generalization ability of the model.

From the perspective of privacy preservation, in *Score-VAE*, only the decoder part of the local VAE generative model is uploaded to the server, and the server trains the global root cause scoring model by feeding noise to the decoder part to generate new samples. Due to the effective combination of federated learning architecture, preprocessing scheme, and joint noise training scheme, the data privacy of local users in using *Score-VAE* can be preserved.

Table 4.4: Comparison of *Score-VAE* with baseline

Method	Overall test accuracy	Privacy protection
Baseline 1	49.28%	No
Baseline 2	69.56%	No
Score-VAE	**99.14%**	**Yes**

4.8 Conclusion and Future Work

FL-based IoT anomaly detection systems may raise many false-positive alarms caused by the spatial-temporal dynamics in the network environment of the FL architecture, e.g., duplicate TCP packet retransmission due to temporary wireless link failures or network configuration changes. This paper tackled this challenge by developing a new root cause analysis system, called *Score-VAE*, which works as an add-on service to help existing FL-based IoT anomaly detection filter out false positives. *Score-VAE* uses the reconstruction and generation capabilities of the VAE network and integrates three key components, namely joint noise, dynamic VAE, and Hamming test, to ensure the good generalization, lifelong learning, collaboration, and privacy protection capacities required by root cause analysis of IoT systems. Evaluated with real-world IoT data, *Score-VAE* can correctly discover the anomaly's root causes while protecting the privacy of local users.

Score-VAE has the same inherent limitation as any VAE-based solution: its knowledge about root causes is learned from known scenarios. How to break this limitation is a challenging problem left for future work. Actually, it is still unclear whether or not it is possible to break this limitation. We expect *Score-VAE* can attract more attention and motivate more future research on root cause analysis.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this dissertation, we investigated three critical problems for the Internet of Things anomaly detection.

First, the IoT anomaly detection system needs to update the model quickly, either because of mislabelled samples or due to device firmware upgrades. To tackle this problem, we propose a novel machine unlearning method, termed *ViFLa*. *ViFLa* leverages the concept of virtual federated learning and can obtain efficiency and completeness of machine unlearning based on smart partition, enhanced class distribution weighted sum (ECDWS), and state transition ring. We thoroughly evaluate the performance of *ViFLa* using real-world trace data, covering not only the effectiveness of its individual components but also its benefit in different application scenarios. The evaluation result demonstrates that *ViFLa* can obtain similar accuracy of re-training from scratch with significant speedup. We also performed a theoretical analysis of the efficiency of ViFLa using SISA [64] as a baseline. Compared to the baseline, ViFLa can reduce the computational complexity and the amount of data affected by unlearned samples.

Second, in order to solve the non-IID problem in a federated learning-based anomaly detection system, we propose a new anomaly detection approach *ClusterFLADS* based on the knowledge of clustered federated learning. To address the non-IID problem in the temporal domain, *ClusterFLADS* takes advantage of the false predictions of inappropriate global models, together with knowledge of temperature scaling and catastrophic forgetting, to reveal distributional similarities between the

training data of different clusters and the test data. To improve the clustering speed, we introduce an efficient feature extraction scheme by exploiting the difference in the role each layer of a neural network plays in the learning process. We evaluate the performance of *ClusterFLADS* using real-world IoT trace data in various scenarios. The results show that *ClusterFLADS* can cluster clients accurately and efficiently, with a 100% true positive rate and no false positives over various data distributions.

Third, to cope with the diverse anomaly scenarios that may be encountered by federated learning-based IoT anomaly detection systems, we developed *Score-VAE*, a new framework to identify the root causes of anomalies based on a variational autoencoder (VAE) network. *Score-VAE* can work together with existing IoT anomaly detection systems built over the federated learning (FL) framework. To achieve lifelong learning, *Score-VAE* trains a separate global model for each abnormal scenario, so the intervention of new scenarios will not render the existing system unusable. To obtain better generalization and collaboration capacities required by the IoT systems, *Score-VAE* designed a privacy-preserve training scheme and a Hamming tests scheme based on the generation and reconstruction capacities of the variational autoencoder (VAE) network and the knowledge of hamming distance. To further improve model performance, we employ a VAE network with dynamic loss, which exploits knowledge of multi-task learning, stopping gradients and distributions. Evaluation results with real-world IoT trace data collected from different scenarios demonstrate *Score-VAE* can accurately discover the root causes of alarms triggered by the IoT anomaly detection systems and obtain better performance compared to the baselines.

5.2 Future work

The Internet of Things has penetrated every aspect of people's daily life at an unprecedented speed. We believe that the demand for IoT technologies and devices will continue to grow. The close connection with people's life and the lack of built-in security controls make the IoT intrusion detection critical in the IoT industry. This dissertation catered to this need and has several directions for further investigation.

First, in Chapter 2, accurate estimation of unlearning belief is critical for smart partition. we estimate the unlearning belief value (the likelihood that a sample is to be unlearned) of a sample based on the knowledge naive Bayes and k-fold validation. Such an estimation, while being verified effective in our evaluation, still has a large room for further improvement. In the future, we hope to collect more factors that

may affect the demand for samples to be unlearned, and take these factors into consideration in the design of our smart partition method. In this way, we may further improve the unlearning performance and adapt it to different application contexts.

Second, in Chapter 3, our evaluation results were based on the assumption that IoT devices belonging to the same type can be identified. However, this requires prior classification of IoT devices. In the future, we hope to extend our method that can automatically handle different types of devices in different network environments. In other words, we need to answer the question "how can we automatically cluster similar devices together and train a global detection model for each cluster without knowing the device type." This will pose more demanding requirements on clustering, training, and testing mechanisms.

Third, in Chapter 4, the goal of the *Score-VAE* framework is to improve the anomaly detection system by analyzing the root causes when the system raises alarms. From the perspective of network operators, the root causes are critical information to identify true network anomalies. While the inferred root causes can save time for network operators, network operators might still need to verify the root causes to ensure the safe operation of the network. Ideally, if the system can automatically verify the root causes filter out the false alarms in the first place, the work overhead of network operators can be further reduced. We are thus motivated to extend *Score-VAE* for this purpose by automatically filtering false alarms with causal correlation and contextual monitoring.

Bibliography

[1] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 756–767.

[2] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a" kneedle" in a haystack: Detecting knee points in system behavior," in *2011 31st international conference on distributed computing systems workshops*. IEEE, 2011, pp. 166–171.

[3] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[4] Wikipedia contributors, "Plagiarism — Wikipedia, the free encyclopedia," 2004, [Online; accessed 22-July-2004]. [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things

[5] [Online]. Available: https://www.industryweek.com/technology-and-iiot/automation/article/21955301/netsilicon-incwaltham-mass

[6] [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/internet-of-things-market-573.html?utm_source=globenewswire&utm_medium=Referral&utm_campaign=globenewswire

[7] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[8] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

[9] C. Mulvihill, J. Cooper, J. Pavey, and J.-P. Laake, "Remote consultations in primary care during the covid-19 pandemic: student perspectives," *Postgraduate medical journal*, vol. 98, no. e2, pp. e88–e89, 2022.

[10] T. Kalsoom, N. Ramzan, S. Ahmed, and M. Ur-Rehman, "Advances in sensor technologies in the era of smart factory and industry 4.0," *Sensors*, vol. 20, no. 23, p. 6783, 2020.

[11] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[12] Z. Wu, "Initial study on iot security architecture," *J. Strategy and decision-making research*, 2010.

[13] K. Zhao and L. Ge, "A survey on the internet of things security," in *2013 Ninth international conference on computational intelligence and security*. IEEE, 2013, pp. 663–667.

[14] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2002.

[15] [Online]. Available: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/

[16] A. Nascita, F. Cerasuolo, D. Di Monda, J. T. A. Garcia, A. Montieri, and A. Pescape, "Machine and deep learning approaches for iot attack classification," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–6.

[17] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future generation computer systems*, vol. 82, pp. 395–411, 2018.

[18] C. Grossi, *Algorithms for reinforcement learning*. Springer Nature, 2022.

[19] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[20] S. Moganedi, "Undetectable data breach in iot: Healthcare data at risk," in *17th European Conference on Cyber Warfare and Security*, vol. 2, 2018, p. 296.

[21] G. Cox, "Managing the risks of shadow iot," *Network Security*, vol. 2019, no. 1, pp. 14–17, 2019.

[22] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[23] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017.

[24] R. Martin, "The internet of things (iot)–removing the human element," *Infosec Writers*, vol. 28, p. 12, 2015.

[25] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[26] Z. Zhao, "Scalable iot network testbed with hybrid device emulation," *UVic Master's thesis*, 2022.

[27] B. Rios and J. Butts, "Security evaluation of the implantable cardiac device ecosystem architecture and implementation interdependencies," *WhiteScope, sl*, 2017.

[28] [Online]. Available: https://www.fox6now.com/news/felt-so-violated-milwaukee-couple-warns-hackers-are-outsmarting-smart-homes

[29] A. E. Omolara, A. Alabdulatif, O. I. Abiodun, M. Alawida, A. Alabdulatif, H. Arshad *et al.*, "The internet of things security: A survey encompassing unexplored areas and new insights," *Computers & Security*, vol. 112, p. 102494, 2022.

[30] [Online]. Available: https://firedome.io/blog/top-cyber-attacks-on-iot-devices-in-2021/

[31] [Online]. Available: https://unit42.paloaltonetworks.com/mirai-variant-v3g4/

[32] T. Borgohain, A. Borgohain, U. Kumar, and S. Sanyal, "Authentication systems in internet of things," *arXiv preprint arXiv:1502.00870*, 2015.

[33] N. Mhaskar, M. Alabbad, and R. Khedri, "A formal approach to network segmentation," *Computers & Security*, vol. 103, p. 102162, 2021.

[34] P. Spadaccino and F. Cuomo, "Intrusion detection systems for iot: opportunities and challenges offered by edge computing," *arXiv preprint arXiv:2012.01174*, 2020.

[35] H. Shahriar and W. Bond, "Towards an attack signature generation framework for intrusion detection systems," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2017, pp. 597–603.

[36] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN computer science*, vol. 2, no. 3, p. 160, 2021.

[37] V. Nasteski, "An overview of the supervised machine learning methods," *Horizons. b*, vol. 4, pp. 51–62, 2017.

[38] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu, "Unsupervised learning based on artificial neural network: A review," in *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. IEEE, 2018, pp. 322–327.

[39] Z.-H. Zhou and Z.-H. Zhou, "Semi-supervised learning," *Machine Learning*, pp. 315–341, 2021.

[40] J. MacQueen, "Classification and analysis of multivariate observations," in *5th Berkeley Symp. Math. Statist. Probability*, 1967, pp. 281–297.

[41] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[42] C. M. Bishop, "Pattern recognition," *Machine learning*, vol. 128, no. 9, 2006.

[43] B. Dong, Q. Shi, Y. Yang, F. Wen, Z. Zhang, and C. Lee, "Technology evolution from self-powered sensors to aiot enabled smart homes," *Nano Energy*, vol. 79, p. 105414, 2021.

[44] Z. Sun, M. Zhu, Z. Zhang, Z. Chen, Q. Shi, X. Shan, R. C. H. Yeow, and C. Lee, "Artificial intelligence of things (aiot) enabled virtual shop applications using self-powered sensor enhanced soft robotic manipulator," *Advanced Science*, vol. 8, no. 14, p. 2100230, 2021.

[45] C.-J. Chen, Y.-Y. Huang, Y.-S. Li, C.-Y. Chang, and Y.-M. Huang, "An aiot based smart agricultural system for pests detection," *IEEE Access*, vol. 8, pp. 180 750–180 761, 2020.

[46] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[47] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[48] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[49] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[50] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[51] M. Jordan, "Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986," California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, Tech. Rep., 1986.

[52] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[53] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 463–480.

[54] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[55] N. Shoham, T. Avidor, A. Keren, N. Israel, D. Benditkis, L. Mor-Yosef, and I. Zeitak, "Overcoming forgetting in federated learning on non-iid data," *arXiv preprint arXiv:1910.07796*, 2019.

[56] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, vol. 3, p. 3, 2018.

[57] W. Zhang, X. Wang, P. Zhou, W. Wu, and X. Zhang, "Client selection for federated learning with non-iid data in mobile edge computing," *IEEE Access*, vol. 9, pp. 24 462–24 474, 2021.

[58] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, "Personalized cross-silo federated learning on non-iid data." in *AAAI*, 2021, pp. 7865–7873.

[59] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 586–19 597, 2020.

[60] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1283–1297.

[61] I. Csiszár, "I-divergence geometry of probability distributions and minimization problems," *The annals of probability*, pp. 146–158, 1975.

[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[63] S. Zhang, Y. Tay, L. Yao, and A. Sun, "Next item recommendation with self-attention," *arXiv preprint arXiv:1808.06414*, 2018.

[64] L. Bourtoule, V. Chandrasekaran, C. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," *arXiv preprint arXiv:1912.03817*, 2019.

[65] J. Fan, K. Wu, Y. Zhou, Z. Zhao, and S. Huang, "Fast model update for iot traffic anomaly detection with machine unlearning," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[66] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment," *arXiv preprint arXiv:1906.06629*, 2019.

[67] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 8, pp. 3710–3722, 2020.

[68] B. Liu, Y. Guo, and X. Chen, "Pfa: Privacy-preserving federated adaptation for effective model personalization," in *Proceedings of the Web Conference 2021*, 2021, pp. 923–934.

[69] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao, and Y. Zhao, "Resource-efficient federated learning with hierarchical aggregation in edge computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[70] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[71] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 1321–1330.

[72] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[73] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[74] R. Ratcliff, "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions." *Psychological review*, vol. 97, no. 2, p. 285, 1990.

[75] D. Mourtzis, M. Doukas, and N. Milas, "A knowledge-based social networking app for collaborative problem-solving in manufacturing," *Manufacturing Letters*, vol. 10, pp. 1–5, 2016.

[76] G. Fix, "An expert system for test failure root cause discovery," in *2010 Seventh International Conference on Information Technology: New Generations*. IEEE, 2010, pp. 1014–1019.

[77] O. V. Mamoutova, M. B. Uspenskiy, A. V. Sochnev, S. V. Smirnov, and M. V. Bolsunovskaya, "Knowledge based diagnostic approach for enterprise storage systems," in *2019 IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE, 2019, pp. 207–212.

[78] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[79] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[80] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[81] S. Marchal, M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan, "Audi: Towards autonomous iot device-type identification," *IEEE Journal on Selected Areas in Communications (JSAC) on Artificial Intelligence and Machine Learning for Networking and Communications*, 2019.

[82] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[83] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, vol. 2, 2016.

[84] C.-T. Chu, S. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Ng, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems*, vol. 19, pp. 281–288, 2006.

[85] A. Parmisano, S. Garcia, and M. Erquiaga, "Stratosphere laboratory. a labeled dataset with malicious and benign iot network traffic," January, 2020.

[86] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9304–9312.

[87] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *NIPs*, vol. 2. Citeseer, 1989, pp. 598–605.

[88] A. Ginart, M. Y. Guan, G. Valiant, and J. Zou, "Making ai forget you: Data deletion in machine learning," *arXiv preprint arXiv:1907.05012*, 2019.

[89] T. Baumhauer, P. Schöttle, and M. Zeppelzauer, "Machine unlearning: Linear filtration for logit-based classifiers," *arXiv preprint arXiv:2002.02730*, 2020.

[90] N. Aldaghri, H. Mahdavifar, and A. Beirami, "Coded machine unlearning," *IEEE Access*, 2021.

[91] J. Brophy and D. Lowd, "Machine unlearning for random forests," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1092–1104.

[92] T. Bayes, "Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s," *Philosophical transactions of the Royal Society of London*, no. 53, pp. 370–418, 1763.

[93] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[94] A. McCallum, "Graphical models, lecture2: Bayesian network represention," *PDF). Retrieved*, vol. 22, 2019.

[95] S. Neel, A. Roth, and S. Sharifi-Malvajerdi, "Descent-to-delete: Gradient-based methods for machine unlearning," *arXiv preprint arXiv:2007.02923*, 2020.

[96] Z. Harry, "The optimality of naive bayes," in *FLAIRS2004 conference*, 2004.

[97] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.

[98] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[99] G. H. Rosenfield and K. Fitzpatrick-lins, "A coefficient of agreement as a measure of thematic classification accuracy." *Photogrammetric Engineering and Remote Sensing*, vol. 52, pp. 223–227, 1986.

[100] M. Ojala and G. C. Garriga, "Permutation tests for studying classifier performance." *Journal of machine learning research*, vol. 11, no. 6, 2010.

[101] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[102] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Montreal, Canada, 1995, pp. 1137–1145.

[103] S. Geisser, "Predictive inference, volume 55 of monographs on statistics and applied probability," 1993.

[104] F. Mosteller and J. W. Tukey, "Data analysis, including statistics," *Handbook of social psychology*, vol. 2, pp. 80–203, 1968.

[105] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.

[106] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 141–159.

[107] Z. Qadir, K. N. Le, N. Saeed, and H. S. Munawar, "Towards 6g internet of things: Recent advances, use cases, and open challenges," *ICT Express*, 2022.

[108] G. Nebbione and M. C. Calzarossa, "Security of iot application layer protocols: Challenges and findings," *Future Internet*, vol. 12, no. 3, p. 55, 2020.

[109] P. Jain, M. Gyanchandani, and N. Khare, "Big data privacy: a technological perspective and review," *Journal of Big Data*, vol. 3, no. 1, pp. 1–25, 2016.

[110] K. You, Z. Kou, M. Long, and J. Wang, "Co-tuning for transfer learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[111] N. Durrani, H. Sajjad, and F. Dalvi, "How transfer learning impacts linguistic knowledge in deep nlp models?" *arXiv preprint arXiv:2105.15179*, 2021.

[112] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*. PMLR, 2014, pp. 647–655.

[113] N. Tripuraneni, M. Jordan, and C. Jin, "On the theory of transfer learning: The importance of task diversity," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7852–7862, 2020.

[114] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[115] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[116] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[117] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[118] T. Calinski, "A dendrite method for cluster analysis," *Communication in statistics*, vol. 3, pp. 1–27, 1974.

[119] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.

[120] L. Van Der Maaten and G. Hinton, "Visualizing high-dimensional data using t-sne. journal of machine learning research," *J Mach Learn Res*, vol. 9, no. 26, p. 5, 2008.

[121] H. F. Atlam and G. B. Wills, "Iot security, privacy, safety and ethics," in *Digital twin technologies and smart cities*. Springer, 2020, pp. 123–149.

[122] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu, "Security vulnerabilities of internet of things: A case study of the smart plug system," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1899–1909, 2017.

[123] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517300802

[124] C. M. Roelofs, M.-A. Lutz, S. Faulstich, and S. Vogt, "Autoencoder-based anomaly root cause analysis for wind turbines," *Energy and AI*, vol. 4, p. 100065, 2021.

[125] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 2004, pp. 36–43.

[126] R. M. A. Velásquez and J. V. M. Lara, "Root cause analysis improved with machine learning for failure analysis in power transformers," *Engineering Failure Analysis*, vol. 115, p. 104684, 2020.

[127] A. Lokrantz, E. Gustavsson, and M. Jirstrand, "Root cause analysis of failures and quality deviations in manufacturing using machine learning," *Procedia Cirp*, vol. 72, pp. 1057–1062, 2018.

[128] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 326–327, 1995.

[129] Q. Chen, C. Xiang, M. Xue, B. Li, N. Borisov, D. Kaarfar, and H. Zhu, "Differentially private data generative models," *arXiv preprint arXiv:1812.02274*, 2018.

[130] M. Z. Islam and L. Brankovic, "Privacy preserving data mining: A noise addition framework using a novel clustering technique," *Knowledge-Based Systems*, vol. 24, no. 8, pp. 1214–1223, 2011.

[131] T. T. Nguyen, T. T. Huynh, P. L. Nguyen, A. W.-C. Liew, H. Yin, and Q. V. H. Nguyen, "A survey of machine unlearning," *arXiv preprint arXiv:2209.02299*, 2022.

[132] S. Chockalingam and V. Katta, "Developing a bayesian network framework for root cause analysis of observable problems in cyber-physical systems," in *2019 IEEE Conference on Information and Communication Technology*. IEEE, 2019, pp. 1–6.

[133] J. Wang, Z. Yang, J. Su, Y. Zhao, S. Gao, X. Pang, and D. Zhou, "Root-cause analysis of occurring alarms in thermal power plants based on bayesian networks," *International Journal of Electrical Power & Energy Systems*, vol. 103, pp. 67–74, 2018.

[134] L. Yang, J. Zhang, F. Deng, and J. Chen, "Sensor fault diagnosis based on on-line random forests," in *2016 35th Chinese Control Conference (CCC)*. IEEE, 2016, pp. 4089–4093.

[135] A. Detzner, R. Rückschloß, and M. Eigner, "Root-cause analysis with interactive decision trees," in *2020 24th International Conference Information Visualisation (IV)*. IEEE, 2020, pp. 322–327.

[136] N. G. Lo, J.-M. Flaus, and O. Adrot, "Review of machine learning approaches in fault diagnosis applied to iot systems," in *2019 International Conference on Control, Automation and Diagnosis (ICCAD)*. IEEE, 2019, pp. 1–6.

[137] S. S. Tayarani-Bathaie, Z. S. Vanini, and K. Khorasani, "Dynamic neural network-based fault diagnosis of gas turbine engines," *Neurocomputing*, vol. 125, pp. 153–165, 2014.

[138] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[139] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-iid data: A survey," *Neurocomputing*, vol. 465, pp. 371–390, 2021.

[140] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Survey on incremental approaches for network anomaly detection," *arXiv preprint arXiv:1211.4493*, 2012.

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

[142] S. Lin, R. Clark, R. Birke, S. Schönborn, N. Trigoni, and S. Roberts, "Anomaly detection for time series using vae-lstm hybrid model," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 4322–4326.

[143] M. Cerliani, "MEDIUM notebook," 2021. [Online]. Available: https://github.com/cerlymarco/MEDIUM_NoteBook

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

[145] D. Park, Z. Erickson, T. Bhattacharjee, and C. C. Kemp, "Multimodal execution monitoring for anomaly detection during robot manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 407–414.

[146] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[147] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491.

[148] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[149] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[150] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[151] R. G. Bartle and D. R. Sherbert, *Introduction to real analysis*. Wiley New York, 2000, vol. 2.

Appendix A

Additional Information

LSTM Summation Form: LSTM is a type of recurrent neural network (RNN) architecture [150]. Each LSTM neuron (shown in a colored box in Fig. A.1) has four inputs: the original input x, the input gate that controls if x could be written into the memory cell, the output gate that decides if the value of the current neuron is accessible to the other neurons, and the forget gate that decides what time to forget the content in the memory cell. We train an LSTM of three layers with mini-batches (sample size = 128), and the dimension of its input feature is 20. Following [150], the network is modeled as:

$$
\begin{aligned}
f_t &= \sigma(W_f * [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i * [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C * [h_{t-1}, x_t] + b_C) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \\
o_t &= \sigma(W_o * [h_{t-1}, x_t] + b_o) \\
h_t &= o_t \circ \tanh C_t
\end{aligned}
$$

where $W_f, W_i, W_C, W_o \in \mathbb{R}^{128\times(20+128)}$, $b_f, b_i, b_C, b_o \in \mathbb{R}^{128}$, 128 is the number of hidden units, and 20 is the dimension of the input feature. Each parameter state consists of weights and biases of three LSTM layers and two fully connected layers. The initial parameter state of the network θ_{S_0} is the parameter state stored in node n_0 of the state transition ring in Fig. 2.5. At each state, a mini-batch (128 samples) $batch_b$ is input to the network and estimates the batch loss if $b = j$ (refer to Eq. (2.3)). We backward propagate this batch loss and calculate the gradient $g^i_{\theta_{S_0}(batch_b)}$ for each

parameter i and use this gradient as the mapping result. The summation form of the LSTM network is the sum of mapping results of all the batches at state θ_{S_j}, i.e., $\sum_b f_{\theta_{S_j}}(batch_b)$.

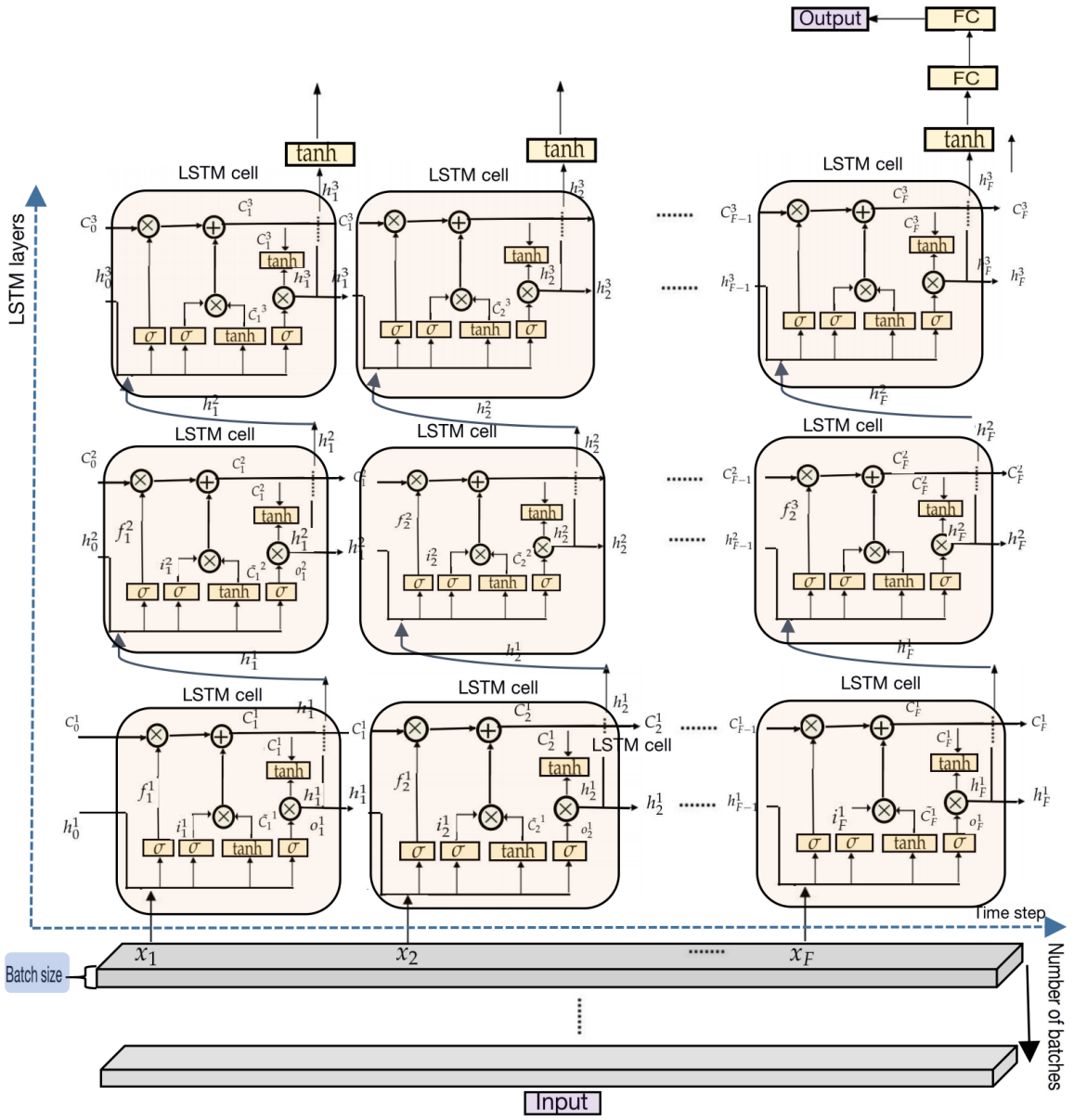


Figure A.1: Structure of LSTM.

Proof of the convergence of KL-attention: We provide a sketch proof. The proof is based on the concept of Cauchy sequence. The vector update rule defined by Eq. (2.7) is equivalent to a mapping function $\mathcal{T}(v_j) = \sum_{i=1}^{K} \sigma(\mathcal{R}(j,i))v_i$. We use $||.||$ to denote the vector norm. For $\forall j \in \{1, \ldots, K\}$, we can calculate the difference

between v_j and $\mathcal{T}(v_j)$ as:

$$||v_j - \mathcal{T}(v_j)|| = ||v_j - \sum_{i=1}^{K} \sigma(\mathcal{R}(j,i))v_i||$$
$$= || \sum_{i\neq j, i=1}^{K} \sigma(\mathcal{R}(j,i))(v_j - v_i)||$$

We slightly abuse the notation by using $\boldsymbol{v}$ to represent the set of vectors $\boldsymbol{v} = \{v_1, v_2, \ldots, v_K\}$ and $\mathcal{T}(\boldsymbol{v})$ to represent the set vectors $\mathcal{T}(\boldsymbol{v}) = \{\mathcal{T}(v_1), \mathcal{T}(v_2), \ldots, \mathcal{T}(v_K)\}$. We define the distance between $\boldsymbol{v}$ and $\mathcal{T}(\boldsymbol{v})$ as: $d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) = \sum_{j=1}^{K} ||v_j - \mathcal{T}(v_j)|| = \sum_{j=1}^{K} || \sum_{i\neq j, i=1}^{K} \sigma(\mathcal{R}(j,i))(v_j - v_i)||$.
Similarly, we can obtain the distance $d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v}))$ as follows:

$$d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v}))$$
$$= \sum_{j=1}^{K} || \sum_{i\neq j, i=1}^{K} \sigma(\mathcal{R}'(j,i))(\mathcal{T}(v_j) - \mathcal{T}(v_i))||$$

where $\mathcal{T}^2$ means applying the mapping function again, i.e., the second-round update with Eq. (2.7). The mapping function of a vector is the weighted average of all the vectors, and the similarity of the vector and other vectors will become smaller after mapping $\mathcal{T}$. So the distance $d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v})) \leq d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))$. There must exists a number $a (0 < a < 1)$ that makes $d(\mathcal{T}(\boldsymbol{v}), \mathcal{T}^2(\boldsymbol{v})) \leq a * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))$.

By induction and by triangle inequality, we have $d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) \leq d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^{n+1}(\boldsymbol{v})) + \ldots + d(\mathcal{T}^{m-1}(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) \leq a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) + a^{n+1} * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) + \ldots + a^{m-n-1} * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) = a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) * \sum_{k=0}^{m-n-1} a^k$. So $d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) \leq a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) * \sum_{k=0}^{n-m-1} a^k = a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v})) * \frac{a^{m-n}}{1-a} < \frac{a^n * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))}{1-a}$ for all $n < m$. For any $\epsilon > 0$, we can choose an N satisfying $\frac{a^N * d(\boldsymbol{v}, \mathcal{T}(\boldsymbol{v}))}{1-a} < \epsilon$. Then for $N \leq n \leq m$, $d(\mathcal{T}^n(\boldsymbol{v}), \mathcal{T}^m(\boldsymbol{v})) < \epsilon$.

So **the sequence $\{\boldsymbol{v^n} = \mathcal{T}^n(\boldsymbol{v})\}_{n=1}^{\infty}$ is a Cauchy sequence.** Convergence is proved due to the Bolzano–Weierstrass theorem [151] that "each bounded sequence in $\mathbb{R}^n$ has a convergent sub-sequence", and a Proposition [151] that "if a sub-sequence of a Cauchy sequence converges to x, then the sequence itself converges to x."

Appendix B

List of abbreviations

The following table describes the significance of various abbreviations utilized in this dissertation. The page indicates where the abbreviation first appears.

Table B.1: The list of abbreviations

Abbreviation	**Meaning**	**Page**
KL	Kullback–Leibler	16
IID	independently and identically distributed	14
Non-IID	non-independently and identically distributed	13
IoT	Internet of Things	12
ML	machine learning	12
FL	federated learning	12
CFL	clustered federated learning	17
ECDWS	enhanced class distribution weighted sum	16
SQ	statistical query	16
TPR	true positive rate	50
FPR	false positive rate	50
LSTM	Long Short Term Memory	12
GRU	Gated Recurrent Units	12
PCA	principal component analysis	63
TSNE	T-distributed Stochastic Neighbor Embedding	75
CnC	command and control	107
VAE	variational autoencoder	15
RQ	research question	69
ViFLa	*virtual* federated learning approach	14
ClusterFLADS	clustered federated learning-based anomaly detection system	18