University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Intelligent Analysis of System Calls to Detect Cyber Attacks Affecting Spectrum Data Integrity in IoT Sensors

Chao Feng
Zürich, Switzerland
Student ID: 19-763-796

Supervisor: Dr. Alberto Huertas Celdrán, Muriel Figueredo Franco
Date of Submission: January 14, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlesasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

In den letzten Jahrzehnten haben sich IoT-Technologien rasant entwickelt. Milliarden von Geräten greifen über drahtlose Netzwerke auf das Internet zu und erleichtern das Leben der Menschen, verbrauchen aber gleichzeitig wertvolle Funkfrequenzen. Zur Optimierung des Funkfrequenzspektrums werden Crowdsensing-basierte Netzwerke zur Überwachung des Funkfrequenzspektrums vorgeschlagen, die aus verteilten IoT-Sensoren bestehen, die zusammenarbeiten, um weltweit Funkfrequenzdaten zu sammeln, zu übertragen und zu verarbeiten. Diese IoT-Sensoren mit begrenzten Ressourcen sind jedoch äußerst anfällig für Cyberangriffe, die die Integrität der Funkfrequenzspektrumsdaten gefährden und den Betrieb der gesamten Plattform beeinträchtigen.

Einerseits gilt das auf maschinellem Lernen basierende Fingerprinting des Geräteverhaltens zur Identifizierung von Cyberangriffen als vielversprechend. Andererseits sind die Daten über das Geräteverhalten sehr sensibel, und der Datenschutz muss berücksichtigt werden. Vor diesem Hintergrund wird in dieser Arbeit ein auf föderativem Lernen basierendes System zur Erkennung von IoT-Netzwerkangriffen unter Verwendung von Verhaltensdaten von Systemaufrufen vorgeschlagen. Dieser Ansatz erreicht sowohl den Schutz der Privatsphäre als auch eine effektive Identifizierung von Cyberangriffen durch seine einzigartige Trainingsstrategie, d.h. er gibt nur die Modellparameter, nicht aber die Trainingsdaten weiter. Nach einem systematischen Vergleich wählt diese Arbeit den am besten geeigneten Ansatz zur Merkmalsextraktion und den lokalen Identifikationsalgorithmus aus. Die Wirksamkeit und Zuverlässigkeit des vorgeschlagenen Modells wird anhand einer quantitativen Analyse in einer Vielzahl unterschiedlicher Szenarien nachgewiesen.

Abstract

Over the past few decades, IoT technologies have surged, with billions of devices accessing the Internet through wireless networks, bringing convenience to human lives while consuming the valuable wireless spectrum. To optimize the radio frequency spectrum, crowdsensing-based radio frequency spectrum monitoring networks are proposed, consisting of distributed IoT sensors that collaborate to collect, transmit, and process radio spectrum data worldwide. However, these IoT sensors with constrained resources are extremely vulnerable to cyberattacks that compromise the integrity of the radio frequency spectrum data and affect the operation of the entire platform.

On the one hand, Machine Learning-based device behavior fingerprinting for cyberattack identification is considered highly promising. On the other hand, the device behavior data is strongly sensitive, and its data privacy becomes an issue that has to be considered. Taking these into consideration, this thesis proposes a Federated Learning-based IoT network attack detection system using system calls behavioral data. This approach achieves both data privacy protection and effective identification of cyberattacks through its unique training strategy, i.e., sharing only model parameters but not the training data. After a systematic comparison, this thesis selects the most suitable feature extraction approach and local identification algorithm. The effectiveness and reliability of the proposed model is demonstrated by using quantitative analysis through a variety of different scenarios.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Alberto Huertas Celdrán for his sustained support and guidance during this master thesis. His excellent knowledge of data science and cybersecurity has been a great help and inspiration for this project.

Additionally, I want to thank Mr. Pedro Miguel Sánchez Sánchez for repeatedly providing his insights to this project and Mr. Muriel Franco for his valuable suggestions for this work.

Finally, I want to thank Prof. Dr. Burkhard Stiller for giving me the opportunity to carry out this master thesis at the Communication Systems Group. It is a pleasant experience to work on this project at the CSG.

Contents

Chapter 1

Introduction

With the development of Internet-of-Things (IoT) and wireless communication technologies, an increasing number of IoT devices are wirelessly connected to the internet, and by 2025, it is expected that 64 billion IoT devices will be online [40]. There is no doubt that these connected devices bring countless conveniences to human lives, but simultaneously they also get the problem of spectrum conflicts. Consequently, the overcrowded Radio Frequency (RF) bandwidth brings interference that affects the efficiency and quality of data transmission. To solve this interference, the crowdsensing RF monitoring platform became a reality, which is oriented to optimize spectrum usage and to detect illegal transmissions and cyberattacks [37]. The Crowdsensing RF monitoring platform uses distributed sensors to collect RF spectrum data, as an infrastructure for Cognitive Radio Networks (CRN), which can be used to balance RF spectrum usage and improve the efficiency of the wireless networks. Meanwhile, through the analysis of spectrum data, these crowdsensing platforms can also be used to identify illegal data transmission, detect cyberattacks, provide data support for the development of transmission technologies, etc.

Each of the previously mentioned tasks is highly reliant on the spectrum monitoring sensors being able to collect reliable data. Therefore, protecting the integrity of RF spectrum data is critical but also fraught with challenges. The most common cyberattacks against RF spectrum sensing include two groups, Primary User Emulation Attacks (PUEA) and Spectrum Sensing Data Falsification Attacks (SSDF), where SSDF directly compromises the integrity of the spectrum data [23]. To make the situation even more complicated, these RF spectrum monitoring platforms often use resource-constrained devices as their sensors, such as Raspberry Pis, and these devices are well-known vulnerable to insider attacks or malwares [29].

The behavioral data generated by the device during its execution, such as resource usage, or system calls, could be monitored to generate precise device behavioral fingerprinting. By combining with Machine Learning (ML) and Deep Learning (DL), approaches based on device behavioral fingerprinting are considered as one of the most promising cyberattacks detection methods [42]. Research has demonstrated that the device behavior fingerprinting approach using system calls has an advantage in the performance of identifying cyberattacks [11, 32]. Meanwhile, system calls data is extremely sensitive and may have severe consequences if leaked, and on the other hand, these crowdsourced sensors

come from a variety of holders, such that data privacy becomes an essential concern of the RF spectrum monitoring platform. Hence, a privacy-preserving approach is needed for SSDF attacks identification. Federated Learning (FL) is such a privacy-preserving framework that shares only the model but not the data. Therefore, this thesis aims to design and develop a FL-based privacy-preserving and effective SSDF cyberattack detection method to protect the IoT spectrum data integrity. This chapter describes the motivation for such an approach and the scope and structure of this thesis.

1.1 Motivation

The integration of device behavioral fingerprinting with system calls and ML techniques has been proved as a promising way to detect anomalies produced by attacks manipulating spectrum data [42]. However, the following challenges commonly appear when combining ML/DL technologies with system calls for detecting cyberattacks affecting RF spectrum data integrity:

- There is no analysis of different feature extraction approaches using system calls to detect various SSDF attacks. The current practice of using ML/DL and system calls for network attack detection mainly focuses on traditional areas rather than spectrum data integrity, and the types of attacks detected are mostly malware or DDoS attacks. There is a lack of research on the use of ML/DL and system calls to detect SSDF attacks.

- There is no work comparing the performance and cost of ML/DL-based solutions that use different feature extraction approaches of system calls. Current works focus on using one system calls feature extraction method to extract one specific feature and then combining them with ML/DL techniques to detect cyberattacks. Still, there is a lack of systematic comparison of different extraction approaches in performance and cost.

- No ML/DL-based solution is preserving the privacy of the devices fingerprints while detecting SSDF attacks. In a traditional ML/DL cyberattack detection pipeline, client system calls must be transmitted to the central server for model training, which does not consider potential privacy issues. However, the device system calls data is highly sensitive for client sensors, and data privacy must be concerned when designing the behavioral fingerprinting-based SSDF attack detection system.

- There is no comparison between the detection performance of traditional ML/DL and FL with system calls features. Since SSDF attack detection methods using FL are lacking, the comparison of the performance of FL with system calls and traditional ML/DL is also missing.

1.2 Description of Work

To improve these challenges, the main goal of this thesis is to design and develop a privacy-preserving SSDF cyberattack detection prototype system using the integration of system calls features and FL framework.

To reach that objective, the main contributions of this work include:

- The analysis of the state-of-the-act regarding the usage of system calls in resource-constrained devices to create accurate behavior fingerprints.
- The creation of a system calls monitoring module able to be executed in different IoT devices such as Raspberry Pi 3 and 4. Besides, datasets that contain both normal and eight SSDF attack system calls for each kind of device have been created.
- The design of the module for the intelligent cyberattacks detection system, which uses FL and system calls features in resource-limited spectrum sensors affected by SSDF attacks.
- The systematic comparison of the computational cost, including computation time, CPU and memory utilization, etc., of different system calls feature extraction approaches.
- The performance of a pool of experiments measuring the detection capabilities and resource consumption of different ML/DL algorithms detecting SSDF attacks, including training and testing time, resource utilization, and model size.
- The evaluation and analysis of FL for SSDF attack identification in different scenarios, including (*i*) local model in individual sensor, (*ii*) FL-Based training strategies with 33.33%, 66.67% and 100.00% known devices, and (*iii*) central model containing datasets from all types of devices.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, as background knowledge, RF spectrum data and its monitoring platforms are briefly described, as well as various types of cyberattacks against its data integrity. Also, this section briefly introduces common approaches using device behavioral fingerprinting for anomalous data detection. In the third chapter, a systematic summary of the research related to the detection of cyberattacks using system calls features is presented. In Chapter 4, a use-case analysis is used to summarize the requirements of a cyberattack identification system based on FL-based technology and system calls features in an IoT environment, and then the architecture of the system is proposed. In Chapter 5, the most suitable features and algorithms for the system are found through a multifaceted comparison and evaluation. In Chapter 6, the FL-based system is evaluated and tested in all aspects through different perspectives and scenarios to verify the effectiveness and stability of the system. Lastly, Chapter 7 concludes the thesis by summarizing the main findings of the thesis and also gives an outlook on future work.

Chapter 2

Background

This chapter presents necessary background knowledge and information for the concepts and methodologies used in this thesis. First, an introduction of the spectrum data is provided, along with an overview of security issues in its integrity. Second, a brief summary of device behavior fingerprinting is presented. Lastly, different approaches and techniques for anomaly detection are documented, including ML, DL, and FL frameworks.

2.1 Radio Frequency Spectrum

The Internet of Things is becoming a hot topic of research in academia and industry today, which aims to connect a wide variety of devices to the Internet to improve the environment, facilitate human lives, and increase the efficiency of the industry. Billions of sensors, controllers, and computing platforms connected via wireless and mobile networks in people's homes, offices, factories, and even vehicles [36]. Thanks to these increasingly intelligent devices, smart homes, smart offices, smart manufacturing, etc., are no more extended science fiction scenarios but increasingly actual facts happening around people. However, vast amounts of data are generated as a result, and countless bits consume the RF band. Therefore, there is an urgent need to develop methods to control and optimize RF use effectively.

In order to solve the above problems, researchers have proposed the method of spectrum analysis [23, 37]. The radio spectrum refers to the density distribution of radio signals of different frequencies at different times, and it can be utilized to indicate the degree to which radio signals of different frequencies are used in a specific spatial and temporal range [50]. If such data is accessible, it will be possible to analyze and optimize the wireless transmission network concisely and efficiently by the extent of RF utilization in the area [23]. Thus, it has become important to build an easily extensible, lightweight, and open radio spectrum data acquisition and processing platform [37].

2.1.1 Spectrum Monitoring Network

There is no doubt that the supply of RF resource is always limited, which leads to a conflict between the demand and supply. Although radio spectrum data could help to improve the efficiency of the communication systems, there is limited knowledge about the distribution of spectrum data in time and space. For a long time, a worldwide network for radio spectrum monitoring, collecting, and analyzing did not exist [37]. Therefore, a platform that can monitor RF usage with a view to controlling and optimizing spectrum usage is urgently needed.

The major issue in building such a system is cost, i.e., collecting as much data as possible on a larger scale using limited resources. Considering the balance between cost and benefit, the platform must meet several requirements:

- Low cost. That is, both the development of the platform itself and the collection of data, the cost should be kept at a low level.
- Easy to scale. In order to ensure that the geographic scope of data collection is as wide as possible, the platform's data collection tools need to be simple and easy to use to facilitate the geographic expansion of data.
- Openness. Different users may be interested in different parts of the spectrum. For example, indoor users may only be interested in frequency data for wireless LAN and cellular networks, while users from airports may be interested in frequencies related to aircraft communications. Therefore, the platform should also keep open towards the collection and use of data.
- Efficient. In handling the spectrum data from all over the world, the data processing capability of the platform should be efficient enough.

As a result, only a few spectrum collection platforms have been proposed by researchers so far. Microsoft Spectrum Observatory uses custom sensors to collect data [55], but this platform has not been promoted on a large scale because the sensors are too expensive. Google Spectrum is a purpose-built spectrum data collection platform that does not cover most of the frequency bands [15]. The researchers propose ElectroSense [37], an open, collaborative spectrum data monitoring platform that uses lower-cost sensors to collect data while covering a larger frequency band, and thus it becomes the infrastructure for this work.

ElectroSense

ElectroSense consists of two parts, the central server and the sensors. The central server is used to store, process, and analyze the RF spectrum data collected by the sensors. The sensors are used to collect RF utilization for each frequency in the environment and transmit it to the server side. The platform uses a crowdsensing approach to collect data collaboratively, and anyone can purchase or assemble their own sensors and connect them

to the platform for data collection. For the purpose of promoting the use of the platform as much as possible, sensors usually consist of inexpensive computing devices, such as Raspberry Pi and antennas. Thus, the open, scalable, and effective ElectroSense attracts a large number of users and provides various services to a large number of stakeholders.

However, large-scale radio spectrum monitoring network data also faces many unique information security issues [38], such as data integrity issues due to the openness of the platform and data privacy issues due to the collaborative model. Next, this work analyzes the cybersecurity issues of RF spectrum networks.

2.1.2 Security Issues on Radio Frequency Spectrum Networks

Due to the RF spectrum platform's open and collaborative characteristics, IoT spectrum data is particularly vulnerable to cyberattacks. Generally, network attacks can be divided into two categories, active and passive attacks [24]. Active attacks are attacks that attempt to disrupt the availability of the network through various means, such as Denial of Service (DoS). While passive attacks do not attempt to disrupt the network but rather listen to and steal information transmitted over the network. Table 2.1 summarizes the characteristics of each type of attack and the main damage caused by them.

Table 2.1: Summarization of different types of attacks

Attack Category	**Attack Type**	**Characteristics**	**Main Damage**
Active attacks	Primary Users Emulation Attack (PUEA)	Imitate the behavior of the normal senses, and transmit fictitious data to the server	Denial of Service
	Spectrum Sensing Data Falsification attack (SSDF)	Transmit false data to the server	Corrupt the spectrum data integrity
	Congestion Attack	Attack control channel	Disrupting network performance
	Jamming Attack	Attack the physical layer of the network	Denial of Service
Passive attacks	-	Monitoring data transmission	Data privacy issues

Active attacks against the RF spectrum can be classified into four different types [23]. The first type is *Primary Users Emulation Attack* (PUEA). This type of attack intends to imitate sensor data and transmit these fictitious data to the server to disrupt network performance or cripple the network infrastructure. The second type is *Spectrum Sensing Data Falsification attack* (SSDF). This type of attack aims to corrupt the correctness and integrity of data by sending false data to the server to demoralize the correctness of network data or confuse the neighboring nodes. The third type of attack is *Congestion Attack*. It intends to mimic or flood the control plane data between sensors and servers

to undermine work performance. The last is *Jamming Attack*, which attempts to attack the physical layer of the network and cause unavailability of the platform.

For passive attacks, the malicious users do not intend to damage the platform infrastructure or reduce the availability of the network but to steal the transmission data, including spectrum data and control data. Usually, this kind of attack can be avoided by using encryption, and such an attack could be caused due to the lack of consideration when designing the platform or simple encryption algorithms.

SSDF Attacks

In this work, the research on cyberattacks detection mainly focuses on SSDF. For this type of attack, device behavioral fingerprinting is a common attack detection method [42].

Table 2.2: Description of each type of SSDF attack

SSDF Attacks Family	**Attack**	**Description**
Hybrid	Repeat	Select the spectrum data for a specific period of time and then continuously forwards it to the server side.
	Mimic	This attack first defines two RF spectrum segments S_a and S_b, and when the attack detects that the sensor is about to send S_b, it replaces it with the spectrum data that maintains S_a and then sends it to the server.
	Disorder	This attack first defines two RF spectrum segments S_a and S_b and two files F_a and F_b, if the data to be sent contains S_a, then save the data to F_a, and then send the data of F_b to the server; similarly, if the data to be sent contains S_b, then keep the data only F_b, and then send F_a to the server.
Transmission Simulation	Noise	Add noise to the data containing a particular segment before sending the data.
	Hop	For each specific time, which is configurable, the attack randomly selects one segment of a given frequency band and adds noise.
	Spoof	Similar to mimic, but with the addition of a noise addition mechanism.
Transmission Hiding	Freeze	Copy the spectrum data for a specific time period and replaces the spectrum data containing a specific pattern with them.
	Delay	Similar to Freeze, but with the difference that Freeze uses the same data for substitution, while Delay uses a sliding window to save the replacement data.

Further research divides SSDF attacks into three categories, including Hybrid Attacks, Transmission Simulation Attacks, and Transmission Hiding Attacks [9]. Among them,

Transmission Simulation SSDF Attacks are used to modify the original data and transmit fake data. Transmission Hiding is used to disguise the transmission to propagate illegal or unauthenticated data, while Hybrid SSDF Attacks are used for both. Table 2.2 shows the name of each specific attack and its attack description. These eight attacks shown in the table will be used for attack identification in the later experiments. Next, a brief introduction of device behavioral fingerprinting is presented in this work as a supplement to the background information.

2.2 Device Behavior Fingerprinting

An exponential number of devices are connected to the network, various cybersecurity risks and threats have exploded. As people's life and work become more and more inseparable from these smart networked devices, e.g. mobile phones, computers, IoT, etc., how to protect these devices from malware attacks in offices, factories, and homes has become more and more important. To solve this challenge, inspired by the human behavioral data science, modelling of device behavioral data has become a promising way [46]. Flourishing researches focus on creating device behavior patterns and fingerprints are hopeful ways to improve the algorithm performance in cyberattack detection [48, 47]. In this section, this work introduce the common life cycle of behavioral fingerprinting applications, and provide a brief analysis of each of the phase.

2.2.1 Life Cycle of Behavioral Fingerprinting Application

To meet the paradox between the growing number of cyberthreats and limited resources, two kinds of methods have been applied for cyberattacks detection: static and dynamic analysis.

A static method is to analyze static features extracted before running an application [11]. Typically, these static techniques rely on source code analysis, permission requests analysis, control flow analysis, and signature-based detection [13]. These techniques are very effective and efficient. However, researchers also point out that this static method could be easily bypassed or circumvented with some confusion methods and packing techniques [11, 13, 54].

The second option is dynamic analysis, which dynamically analyzes various run-time states of the application to determine whether it is an attack. As discussed in [14], the dynamic analysis method shows more robust capabilities and potential than the static analysis way. Therefore, behavioral data analysis becomes a necessity to dynamically analyze the state of devices and applications [30, 42].

Figure 2.1: Life cycle of the device behavioral fingerprinting application

In this context, this work has studied the recent research of device behavior fingerprinting techniques for attack detection, including the behavior sources, processing approaches, and evaluation methods. Figure 2.1 shows a typical lifecycle for the device behavioral fingerprinting applications. As illustrated, a typical lifecycle consists of three aspects. Firstly, the device behavioral data is obtained through the data monitoring program, and the raw data is obtained after data cleaning and pre-processing. Secondly, various technical means are applied to extract the desired features from the raw data, and then these feature data are used to train and evaluate the model. Finally, the trained models are used in practical applications, and feedback is obtained to improve the models continuously.

2.2.2 Behavior Source

Behavioral data refers to the dynamic data generated during the operation of a device or application, and it comes from the generation of its operation or the interaction with the outside. In general, behavioral data sources can be divided into the following categories: network and communication, hardware events, system operation, resource usage, program operation, and sensors [42].

- **Network and communication**. When a device communicates with the outside devices over the network, network-related behavioral data will be generated, including packets, traffic, and other related statistical and analytical data. The advantage of this type of behavioral data is that their acquisition is universal and applicable to a variety of different platforms, as long as they are networked. But the disadvantage is also obvious, that is, the dimensionality of the raw data will be huge [42].

- **Hardware events**. As microprocessor technology improves, hardware behavior data from the underlying layers can also be well monitored and obtained. This kind of data could help improve the accuracy of high-level applications. However, the variety of CPU architectures brings the difficulty of obtaining data.

- **System operation**. On top of hardware, data from operating systems is also a bright source of device behavior. Similar to hardware data, they can also improve the effectiveness of the applications. However, the diversity of operating systems makes it challenging to obtain this data.
- **Resource usage**. This is a common source of device behavior data, such as the status of CPU, memory, disk, etc. It is easy to obtain and simple to process, but continuous monitoring of resource usage can consume a lot of system resources.
- **Program operation**. The software generates a large amount of behavioral data, including status, requests, etc. These behaviors are particularly suitable for program-specific analysis to determine whether a program is a malware or inside attack. Commonly used features include system calls, software status, etc.

Many studies have shown that system calls are particularly suitable for anomalies detection tasks [11, 12, 13, 27, 11, 54], therefore this work concentrates on using system calls to identify cyberattacks in IoT environments.

2.2.3 Approaches for Extracting Features from System Calls

The system call is the way in which a user program requests a service from the operating system (OS) kernel when executing. It contains the time of the request, the instruction to be called, and the associated parameters. Formally, system calls can be viewed as a sequence of function instructions, similar to a text sequence. Consequently, various feature extraction techniques commonly used in Natural Language Processing (NLP) tasks can be utilized in the feature extraction of system calls, such as bag-of-words, sequential encoding, graph-based encoding, and statistical encoding approaches. These four encoding approaches are summarized and compared in Table 2.3.

Table 2.3: System calls features comparison

Syscalls Features	**Computational Complexity**	**Dimension**	**Positional Information**
Bag-of-words	Low	Low	Lost
Sequential	Low	Very High	Remain
Graph based	Very High	Low	Partly Lost
Statistical	Median	Low	Lost

Bag-of-words Encoding

The bag-of-words model is a simple method for data representation in NLP or Information Retrieval tasks [17]. In this approach, the system calls trace is represented as the bag of its terms (either the single system calls word or the n-gram tokens), but discarding the order of the terms [30]. This approach first constructs a system call dictionary and then creates a feature vector based on this dictionary. Each dimension of the vector represents

one type of system call, and therefore, the dimension of the feature vector is consistent with the type of system calls. This approach is cheap in computation, but the positional information is lost or partly lost.

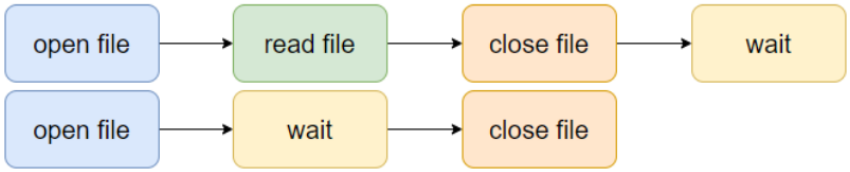

Figure 2.2: Example of system calls

Generally, there are two ways to fill this bag, one is to directly use the frequency of term's occurrence in the sequence. For example, Figure 2.2 illustrates two sequences of system calls. Thus a system dictionary with {0: open file; 1: read file; 2 close file; 3: wait} is constructed. Based on this dictionary and the frequency of the system calls, the 1Gram system calls frequency feature of these above system calls sequences are presented as [1, 1, 1, 1] and [1, 0, 1, 1].

And the other common method is Term Frequency-Inverse Document Frequency (TF-IDF), which is intended to reflect the importance of a term to a document in the whole corpus [6]. Formula 2.1 presents the calculation of the TF-IDF value for a term t in a document d, which belongs to the corpus D, where n stands for the number of terms in d, $f_{i,d}$ stands for the frequency of term i in d, N is the number of documents in the corpus, and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears. Therefor, the 1Gram TF-IDF features of above system calls are [0.45, 0.63, 0.45, 0.45] and [0.58, 0, 0.58, 0.58]. Since the $readfile$ appears only in the first data, it has a more significant value in the first feature vector. Compared with term frequency approach, TF-IDF is able to extract the information on the overall corpus.

$$tfidf(t, d, D) = \frac{f_{t,d}}{\sum_{i=1}^{n} f_{i,d}} * \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2.1}$$

Sequential Encoding

The sequential feature is to use numerical data to directly represent the system calls sequence. The value of this numerical representation could be the index of the term-dictionary, or the one-hot vector [11, 8]. For example, with one-hot encoding, the above system calls could be presented as [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]] and [[1, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0]]. Although the sequential features maintain the most of the order and semantic information, the size of such a feature space could be very large [17]. For example, if the system calls word dictionary is 250, and the trace length is 100000, then the system calls sequence is represented by a feature matrix with a size of 250 * 100000, but in the resource-limited platforms, this high computational consumption is unacceptable.

Graph-based Encoding

If system calls sequence is taken as a directed graph, then some knowledge of graph theory can be applied to the encoding of system calls [52, 18]. A common encoding algorithm is the dependence graph [2], in which a directed graph of system calls is first constructed, where the nodes are specific system calls and the node-to-node edges are the Euclidean distances between them. Then, by traversing the whole sequence, an dependence matrix is constructed [13]. The dependency of two system calls s_i, s_j could be calculated by formula 2.2, where $p(s_i)$ and $p(s_j)$ are the position of these two system calls in a sequence, and k is the total length of the system calls. Thus, the dependency of system calls can be calculated pairwisely.

$$dependency(s_i, s_j) = \begin{cases} 0, & \text{if} \quad i = j \\ \sum_{i<j<k} \frac{1}{p(s_j)-p(s_j)}, & \text{else} \end{cases} \tag{2.2}$$

The first example in Figure 2.2 can be calculated based on the formula that its dependency graph feature is represented as:

$$\begin{bmatrix} & openfile & readfile & closefile & wait \\ openfile & 0 & 1 & 1/2 & 1/3 \\ readfile & 0 & 0 & 1 & 1/2 \\ closefile & 0 & 0 & 0 & 1 \\ wait & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.3}$$

In this approach, the obtained dependence matrix retains part of the distance information of the sequence, which could be calculated with. However, in practice, to compute such a matrix requires a large amount of computation resource, especially when the sequence length is long.

Statistical Encoding

The previous methods are word or term level encoding techniques, but in this kind of model, high level statistical features were used, like the max frequency of these terms, and the standard deviation of the frequencies in the sequences [27]. In general, it is used as a complement to other features rather than using it alone.

2.3 Artificial Intelligence in Anomaly Detection

Once behavioral data is obtained from different types of sources, fingerprints could be created by processing these monitored raw data. Dozens of approaches could be applied

in behavioral fingerprints creation, but this work focuses on artificial intelligence (AI) techniques, that is ML and DL approaches. This section briefly introduces the fingerprint creation techniques and evaluation methods, including the centralized technologies, as well as the distributed and privacy persevering approach.

2.3.1 Machine Learning and Deep Learning

Nowadays, ML and DL algorithms are dominating the field of data processing and data mining and playing an increasingly important role in scientific and industrial practices [5]. ML and DL can be applied to many different tasks, such as classification, regression, clustering, etc. This work focuses on using them in anomaly detection tasks.

Anomaly detection refers to finding the outliers that differ from normal data patterns [10]. In other words, the goal of anomaly detection is to separate normal data from abnormal data. Frequently used ML-based anomaly detection algorithms include One-class Support Vector Machine, Robust Covariance, Isolation Forest [3].

Based on the improvement of computing power and artificial intelligence theory, Deep Neural Networks (DNN) have made significant progress after 2010s [22]. Whether in the areas of natural language processing, computer vision, or recommendation systems, DL has demonstrated its impressive capability. Autoencoder-based models are widely used in various unsupervised learning tasks, and their excellent performance allows DL algorithms to prevail in anomaly detection tasks as well [34].

One-class Support Vector Machine

Traditional support vector machines (SVM) are commonly used for binary or multi-label classification tasks, i.e., finding the appropriate hyperplane to slice the data space accordingly to separate data [49]. This means that the training data corresponds to different labels, and the model needs to find the patterns to map to the appropriate labels. However, One-class SVM assumes that all of the training data only contain one class of label, and the model needs to find the support vectors which could discriminate the normal data from outliers [43, 44]. Any data point that falls out of the discriminative boundary is called an outlier. With the kernelized methodology, One-class SVM is well suited for tasks with high data dimensionality and large data volumes [43]. Meanwhile, the One-class SVM algorithm makes no assumptions about the distribution of the data, which adds to its attractiveness.

Robust Covariance

Robust Covariance is another commonly used anomaly detection algorithm. Different from One-class SVM, Robust Covariance assumes that normal data follows the Gaussian distribution while defining the boundaries of inlier and outlier data as elliptical or ellipsoidal [41]. Based on this, the algorithm estimates the location and covariance of the

normal data from the training data and try to find the optimal elliptic boundary. The algorithm is stable during the border selection and not easily disturbed by outlier data. Besides, Mahalanobis distance instead of Euclidean distance is used to metric the distance between the data points. Overall, Robust Covariance has high robustness and effectiveness. However, its data distribution assumptions and high computational consumption limit its usage scenarios.

Isolation Forest

When dealing with high-dimensional datasets, random forest is always an attractive way. Hence, naturally, the Isolation Forest is also widely applied for anomaly detection. The algorithm recursively selects a feature as the segmentation feature of the data. It randomly selects a value as the basis for partitioning to build a tree-like model. A set of such trees eventually forms a random Isolation Forest, where the path length from the root to the leaf nodes of the tree is used as a basis for distinguishing normal data from abnormal data [25, 26]. Compared with other models, Isolation Forest has better interpretability.

Autoencoder

Autoencoder method aims to acquire the recondite features of the training data by reconstruction the original data using the hidden layers [51]. Figure 2.3 shows a common architecture of the Autoencoder Neural Networks. The left part is called encoder, which is used for mapping the original data to a low-dimensional space. And the right part becomes decoder, which is responsible for reprojecting the low-dimensional data into the original dimensional space. During this process of compression and reconstruction, the hidden information in the data is revealed. In this process, normal data will be reconstructed reasonably, but abnormal data will not. As long as a suitable threshold can be selected, normal data and abnormal data would be separated effectively [34].

As aforementioned, these algorithms are all centralized, which means that they do not take into account the privacy issues may arise during the training and evaluation process.

2.3.2 Federated Learning

These before mentioned ML and DL algorithms have a wide range of applications in anomaly detection, but they are not designed for privacy-preserving scenarios. In the traditional ML/ DL-based IoT anomaly detection pipeline, data are collected in IoT sensors and then uploaded to the central server for model training and evaluation. This process is shown in Figure 2.4 (A), where the data has to be shared to the server in order to train the model, even if the data contains private or sensitive data. In this centralized training process, the risk of data leakage may occur [39].

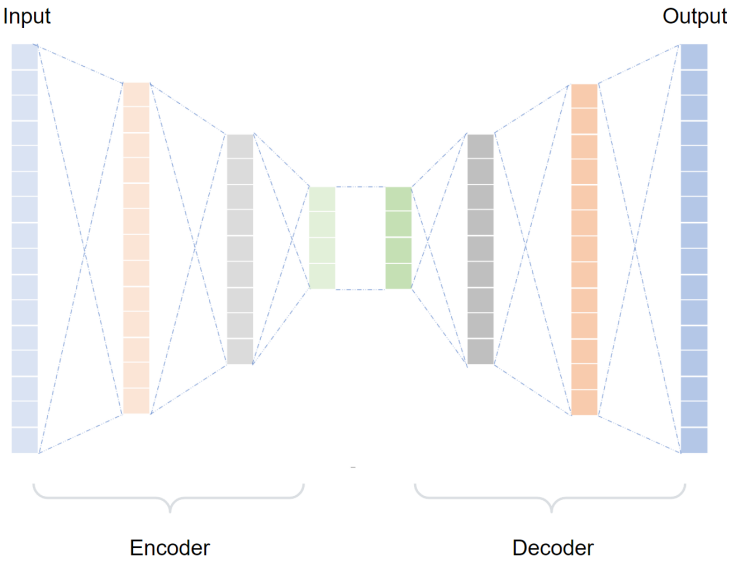

Figure 2.3: Network Architecture of Autoencoder

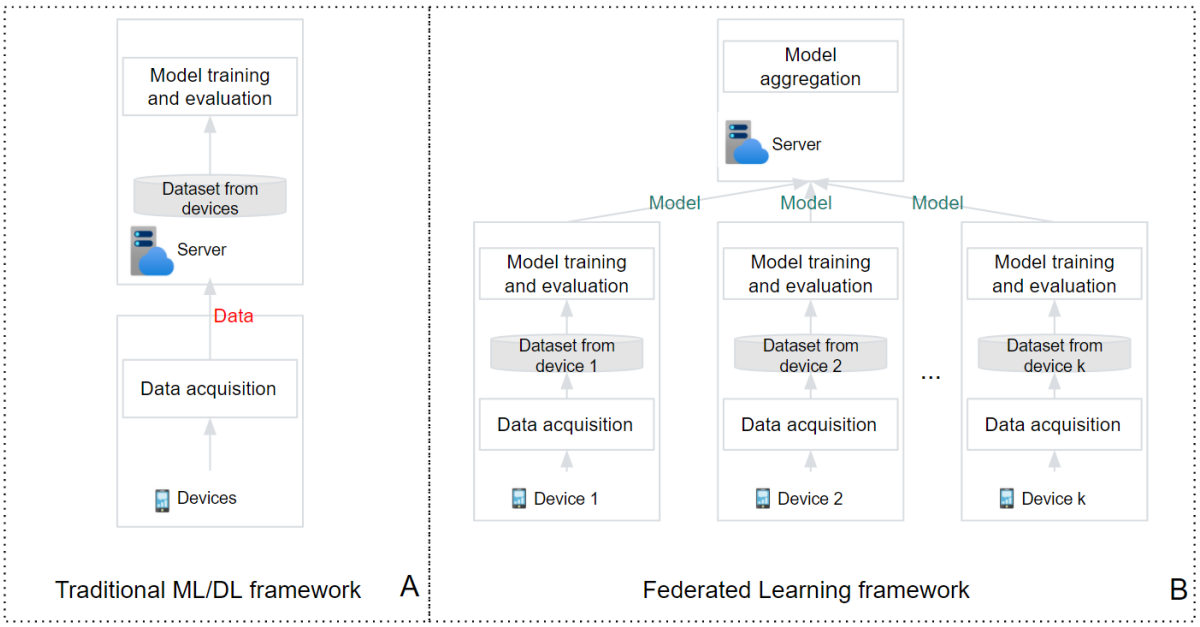

Figure 2.4: Pipelines for traditional ML and DL frameworks (A) and FL framework (B)

Recently, FL [28] has gained the attention of academia and industry because of its decentralized data architecture and privacy protection strategy. As shown in Figure 2.4 (B), in FL, individual nodes, or called clients, generate and collect their own datasets in a decentralized way. Differ from the traditional approach, data will not be uploaded to the server or shared with other peers but only used for the training and evaluation of the nodes' local models. Thereafter, the trained local model parameters of each node are directly uploaded into the server, and a global model is integrated by the server. After multiple iterations

of training, the converged global model is distributed to individual nodes [53]. Privacy protection has been considered in the architectural design of FL. Thus it is particularly suitable for the field of cybersecurity [33].

Chapter 3

Related Work

This chapter analyzes existing works on attack detection using system calls. The first section of this part reviews and analyzes the current state-of-the-art techniques that adopt system calls for cyberattack detection. Finally, this document summarizes and compares the characteristics of the various cyberattack detection approaches through system calls.

3.1 Cyberattack Detection Through System Calls

ML and DL technologies have shown great power in numerous tasks. As a data-driven task, cyberattack behavior detection is highly suitable for ML methods. This section detailed discusses the application of AI techniques to identify cyberattacks.

3.1.1 Traditional Machine Learning Methods

Many tasks still use traditional ML architecture because of its superior performance, simple architecture, and strong interpretability. Similar to natural language, system call traces are also sequential data. Thus several techniques of NLP could be transplanted into system calls-based malicious identification methods. VMGuard [30] is an intrusion detection tool in cloud environment. In this work, system call traces are collected by a syscall tracer, and then Bag-of-Ngram vectors are generated by a pre-processor. Then these features are selected by the TF-IDF method, which is also commonly used in NLP tasks. Finally, selected features flow into a Random Forest classifier to identify the attack traces.

[27] has pointed out that most of the existing works are platform-dependent, thus these works are hardly transplanted into other platforms. Therefore, this work proposes a low-time consumption cross-platform system. This system extracts and computes the system call sequences, Ngram features, and frequency metrics from the input data. Instead of using these features directly, this system calculates the statistical metrics of the original features, e.g., the entropy, standard deviation. Several ML methods are applied in this

work, including Isolation forest, LOF, One-class SVM, and kNN. After comparing different datasets, their method achieves the best average AUC result with 0.737. Experimental results also show that their method presents a high platform-independent ability. It can detect malicious behaviors in both Linux and Windows platforms simultaneously without changing any parts of the architecture.

Hindroid [20] is an Android-based malware detection system. Researchers use a heterogeneous information network to represent the system calls semantically. They use the multi-kernel learning algorithm to integrate different similarities learned from the HIN and determine the final label for each input data. With a comprehensive experimental study on the real-world data, their models show a strong identification ability for ten malware families with an average detection rate of 0.991.

When applying system calls features for malware detection, the real problem is that the dimension of the feature vector extracted from system call traces could be vast. This may cause resource consumption problems in the learning and detection phases. [13] proposes a dependency function, which could significantly reduce the feature dimensions. Experiments show that the dependency graph encoding is better than one-hot and frequency representation. Another dimension reduction method is proposed by [4], which uses the Gaussian dissimilarity method to select the most valuable features from the system call sequences. They use a simple Gaussian Bayes classifier for the malware classification and achieve an accuracy score of 0.98.

The aforementioned methods and researches focus on the traditional environment, i.e., the desktop and mobile environment. However, there are many constraints in IoT devices, like computing capabilities and energy consumption. Consequently, lightweight overhead, wide-coverage, and real-time detection methods are required for IoT defense solutions [7, 31].

A frequently-used method for IoT intrusion detection is the signature-based technique. While the signature-based method is simple enough, it could be easily bypassed with well-decorated malware. Thus [1] proposes a refining signatures selection method. This system uses the system calls to generate dynamic signatures for applications. Then a refinement module is used to select a minimal set of run-time signatures to classify the malware and benign. Their experiments show that this simple system calls-based signature system could identify the malware with 100% accuracy, and only seven signatures are enough to detect seventy types of malware. [7] presents a white-list mechanism for malware detection. They use system calls and hashing functions to generate fingerprints of normal software. During the detection period, they compare the hash code created by the system calls with the white-list and decide whether to terminate the application. They use seven different types of IoT devices for model testing. Results show that their model could provide identity malware with a 100% accuracy score.

However, this offline-generation-online-detection mechanism frequently updates signatures and rules database, which puts additional pressure on IoT devices [16]. Similar to traditional environment malware detection, ML methods are applied in IoT malware classification. [19] implements a multiple one-class SVM classifiers system to identify six families of malware. Experiments show that their model could recognize unknown malware instances with similar behaviors (up to 0.97 detection rate). [16] combines eighteen system

calls, twelve network-based features, and two distributed hash table features to represent the network and system-level information. Their experiments show that the system calls features provide the most supports, followed by network features and DHT features.

[31] combines the white-list with ML method to address the computational cost issues. They propose a three-submodel system, including a process white-list module to generate a process white-list, a process behavior module to collect the run-time process parameters, and a system call behavior module to generate 136 dimension metrics. This device-edge split architecture system could balance efficiently and effectively.

3.1.2 Deep Learning Methods

Besides these traditional ML techniques, recent approaches based on end-to-end neural networks offer a promising alternative. For example, recurrent Neural Networks (RNN), Long-Short Term Memory (LSTM), and Convolutional Neural Networks (CNN) are commonly used for malicious behaviors detection.

Traditional ML methods use many simplifying assumptions, like the data balance, which may not be practical in a real-time environment. [21] introduces a DL architecture model for malware classification. This model uses one-hot vectors to encode the system call sequences and then applies a jointed model with CNN and LSTM networks. This model shows excellent performance for eleven families malware identification, which achieves an average precision of 0.856.

VizMal [11] is a visualization and analysis tool of system calls for Android systems. This application consists of an image builder for execution traces' activity-level visualization and a trace classifier for maliciousness computation. Extracted from system calls, execution traces are represented as one-hot metrics, and then a LSTM neural network has been applied for malware classification. Experiments show a high performance of its malware detection effectiveness with a 0.098 False Positive Rate (FPR). This VizMal tool provides a tool that visually displays application activity and threat levels to help us better understand the behaviors of malware. At the same time, the experiments in this article also verify the potential of DL in the detection of malicious behaviors.

[2] proposes a multi-level architecture for malicious behaviors identification. After being extracted from the system calls traces, a dependency matrix is computed by using a dependency function based on [13]. Then, a four-layers CNN model, which contains two convolutional layers and two dense layers, is used for final inference. This system achieves a high accuracy score of 0.9333.

However, even with these powerful ML and DL detection techniques, malware packed by complex packers could bypass these system [54]. Therefore, they implement a principal component initialized DNN to solve the packing problem. They use the Information Gain to construct a sensitive system calls set and apply their PC-DNN to train their model. With careful experiments of feature engineering and hyper-parameter selection, their model gets a 0.956 detection accuracy score.

Table 3.1: Comparison of different cyberattacks detection approaches through system calls

Solution	Device Type	Features	Attack	Algorithms	Performance
[20], 2017	Android	Bag-of-words (Frequency)	Android Malware	Multi-Kernel Learning	99.01% TPR
[12], 2018	Cloud platform	Bag-of-words (Frequency)	Malicious Intrusions	kNN	90% accuracy, 96% TPR 42.5% TNR
[54], 2018	Desktop	Bag-of-words (Frequency)	Malware	DNN	95.6% TPR
[16], 2020	IoT	Bag-of-words (Frequency)	Malicious Intrusions	NN, SVM, Adaboost	>99.3% accuracy
[19], 2020	IoT	Bag-of-words (Frequency)	Malicious Intrusions	One-class SVM	98% F1-Score
[31], 2019	IoT	Bag-of-words (Frequency)	Malicious Intrusions	Random Forest	100% TPR
[13], 2016	Android	Bag-of-words (Frequency) +Graph-based	Android Malware	Random Forest, SVM, LASSO, Ridge Regression	>93% accuracy
[30], 2018	Cloud platform	Bag-of-words (TF-IDF)	Malicious Intrusions	Random Forest	>94% TPR
[2], 2019	Android	Graph-based	Android Malware	CNN	93.3% accuracy, 0.956 AUC
[52], 2019	IoT	Graph-based	Malicious Intrusions	Autoencoder	98.6% precision
[4], 2018	Android	Sequential	Android Malware	Gaussian Bayes Classifier	98% accuracy
[11], 2020	Android	Sequential	Android Malware	LSTM	90.2% TPR
[32], 2019	Cloud platform	Sequential	Malicious Intrusions	LSTM	>90% accuracy
[21], 2016	Desktop	Sequential	Malware	CNN+RNN	85.6% precision 89.4% on recall
[1], 2017	IoT	Sequential	IoT Malware	White List	100% accuracy
[7], 2019	IoT	Sequential	Malicious Intrusions	White List	100% accuracy
[45], 2020	IoT	Sequential	IoT Malware	RNN	98.7% accuracy
[27], 2020	Desktop	Sequential +Bag-of-words (Frequency) +Statistical	Malicious Intrusions	Isolation forest, LOF, kNN, One-calss SVM	0.737 AUC score

[45] implements a feature generation module with combining the Bag of N-gram and TF-IDF techniques in IoT environment attack detection. Unlike the previous work, [45] uses a RNN model for malware identification. Experiments show that this simple RNN architecture model achieves a 0.987 accuracy score for telnet-based attack detection in the IoT environment.

3.2 Summary and Comparison

Table 3.1 gives an overview and comparison of the reviewed works from the aspects of their detection scenario, features extraction approaches, detected attack, technique, and performance.

- Overall, most of the studies focus on platforms with high computational power, such as desktop and cloud platforms, and there are fewer studies based on IoT environments.
- In terms of feature extraction methods, considering various aspects such as the dimensionality of the features and the ease of extraction, researches using Bag-of-word approaches, especially Frequency approaches, dominate among the various methods.
- Except for [7] which compared the computational cost of the proposed system on different datasets, none of the other works provided a discussion of the computational cost. Meanwhile, no study compared the performance and computational cost of different feature extraction approaches.
- For the selection of algorithms, there are more works based on traditional ML algorithms than DL algorithms, but they all achieve good results. However, there is no work comparing different algorithms' performance and resource usage simultaneously.
- None of the existing solutions consider the privacy of data, i.e., there is no restriction on the transmission of data during the training of the model.

Based on the previous facts, a SSDF cyberattack detection approach which aligned with system calls fingerprinting and privacy-persevering detection framework is urgently needed.

Chapter 4

Scenario and Solution Design

This chapter describes the analysis and design of the system in detail. First, an analysis of the characteristics of the system is done through use cases and scenarios, and then the requirements to be met by the system are summarized. Based on the requirements analysis, a further design of the architecture is completed. The methodology of the system is introduced by a detailed explanation of each module in the architecture.

4.1 Uses Case

Billions of wireless devices are connected to the Internet, and Petabytes of data are transmitted over networks. In order to optimize the spectrum utilization of wireless networks, collaborative RF spectrum data platforms have emerged. As demonstrated, because of the limited computational resources, these sensors are extremely vulnerable to cyberattacks. Therefore, a system that could analyze and detect different types of cyberattacks and adapt to these limited resource devices needs to be developed urgently. In this section, the functions and characteristics of the system are analyzed and elaborated through use cases and scenarios, and the requirements of the system are refined to provide the basis for the system design work.

4.1.1 Infrastructure

Before describing the scenario, the information infrastructure of the system is introduced, including the RF spectrum monitoring platform, sensors, and the SSDF attacks involved in the experiment.

RF Spectrum Monitoring Platform

ElectroSense [37] is the selected crowdsensing RF spectrum monitoring platform for validating the proposed system and evaluating the performance when identifying anomalies

generated by SSDF attacks. On the one hand, sensors connect to the ElectroSense platform through the Internet, monitor the RP spectrum data over the environment, and send the data to the platform. On the other hand, multiple SSFD attacks infect the sensors and affect the integrity of the spectrum data. The proposed system needs to detect these anomalies through device behavioral fingerprint data, specifically system calls, combined with AI algorithms.

Sensors

As aforementioned, this system is embedded in the sensors which are connected to the ElectroSense platform. This work uses a broad range of Raspberry Pi devices equipped with SDR kits as the sensing infrastructure for the proposed system, including Raspberry Pi 3, Raspberry Pi 4 2GB, and Raspberry Pi 4 4GB. The Table 4.1 compares in detail their models, system versions, processing power, and memory size. All the sensors are located in the LAN_1 and then connected to the ElectroSense platform through the Internet to send the RF spectrum data.

Table 4.1: Sensors used in the system

Device Type	**OS Version**	**Kernel**	**CPU Frequency**	**Total Memory**
Raspberry Pi 3	Raspbian 9.13	armv7l Linux 5.4.59-v7+	1.2GHz	925MB
Raspberry Pi 4 2GB	Raspbian 9.13	armv7l Linux 5.4.83-v7l+	1.5GHz	1867MB
Raspberry Pi 4 4GB	Raspbian 9.13	armv7l Linux 5.4.83-v7l+	1.5GHz	3828MB

All the sensors are based on the Raspberry Pi platform, using a customized Linux system based on the Raspbian OS, and the CPU architecture is based on the ARM architecture. The Rasp3 device has the lowest performance, with a CPU frequency of 1.2 GHz and less than 1 GB of RAM. While the Rasp4 2G and Rasp4 4G devices have the same basic technical specification, except that the 4G version runs in a larger memory.

SSDF Attacks Affecting Spectrum Integrity

Chapter 2 has identified eight different SSDF attacks that could affect the integrity of RF spectrum data. Each sensor implements the behavioral monitoring module for monitoring and sending the device's behavior. Meanwhile, the eight SSDF attacks, including Repeat, Mimic, Disorder, Noise, Hop, Spoof, Freeze, and Delay, infect each sensor and generate anomalous system calls behavioral data, which are also monitored and collected for training and evaluation of the proposed approach.

4.1.2 IoT Spectrum Data Integrity Affected by CyberAttacks

This scenario defines two main types of stakeholders for this system, the *crowdsensing client user* and the *server maintainer*. In the client side, sensors collect RF spectrum data and send it to the central server. And the server responses for data storage, processing, and visualization. However, due to the open and collaborative characteristics of RF spectrum platform, malicious users may confuse, interfere, and attack the platform. To identify these malicious users, this system needs to collect data, train models, and then detect attacks. The use cases for these two types of users are defined in the following Table 4.2 and Table 4.3.

Table 4.2: Use cases of crowdsensing client user

ID	**Use Case Description**
UC.1.1	As a client user, I want the system to detect cyberattacks without sharing any personal data with others, including the server and other peers.
UC.1.2	As a client user, even I do not want to share the data directly with others, but I still want to get benefits from the information shared by others.
UC.1.3	As a client user, the fewer system resource used for attack detection, the better.

As shown in Table 4.2, the crowdsensing client user wants its device to be able to effectively detect cyberattacks but also does not want to share its sensitive data with others. The client user cares more about data privacy. However, if a high-level knowledge could help to improve the detection ability of the system, the client user will not refuse such an opportunity. Meanwhile, as the device resource is constrained, the client user does not want to cost too much computational power in cyberattack identification.

Table 4.3: Use cases of server maintainer

ID	**Use Case Description**
UC.2.1	As a server maintainer, I want the system could accurately detect attacks.
UC.2.2	As a server maintainer, I want the system could be applied to as many types of devices as possible. Even for unknown device types, I want the model to still work well.
UC.2.3	As a server maintainer, I want the system could identify as many types of attacks as possible, not only spot these recognized attacks but also could detect these unknown attacks.

The Table 4.3 defines the use case for the server maintainer, who wants to detect attacks accurately. Since various sensors are connected to the system, he wants the model to adapt to as many different types of sensors as possible. At the same time, the detection of attacks should not be limited to known ones but should ideally be for unknowns as well.

4.2 Requirements Analysis

According to the above use cases, this section summarizes and concludes the user requirements from several aspects, such as monitoring of system calls, extraction of data features, and the training process of the model.

Table 4.4 summarizes the requirements of the system. In data collection, system calls should be collected from various devices. Secondly, to protect user privacy, these data must be used only for local model training and evaluation and should not be uploaded to the server or shared with other clients. At the same time, system calls monitoring should consume only a few system resources. Since the data are collected from different devices, they are non independently identically distributed (non-IID).

Table 4.4: System requirements analysis

System Process	ID	Requirement Description
Data Monitoring	R1.1	System calls should be collected from different types of devices.
	R1.2	Data can only be used for local model training and evaluation.
	R1.3	System calls monitoring should not consume many system resources.
Feature Extraction	R2.1	The process of feature extraction should not take up a large amount of computational resources.
	R2.2	The dimensionality of the features should not be too high.
Model Training and Evaluation	R3.1	The architecture of the model should be kept simple and efficient.
	R3.2	The model should be able to be applied to various types of clients.
	R3.3	The model should consume as few system resources as possible during both training and evaluation.
	R3.4	The model should be able to detect not only known attacks, but also unknown ones.

During the feature extraction processing, a small amount of system computational resources should be taken up. Meanwhile, to balance the identification efficiency and effectiveness, the feature dimensionality should be at a low level.

Considering the limited computing resources of the devices connected to the platform, the model should be constructed as simply as possible and be generalizable across devices. At the same time, the training and evaluation of the model should not crowd out the normal operating resources of the device. Besides, the model should be able to identify

unseen attacks and detect even zero-day attacks. Finally, all training data should not be uploaded to the server for training in order to protect the data privacy of users.

These user requirements are identified during the system analysis phase, which helps a better system architecture design and the correct selection of features, algorithms, and training strategies.

4.3 System Architecture

The previous section has defined the requirements of this system. This section describes the architecture that is chosen to implement the above functionality. First, the overall architecture of the proposed system for identifying and detecting cyberattacks affecting IoT sensors is drawn. After that, an introduction of the layers and modules of this detection system is presented with a detailed description of each component.

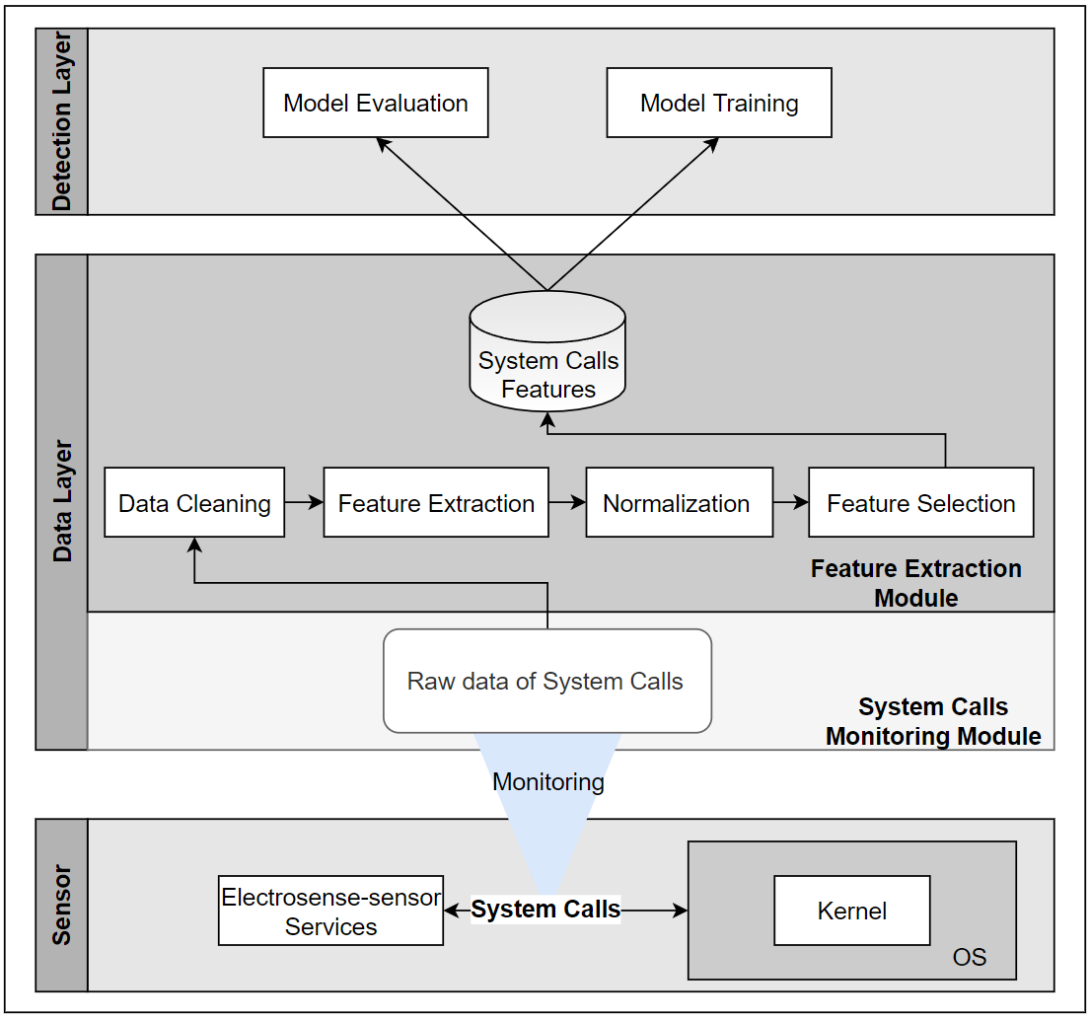

Figure 4.1: System Architecture Overview

Visualized in Figure 4.1, this system has been integrated into the IoT spectrum sensors and consists of two main components, the data layer, and the detection layer. The data layer is responsible for system calls monitoring, data prepossessing, and feature extraction. The detection layer is used for cyberattacks identification.

4.3.1 Data Layer

The data layer is in charge of data collection, cleaning, pre-processing, and feature mining. To ensure data privacy, the data layer only works in the local device, and the obtained data can just be saved in the local machine. This section introduces each module of this layer according to the flow of the data stream.

System Calls Monitoring Module

The Electrosense-sensor Services program runs on the spectrum sensors, and this monitoring module needs to monitor its system calls when it requests services from the OS kernel. In order to meet the requirements defined above, this monitoring application must not consume too many device resources and could run on all listed types of devices.

Thus, the data monitoring module is used to collect system calls from Electrosense-sensor Services and its sub-processes. It can also replace the normal service component with the attacker component to simulate collect system calls of an attacked process. The collected normal and attacked system calls are stored locally for further data cleaning and feature extraction.

```
1  #!/bin/bash
2
3  # copy attack to the electrosense-sensor services and restart the services
4  service electrosense-sensor-mqtt stop
5  cp $attack /usr/bin/es_sensor
6  service electrosense-sensor-mqtt start
7
8  # get the pid
9  pid=$(ps aux | grep es_sensor | grep -v sudo | grep -v grep | awk '{print
       $2}');
10
11 # start perf to monitor the system calls and save results to local
12 timeout -s 1 ${time_window} perf trace -o /data/attack_setup/dataset/${path
       }.txt -e !nanosleep -T -p ${pid};
```

Listing 4.1: Code for system calls monitoring

Listing 4.1 shows the main code blocks of the system calls monitoring module. Firstly, the script replaces the service with the wanted one and restarts the changed service. Then, the target PID is obtained with some filters. Moreover, the program uses the perf tool [35], which is a commonly used Linux system monitoring and maintenance tool, to get the system calls and save them to local files.

Table 4.5: Resource usage for system calls monitoring in each device

Device Type	**CPU Usage (%)**	**Memory Usage (MB)**
Raspberry Pi 3	29.8	6.3
Raspberry Pi 4 2GB	20.2	6.4
Raspberry Pi 4 4GB	15.6	6.5

In terms of resource consumption, these monitoring scripts fully consider the limitations of resource-constrained platforms, and the overall resource consumption rates are all low. Table 4.5 lists the CPU and memory usage by system call monitoring script on each device, and it can be seen that the average CPU consumption is around 20%, while the average memory consumption is around 6MB, which is at a low level.

For each device, this module monitored two separate rounds of system calls. Each round was monitored for a total of 56 hours, with 6 hours of normal data and 6 hours of monitoring for each of the eight types of SSDF attacks.

Feature Extraction Module

- **Data Cleaning**. Typically, the obtained native system calls contain the request time, the requested calls, and the associated parameters. However, only the specific system calls are of interest, and hence other non-relevant data should be removed in the data cleaning phase. After this step, all non-relevant data have been removed, and system call sequences are retained.

- **Feature Extraction**. However, the obtained system calls trace is still textual data and cannot be directly used for model computation. Therefore, the raw data needs to be converted to feature vectors. This work uses three families of feature extraction approaches, which have been introduced in chapter 2, for encoding the system calls behavioral data. The Table 4.6 shows that the bag-of-words extraction approach creates 13 specific features from three techniques, including 1Gram to 5Gram Frequency, 1Gram to 5Gram TF-IDF, and a simple Hashing encoding technique. For the Sequential approach, the One-hot and the Dict-index encoding approaches are used. Furthermore, for the Graph-based method, the Dependency-graph encoding is used.

- **Normalization**. Normalization is necessary for some ML algorithms since they are sensitive to the range of values of the input data. Min-max feature scaling approach is used for data normalization, which is an element-wise operation to map each dimension of the feature value to [0, 1] by the formula 4.1, where x is the original data, and x_min and x_max are the extreme values for each dataset. The normalization scalers are only calculated in the training processing for each dataset. Note that each client has its own data scaler.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \in \mathbb{R}^d \tag{4.1}$$

- **Feature Selection**. Different approaches differ in terms of their computational cost and the detection performance of cyberattacks. Therefore, in the next chapter, different feature extraction approaches (listed in Table 4.6) are evaluated and compared in detail in terms of computational resource utilization, data dimensionality, and anomaly detection performance to select the most effective methodology.

Table 4.6: Feature Extraction Approaches for Encoding System Calls Behavioral Data

Feature Extraction Approach	**Feature Encoding Type**
Bag-of-words	Frequency 1Gram
	Frequency 2Gram
	Frequency 3Gram
	Frequency 4Gram
	Frequency 5Gram
	TF-IDF 1Gram
	TF-IDF 2Gram
	TF-IDF 3Gram
	TF-IDF 4Gram
	TF-IDF 5Gram
	Hashing
Sequential	One-hot
	Dict-index
Graph-based	Dependency-graph

4.3.2 Detection Layer

Based on the aforementioned use cases and requirement analysis, the detection layer is used by the system to train and evaluate the attack detection model. Due to the need for privacy protection, data cannot be uploaded to the server, and thus FL is the most suitable mechanism for model training.

FL based Approach

The flows of data and models during model training and evaluation processes are illustrated in detail in Figure 4.2. After feature extraction, the data is used for local model training (step 1), and then the model parameters are transferred to the server (step 2). The server updates its global model parameters through the Aggregation strategies, and then the global model is distributed to each client (step 3). Through multiple iterations, the model converges and the final global model is used to evaluate the client data and detect possible attacks (step 4). On the client side, five algorithms for anomaly detection are used, experimented with, and compared to meet the purpose of identifying attacks, including One-class SVM, Robust Covariance, Isolation Forest, Stochastic Gradient Descent One-class SVM, and Autoencoder.

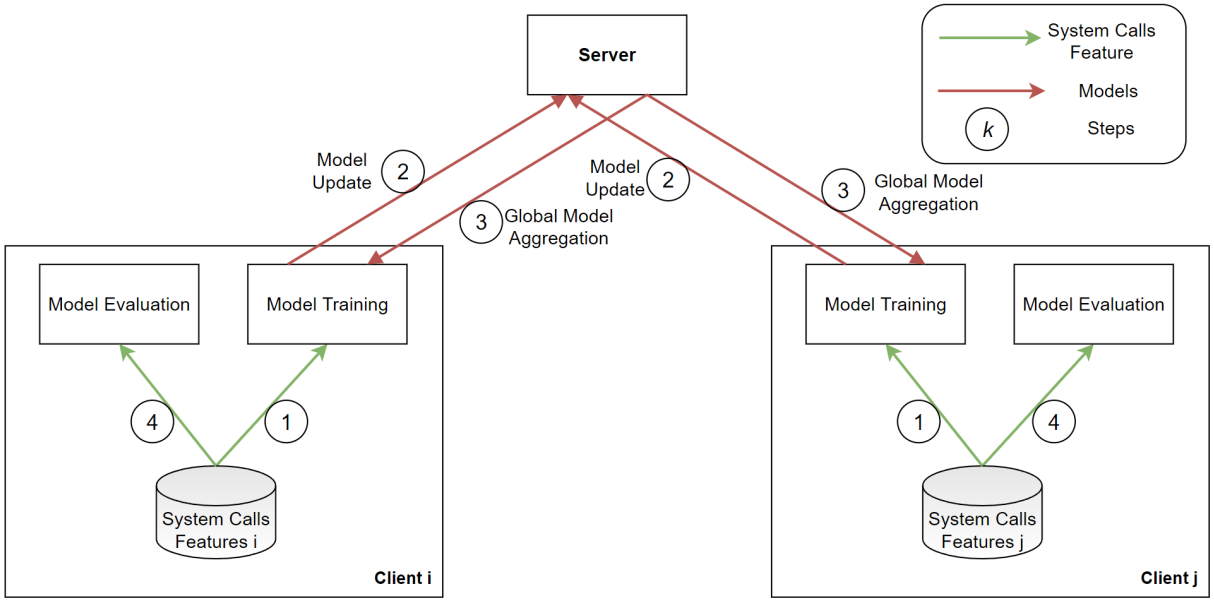

Figure 4.2: Detailed view of the detection layer

Model Initialization

The server side is responsible for the parameter initialization of the global model. First, a global model is created and be initialed with some random weights. Then, this initial model is transmitted to the clients, and the FL process is started.

Parameter Aggregation

Once the training process starts, weights w_k of local models k in $[K]$ clients are updated by the training strategies, and then these parameters are sent to the server side. A set new global model weights w will be aggregated from these local models. A federated average function will be used for this parameter aggregation, which is calculated by formula 4.2, where w is the weights of the updated global model, w_k is the parameter of each local model, and K is the number of clients.

$$w = \sum_{k=1}^{K} \frac{1}{K} w_k \tag{4.2}$$

Evaluation Metrics

It is crucial to correctly evaluate the performance of different solutions. In this work, four main metrics are used to make comparison and evaluation of each approach, including True Positive Rate (TPR), True Negative Rate (TNR), Accuracy, and F1-Score. Their

definitions and descriptions are shown in the Table 4.7, where TP stands for True Positive, TN stands for True Negative, FP means False Positive, and FN means False Negative.

Table 4.7: Evaluation metrics for cyberattacks detection solutions

Evaluation Metrics	**Description**	**Definition**
True Positive Rate (TPR)	The ratio of actual positives correctly predicted.	$\frac{TP+TN}{TP+FP+TN+FN}$
True Negative Rate (TNR)	The ratio of actual negative correctly predicted.	$\frac{TP}{TP+FN}$
Accuracy	The ratio of correctly predicted items to the total items.	$\frac{TN}{FP+TN}$
F1-Score	The harmonic mean of precision and recall score.	$\frac{2TP}{2TP+(FP+FN)}$

Chapter 5

Implementation

The previous chapter develops the architecture of the FL-based cyberattack detection system. In accordance with it, this chapter describes the implementation details of the proposed system. First, it compares the distribution of system calls in each type of device. As the basis of model training, finding the most suitable feature extraction approach is very important. The second part of this chapter compares the advantages and disadvantages of different feature extraction methods in detail and selects the most appropriate one. Finally, as the key to the detection system, the efficiency of the algorithm is crucial. The performance of different ML and DL algorithms is compared in the last section.

5.1 Data Exploration

This section describes the destruction of system calls in each device. Their distributions follow a similar pattern but still have some differences. Therefore, this section first analyzes the types of system calls among different devices and then compares the distribution of different system calls over different attacks. The data show that the distribution of system calls for normal behavior differs significantly from that for attacks, which provides a solid basis for subsequent attack detection.

Table 5.1: Appeared system calls in each type of device

Device	**System Calls**
Raspberry Pi 3	ioctl, timerfd_settime, poll, getpid, write, futex, mkdir, open, close, fstat64, getdents, read, unlink, mprotect, madvcise
Raspberry Pi 4 2G	ioctl, timerfd_settime, poll, getpid, write, futex, mkdir, open, close, fstat64, getdents, read, unlink, mprotect, madvcise, **clock_gettime**, **gettimeofday**
Raspberry Pi 4 4G	ioctl, timerfd_settime, poll, getpid, write, futex, mkdir, open, close, fstat64, getdents, read, unlink, mprotect, madvcise, **clock_gettime**, **gettimeofday**

Table 5.1 illustrates the system calls obtained by monitoring that appears on each device. It is clear that 15 different system calls appear on the Raspberry Pi 3, and they also appear on the other two devices. However, there are two system calls, clock_gettime and gettimeofday, which only appear on the Raspberry Pi 4 family of devices. Both of these system calls are used to get the system time and time zone, but they do not appear in the Rasp3 devices due to differences in the OS kernel.

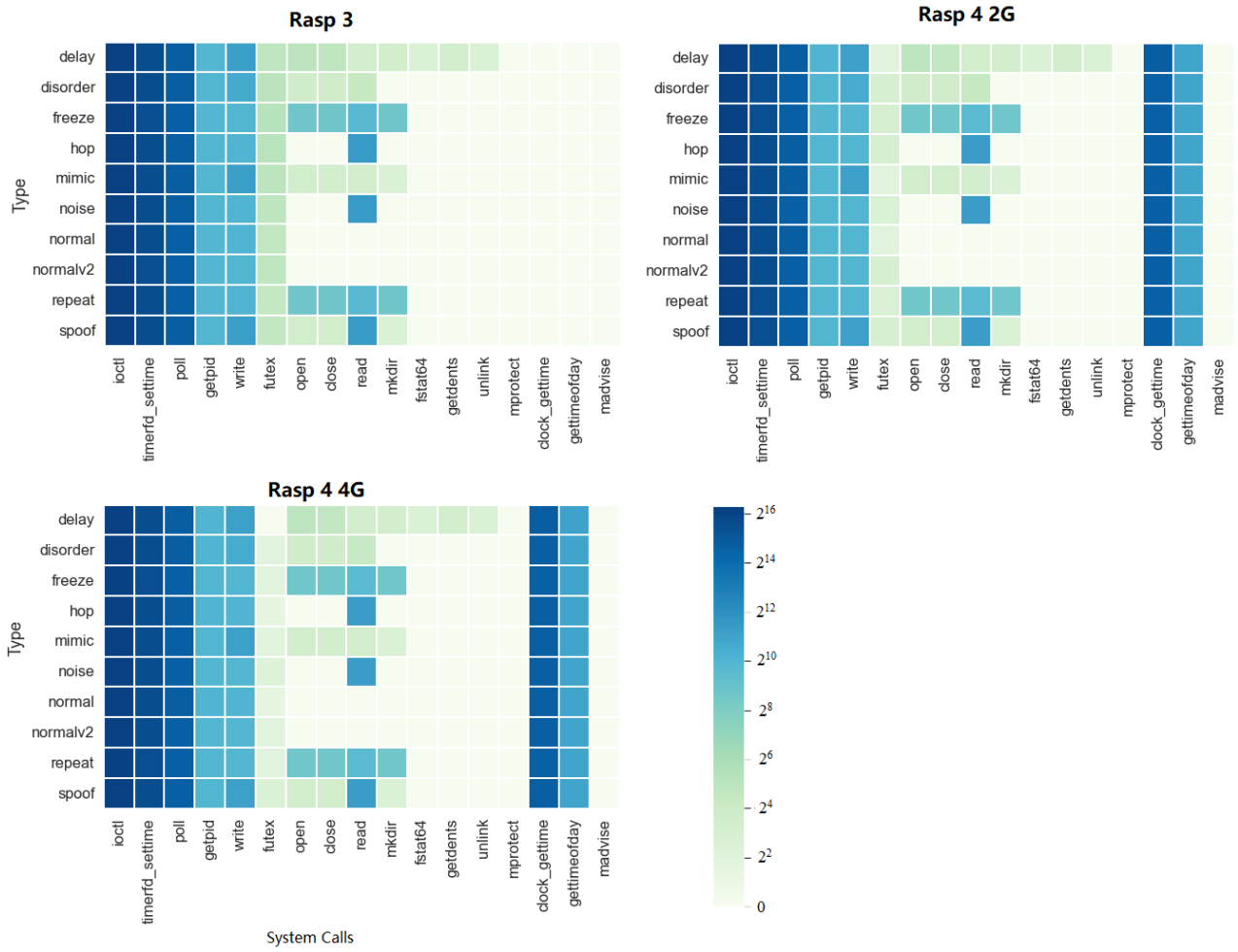

Figure 5.1: Heatmap of average frequency of system calls on different types of datasets

Another question is whether there is a distinction among different system calls under normal and under attack. Figure 5.1 uses a heatmap to show the average frequency of different system calls in different situations. Obviously, except for the two aforementioned system calls, clock_gettime and gettimeofday, the average number of rest system calls are close on different devices. Besides, there is a huge difference between normal data and attack data. For example, system calls such as open, close, read and mkdir only appear in the attack data. This data visualization implies that the attack and normal data are linearly separable.

5.2 Feature Selection

Chapter 4 identifies three feature families, including bag-of-words, sequential, and graph-based features. Since the proposed system operates in IoT devices, the efficiency of extracting these features, as well as the performance of identifying cyberattacks, is a crucial consideration. This section compares various feature extraction methods in several aspects, including the dimensionality of the feature vector, the time of feature transformation, and the system resource consumption. On the other hand, the computational resource consumption of feature extraction approaches also depends on the hardware. Therefore, these approaches' computational cost and detection performance will be considered in combination with the actual cost and cyberattacks detection performance in different types of sensors. Taking into account the efficiency and effectiveness, the final selected features will be used in the FL-based attack recognition system.

The feature extraction methods compared in this section fall into three categories: bag-of-words, sequential, and graph_based. The efficiency of each method in extraction, including the features dimension, computation time, and resources required for transformation, is examined. Table 5.2 to 5.4 show the efficiency side of various feature extraction methods in detail. The compared approaches include 1Gram to 5Gram system calls frequency based, TF-IDF based, and hashing based methods, sequential methods based on one-hot encoding and dictionary index encoding, dependency graph approach. To be noted that the data represented in the table is the time or resources used to transform one piece of data.

Table 5.2: Comparison of system usage of various feature extraction approaches on Raspberry Pi 3

Feature Extraction Approach	**Feature Type**	**Feature Dimension**	**Extraction Time (s)**	**CPU Usage (%)**	**Memory Usage (MB)**
BoW based	Frequency 1Gram	17	0.61	96.07	55.03
	Frequency 2Gram	137	1.26	96.37	58.46
	Frequency 3Gram	596	1.94	97.17	61.76
	Frequency 4Gram	1844	2.67	97.75	70.06
	Frequency 5Gram	4377	3.42	97.97	77.77
	TF-IDF 1Gram	17	0.61	95.38	55.20
	TF-IDF 2Gram	137	1.26	96.69	58.61
	TF-IDF 3Gram	596	1.95	97.30	63.81
	TF-IDF 4Gram	1844	2.66	97.60	70.55
	TF-IDF 5Gram	4377	3.41	98.26	77.06
	Hashing	32	0.70	95.40	77.06
Sequential	One-hot	3,060,000	-	-	-
	Dict-index	180,000	-	-	-
Graph based	Dependency-graph	289	-	-	-

Table 5.3: Comparison of system usage of various feature extraction approaches on Raspberry Pi 4 2G

Feature Extraction Approach	**Feature Type**	**Feature Dimension**	**Extraction Time (s)**	**CPU Usage (%)**	**Memory Usage (MB)**
BoW based	Frequency 1Gram	17	0.26	91.93	54.46
	Frequency 2Gram	137	0.52	95.23	56.35
	Frequency 3Gram	596	0.81	95.36	60.43
	Frequency 4Gram	1844	1.12	95.91	65.87
	Frequency 5Gram	4377	1.44	96.71	72.30
	TF-IDF 1Gram	17	0.26	92.38	51.72
	TF-IDF 2Gram	137	0.52	95.92	56.67
	TF-IDF 3Gram	596	0.82	95.66	58.97
	TF-IDF 4Gram	1844	1.12	95.63	64.36
	TF-IDF 5Gram	4377	1.44	96.96	70.99
	Hashing	32	0.35	93.86	51.98
Sequential	One-hot	3,060,000	-	-	-
	Dict-index	180,000	-	-	-
Graph based	Dependency-graph	289	-	-	-

Table 5.4: Comparison of system usage of various feature extraction approaches on Raspberry Pi 4 4G

Feature Extraction Approach	**Feature Type**	**Feature Dimension**	**Extraction Time (s)**	**CPU Usage (%)**	**Memory Usage (MB)**
BoW based	Frequency 1Gram	17	0.26	92.41	51.52
	Frequency 2Gram	137	0.52	94.71	56.33
	Frequency 3Gram	596	0.78	95.77	60.24
	Frequency 4Gram	1844	1.09	95.82	66.15
	Frequency 5Gram	4377	1.40	96.51	71.95
	TF-IDF 1Gram	17	0.26	93.40	54.19
	TF-IDF 2Gram	137	0.53	95.33	56.61
	TF-IDF 3Gram	596	0.80	96.14	60.74
	TF-IDF 4Gram	1844	1.09	95.99	66.25
	TF-IDF 5Gram	4377	1.41	96.39	71.99
	Hashing	32	0.35	93.40	54.66
Sequential	One-hot	3,060,000	-	-	-
	Dict-index	180,000	-	-	-
Graph based	Dependency-graph	289	-	-	-

As shown in the three tables, the dimensionality of the sequential-based features is so high that they cannot be effectively transformed in resource-constrained devices. The

computational complexity of Graph-based features is very high, and thus they cannot be computed in IoT devices either. On an AMD Ryzen 7 3800XT 8-Core Processor 3.89 GHz CPU with 32 GB server, the time to complete a feature extraction of such a dependency graph feature is 6.24s, which is unacceptable.

On the other hand, Bag-of-word based features can be extracted in a shorter period, as well as a low level resource usage, with CPU usage around 95% and memory usage around 50MB to 70MB. Meanwhile, the feature matrix will suffer from sparsity if the NGram takes a large value, i.e., the performance of the approach does not improve with the increase of dimensionality of the sparse matrix but may even decrease. Thus, dimensionality reduction methods need to be introduced to address the computational difficulty and the sparsity issues. In this work, the Principal Components Analysis (PCA) technique is used in dimension reduction. The reduced low-dimensional vectors are used to train the model without losing the original feature information as much as possible.

Table 5.5 shows the time used for one individual raw data to be processed by PCA dimensionality reduction in different devices. The results suggest that, even in a resource-limited device, such as the Rasp 3, it is still possible to complete the dimensionality reduction process in a short time. Besides, the system usage of the PCA transforming is also relatively low, the average peak CPU usage for processing one piece of data on each device is 80%, and the memory usage is around 50 MB. Therefore, in this work, for the frequency and TF-IDF features of 4Gram and 5Gram, a PCA dimensionality reduction function is used for the follow-up experiments.

Table 5.5: PCA transform time for each feature on each device

Feature Type	**Input Dimensions**	**Output Dimensions**	**Transform Time In Rasp 3 (s)**	**Transform Time In Rasp 4 2G (s)**	**Transform Time In Rasp 4 4G (s)**
Frequency 2Gram	137	100	0.002	0.002	0.002
Frequency 3Gram	596	100	0.003	0.002	0.003
Frequency 4Gram	1844	100	0.005	0.003	0.003
Frequency 5Gram	4377	100	0.007	0.004	0.003
TF-IDF 2Gram	137	100	0.002	0.002	0.002
TF-IDF 3Gram	596	100	0.003	0.002	0.003
TF-IDF 4Gram	1844	100	0.004	0.003	0.003
TF-IDF 5Gram	4377	100	0.007	0.003	0.003

Overall, by comparing various types of feature extraction methods in terms of efficiency, this work adopts Bag-of-word based approaches as the primary methodologies for feature extraction, while the selection of N-Gram will be evaluated in the next section in conjunction with the performance of the specific algorithms.

5.3 Algorithm Selection

Through the above comparison for feature extraction methods, Bag-of-word based features are selected as the basis for ML algorithm selection in this section. Five ML and DL algorithms are compared in terms of performance and resource usage, including Autoencoder, Isolation Forest, One-class SVM, Robust Covariance, and SGD One-class SVM. Models are trained on the respective datasets of each device.

Experiments Setup

In the division of the dataset, normal data are used to train these anomaly detection models, of which 70% are used for the training set, 30% for the validation set, besides the attack data are used for the test set. For the ML-based approach, the contamination rate, i.e., the percentage of abnormal data in the training dataset, is used with a hyperparameter of 0.05.

For Autoencoder, a deep feedforward neural network architecture with 64, 16, 8, 8, 16, 64 neurons in hidden layers is used, while each layer is nonlinearly transformed with the Rule activation function to recognize hidden patterns in the data. Once the training process of the Autoencoder model is finished, a threshold for anomaly recognition needs to be defined. This work uses Mean Squared Error (MSE) to calculate the distance between the decoded data and the original data. The 95% quantile is defined as the threshold value, beyond which it is considered as an anomaly, to ensure the accuracy of the model but also to ensure that the model is not overfitted.

Performance of Algorithms

Table 5.6 shows the TPR and TNR metrics for each algorithm using the NGram syscall frequency features. Collectively, the Autoencoder, One-class SVM, and Robust Covariance algorithms achieve satisfactory performance (more than 80% for both TPR and TNR) on the 1Gram to 2Gram system calls frequency feature for all three devices datasets. In contrast, Isolation Forest and SGD One-class SVM do not perform satisfactorily, and they both fail to detect the attack data well.

Similar to the algorithms using frequency features, Autoencoder, One-class SVM, and Robust Covariance models using TF-IDF features achieve outstanding performance in low-dimensional cases, as shown in Table 5.7. However, due to the large repetition of system calls in the original data, TF-IDF does not provide more information than frequency, and the models using these features do not have significant improvement in accuracy.

Meanwhile, due to the sparsity problem of the feature matrix mentioned before, the performance of the models does not increase with the increase of NGram; instead, the best models appear at 1Gram and 2Gram, and even after the PCA dimensionality reduction process, the overly sparse feature data does not significantly improve the accuracy of the models.

Table 5.6: Performance of each algorithm with using NGram system calls frequency features

		Rasp 3		Rasp 4 2G		Rasp 4 4G	
Feature Type	**Algorithm**	**TPR (%)**	**TNR (%)**	**TPR (%)**	**TNR (%)**	**TPR (%)**	**TNR (%)**
Frequency 1Gram	Autoencoder	89.81	98.85	89.81	99.65	92.59	99.83
	Isolation Forest	97.22	52.08	90.74	25.69	93.52	19.24
	One-class SVM	95.37	98.85	92.59	99.72	90.74	99.79
	Robust Covariance	96.30	54.34	98.15	51.15	96.3	50.31
	SGD One-class SVM	100.00	0.00	98.15	5.8	100	0
Frequency 2Gram	Autoencoder	93.52	98.89	91.67	99.76	93.52	99.72
	Isolation Forest	97.22	13.23	92.59	24.31	94.44	24.97
	One-class SVM	74.07	98.99	75.93	99.86	74.07	99.9
	Robust Covariance	95.37	76.01	98.15	74.93	95.37	78.37
	SGD One-class SVM	100.00	0.00	100	0	100	49.79
Frequency 2Gram PCA	Autoencoder	90.74	10.90	90.74	22.5	93.52	11.11
	Isolation Forest	95.37	17.71	97.22	11.81	96.3	12.43
	One-class SVM	97.22	72.64	85.19	97.19	91.67	74.62
	Robust Covariance	93.52	79.48	98.15	75.38	93.52	80
	SGD One-class SVM	100.00	41.53	100	49.2	100	51.94
Frequency 3Gram	Autoencoder	90.74	98.96	87.96	99.69	94.44	99.65
	Isolation Forest	93.52	16.22	94.44	17.22	96.3	21.01
	One-class SVM	66.67	98.99	79.63	99.76	72.22	99.83
	Robust Covariance	85.19	81.28	92.59	77.15	95.37	78.26
	SGD One-class SVM	100.00	0.00	100	0	100	0
Frequency 3Gram PCA	Autoencoder	87.96	22.71	89.81	21.63	94.44	8.65
	Isolation Forest	99.07	15.28	93.52	19.9	95.37	15.56
	One-class SVM	93.52	73.13	84.26	97.36	91.67	77.12
	Robust Covariance	92.59	98.89	98.15	75.38	95.37	83.82
	SGD One-class SVM	100.00	51.81	100	39.31	100	40.69
Frequency 4Gram PCA	Autoencoder	85.19	7.57	96.3	7.01	92.59	18.58
	Isolation Forest	94.44	16.28	96.3	17.78	95.37	20.17
	One-class SVM	91.67	73.99	87.96	97.53	90.74	77.71
	Robust Covariance	92.59	98.89	95.37	79.06	97.22	78.47
	SGD One-class SVM	100.00	39.06	100	48.47	100	40.8
Frequency 5Gram PCA	Autoencoder	92.59	3.72	95.37	6.49	92.59	6.88
	Isolation Forest	99.07	9.17	95.37	22.64	96.3	18.65
	One-class SVM	92.59	74.86	89.81	97.67	92.59	78.3
	Robust Covariance	91.67	98.85	94.44	83.06	94.44	77.78
	SGD One-class SVM	100.00	51.63	100	46.7	100	40.8

Table 5.7: Performance of each algorithm with using NGram system calls TF-IDF features

		Rasp 3		**Rasp 4 2G**		**Rasp 4 4G**	
Feature Type	**Algorithm**	**TPR (%)**	**TNR (%)**	**TPR (%)**	**TNR (%)**	**TPR (%)**	**TNR (%)**
TF-IDF 1Gram	Autoencoder	98.15	62.01	96.30	88.85	94.44	84.20
	Isolation Forest	94.44	39.03	97.22	79.20	93.52	81.15
	One-class SVM	85.19	76.84	91.67	99.48	86.11	99.31
	Robust Covariance	96.30	98.40	97.22	99.76	96.30	99.58
	SGD One-class SVM	97.22	0.49	100.00	0.00	99.07	36.25
TF-IDF 2Gram	Autoencoder	95.37	55.66	94.44	99.55	92.59	99.55
	Isolation Forest	91.67	28.68	99.07	51.56	91.67	66.18
	One-class SVM	75.00	84.97	79.63	99.86	72.22	99.86
	Robust Covariance	96.30	98.51	97.22	99.65	96.30	99.51
	SGD One-class SVM	100.00	0.00	100.00	0.00	100.00	49.79
TF-IDF 2Gram PCA	Autoencoder	91.67	93.19	92.59	99.69	95.37	99.48
	Isolation Forest	94.44	32.33	96.30	19.03	97.22	23.85
	One-class SVM	92.59	98.26	96.30	99.65	95.37	99.55
	Robust Covariance	98.15	98.37	97.22	99.72	96.30	99.58
	SGD One-class SVM	100.00	0.00	100.00	0.00	100.00	0.00
TF-IDF 3Gram	Autoencoder	93.52	76.32	90.74	99.58	93.52	99.62
	Isolation Forest	91.67	17.67	97.22	32.88	94.44	38.51
	One-class SVM	65.74	85.69	77.78	99.79	71.30	99.79
	Robust Covariance	88.89	98.92	90.74	99.62	96.30	99.58
	SGD One-class SVM	100.00	0.00	100.00	0.00	100.00	0.00
TF-IDF 3Gram PCA	Autoencoder	94.44	59.76	96.30	99.13	98.15	98.61
	Isolation Forest	95.37	22.50	94.44	19.34	97.22	16.70
	One-class SVM	93.52	97.81	96.30	99.62	96.30	99.55
	Robust Covariance	94.44	89.72	94.44	99.69	92.59	99.62
	SGD One-class SVM	100.00	0.00	100.00	0.00	100.00	0.00
TF-IDF 4Gram PCA	Autoencoder	90.74	53.78	98.15	99.65	97.22	99.51
	Isolation Forest	97.22	14.10	93.52	32.47	97.22	18.92
	One-class SVM	92.59	97.92	96.30	99.62	96.30	99.55
	Robust Covariance	95.37	98.89	97.22	99.72	95.37	99.62
	SGD One-class SVM	100.00	0.00	100.00	0.00	100.00	0.00
TF-IDF 5Gram PCA	Autoencoder	92.59	49.44	97.22	99.58	98.15	99.51
	Isolation Forest	100.00	13.92	93.52	23.26	93.52	20.80
	One-class SVM	93.52	96.77	98.15	99.62	95.37	99.62
	Robust Covariance	95.37	98.89	93.52	99.62	90.74	99.62
	SGD One-class SVM	100.00	0.00	100.00	0.00	100.00	0.00

Computational Costs of Algorithms

At the same time, the efficiency of model training and testing is also an essential factor to be considered. This work compares the computational resources cost of each algorithm trained on different devices (Raspberry Pi 3 and Raspberry Pi 4s) using different features, including training time, CPU usage, memory usage, and the model's size.

Figure 5.2 illustrates the computational cost for each algorithm using different feature extraction approaches on different devices. Overall, the results follow a similar pattern on different devices, and the Robust Covariance has the highest resource consumption among the five algorithms and three devices. It is also the most dimensional-sensitive algorithm, as shown in Figure 5.2, it takes close to 200 seconds for the model to converge when the dimension of the input feature vector is 596 (3Gram) in Raspberry Pi 3, and takes almost 100 seconds in Raspberry Pi 4s. Besides, it has a CPU utilization of over 200% in all three devices. In contrast, One-class SVM is highly efficient, not only in terms of training time and resource consumption in the low-dimensional condition, but also in the high-dimensional condition.

The rest of the three ML algorithms, including One-class SVM, Isolation Forest, and SGD One-class SVM, all keep a low level of training time, CPU usage, memory consumption, and model size. For example, their model training times are less than one second, and their CPU and memory utilization are less than 100%.

For Autoencoder, the Early Stopping strategy is adopted to train the model, and the model will stop automatically when its loss has not changed for ten epochs so that the training time of the model has certain randomness. Even in the Raspberry Pi 3 device, the model can be trained within 10 s. The model's resource usage is relatively low compared to Robust Covariance, and the size of the model after training is at a medium level.

On the other hand, this work presents a detailed comparison of their computational costs for each feature after PCA dimensionality reduction. Figure 5.3 shows the computational cost of different algorithms on different devices using each feature extraction approach after PCA dimensionality reduction. Since the dimensionality of each feature is the same after PCA, which is 100, the computational resource consumption is similar for the same algorithm for different features, and the data follow a similar pattern on each device. However, due to the Early Stopping strategy of Autoencoder, the resource consumption during its training is less stable. As the PCA approach reduces the dimensions of the original data to a lower level, thus for each algorithm, their training resource efficiency decreases a lot compared to the original features.

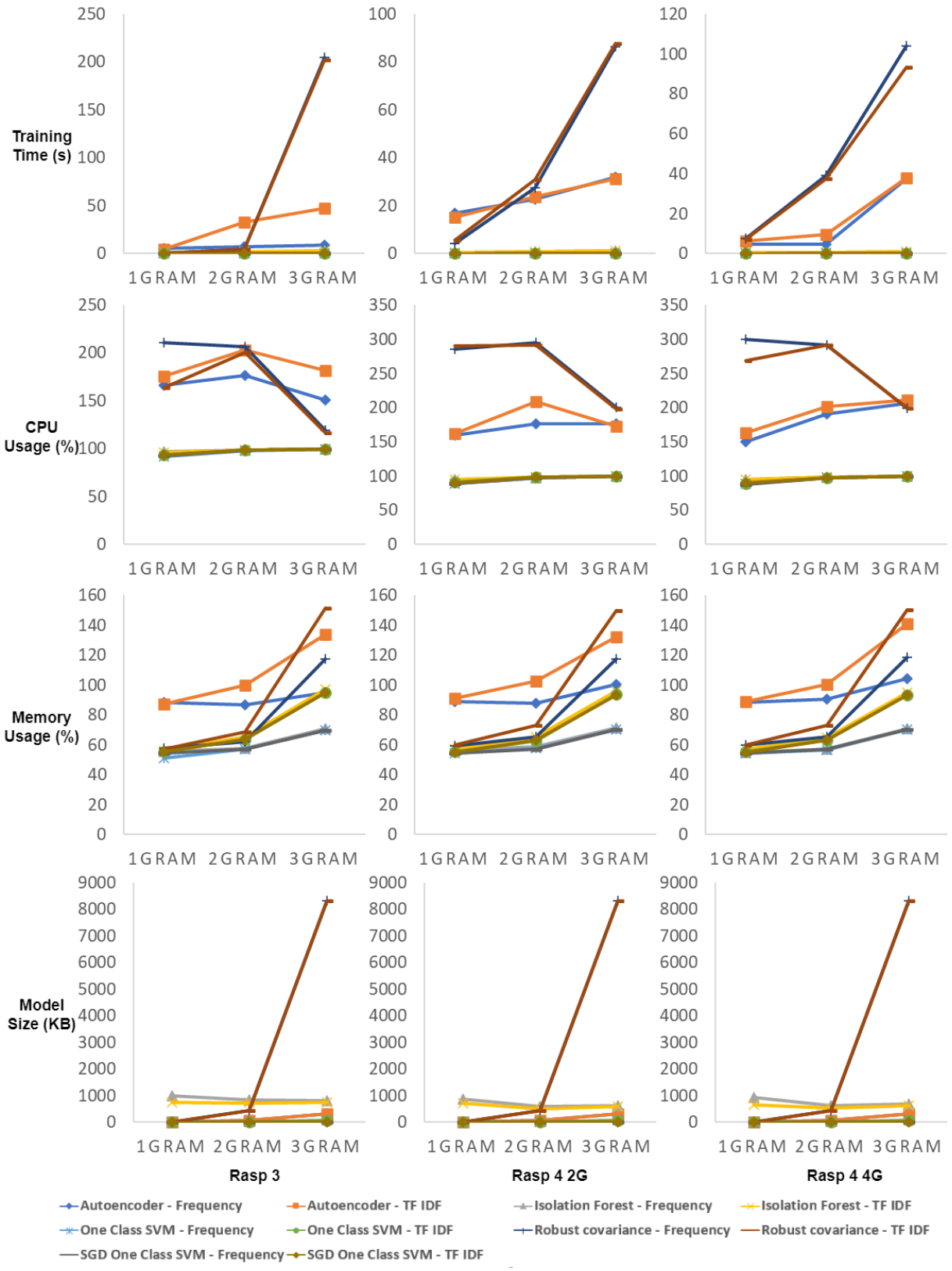

Figure 5.2: Comparison of the cost of each algorithm using different features without PCA

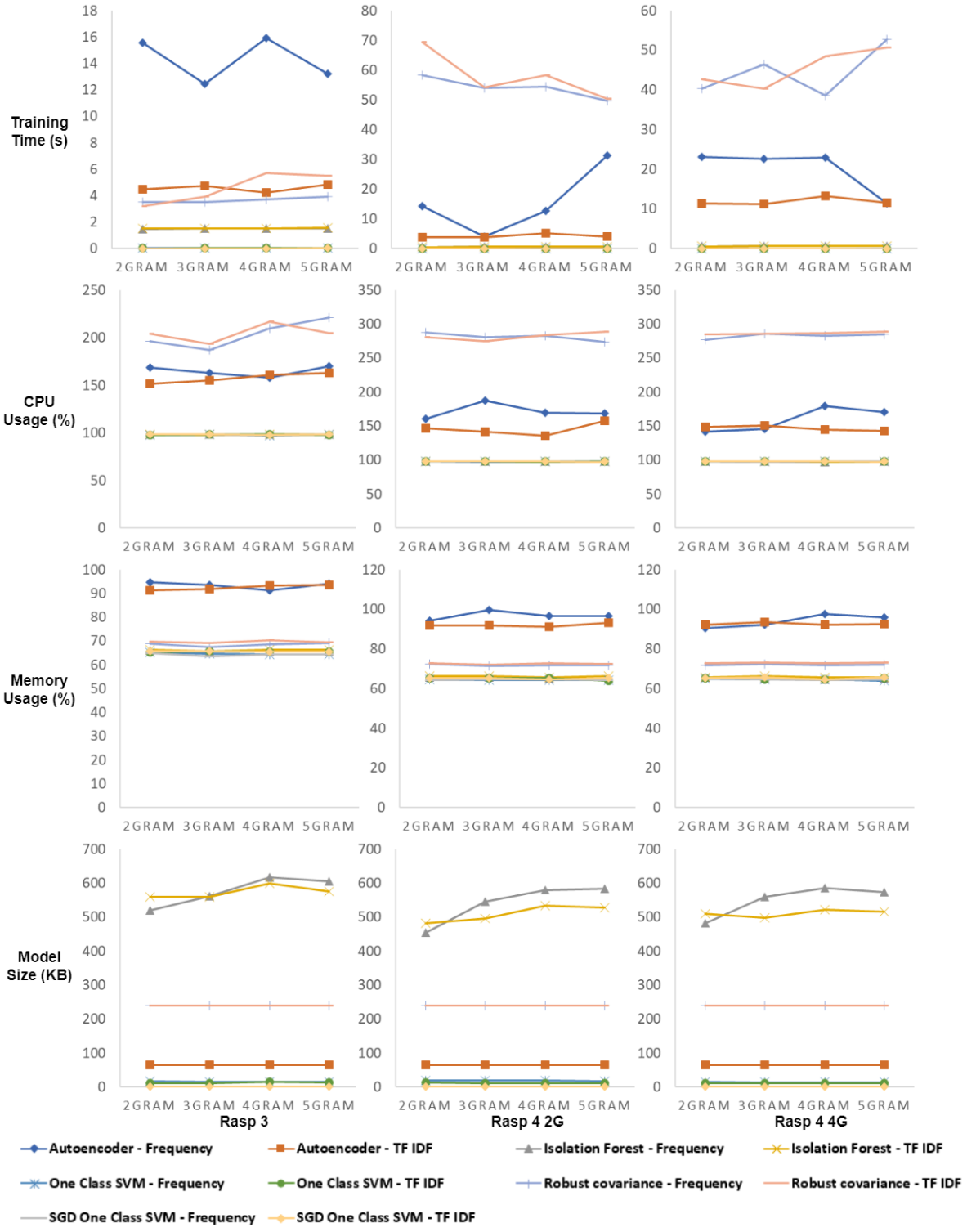

Figure 5.3: Comparison of the cost of each algorithm using different features after PCA

5.3.1 Discussion

On the one hand, all five algorithms can be successfully trained in different IoT sensors. However, in terms of model detection performance, One-class SVM, Robust Covariance, and Autoencoder can successfully identify normal and abnormal data. However, Isolation Forest and SGD One-class SVM algorithms are poor in identifying anomalies, so they are not considered as local models for subsequent FL-based methods.

On the other hand, Robust Covariance has the highest model training time and resource consumption rate among the five algorithms, followed by Autoencoder, but the other three algorithms are cheaper in computational cost. One-class SVM balances performance and efficiency in identifying potential threats. Overall, ML-based One-class SVM and DL-based Autoencoder balance cost and performance and are more compatible for SSDF attack identification in IoT environments.

It can also be seen that the computational resources, including training time, CPU usage, memory usage, and model size, for different algorithms on both frequency and TF-IDF features follow a similar pattern, regardless of the different devices. In other words, for both Frequency and TF-IDF, the same algorithm on the same device, under the same NGram conditions, have similar computational costs. Meanwhile, in NGram Frequency features, a larger N does not bring performance improvement but rather increases the resources consumption. Moreover, even with the PCA dimensionality reduction approach, model performance degradation due to the sparsity of high-dimensional feature matrices is still not well addressed. Therefore, the low-dimensional dense feature matrix has a significant advantage in terms of computational cost and detection performance.

In addition, due to the low variability of system calls, TF-IDF does not provide redundant information at the document level. On the contrary, Frequency provides more interpretability of the model. Therefore, the 1Gram system calls frequency feature is more applicable than the TF-IDF feature in the dataset covered in this work.

To conclude, this work adopts 1Gram Frequency features as the feature extraction scheme on FL-based approaches, and One-class SVM and Autoencoder are used as their local algorithms.

Chapter 6

Evaluation Scenario and Experiments

This chapter details the evaluation process that has been done to validate the effectiveness of the cyberattack detection system based on FL framework. First, experiments are designed with different validation scenarios for exhaustively comparing the performance of traditional and federated approaches on malicious behavior identification. Meanwhile, different use cases are used to verify that the FL-based approach is both effective in identifying anomalous data as well as preserving client privacy.

To validate the FL-based approach, ElectroSense is selected as the crowdsensing RP spectrum platform. System calls have been monitored for both benign and malicious behaviors, where the anomalies were produced by the SSDF attacks described in Chapter 4. As presented in the previous chapter, this work has used a set of Raspberry Pis, including Raspberry Pi 3 and Raspberry Pi 4 2/4 GB, with SDR kits.

In addition, the 1Gram Frequency method is used as the final feature extraction approach, as discussed in Chapter 5, and One-class SVM and Autoencoder are used as the underlying algorithms for the FL-based cyberattack detection methodology. In the following sections, this work uses several case studies to analyze the performance of different cyberattack detection methods in different situations.

6.1 Case Study #1: Privacy-preserving Clients with Local Models

In this case, an assumption is made that all users are privacy-preserving, i.e., they do not want to upload their data to the server or share it with other users, while there is no global model training mechanism based on the FL approach. Consequently, even if the trained models can be shared among users, they can only recognize attacks using local models based on native datasets. For example, customer 1 uses Raspberry Pi 3 device as his sensor and trains a cyberattack detection model based on Raspberry Pi 3 dataset. However, users with other devices, assuming they have no knowledge of ML but are in urgent need of malicious behavior detection models. In this case, the client1 is willing to

share his trained local model, which does not include the dataset, to other clients as their detection system.

This section aims to use this scenario to verify the necessity of FL framework in the context of emphasizing privacy. In the experimental setup, the system calls monitoring module acquires two rounds of data on different devices, and each round includes both benign data and malicious data. The training data uses 70% of the normal data from the first round of data acquired, 30% as validation data, and the benign and malicious data from the other devices as different test sets. The specific data set division and the amount of data in each data set are shown in Table 6.1.

Table 6.1: Number of data in each dataset for User case #1

	Training Dataset	**Validation Dataset**	**Test Dataset**				
	Raps3 Dataset1	**Raps3 Dataset1**	**Raps3 Dataset2**	**Rasp4 2G Dataset1**	**Rasp4 4G Dataset1**	**Rasp4 2G Dataset2**	**Rasp4 4G Dataset2**
Normal	252	108	360	360	360	360	360
Attack	-	2880	2880	2880	2880	2880	2880
	Rasp4 2G Dataset1	**Rasp4 2G Dataset1**	**Raps3 Dataset1**	**Raps3 Dataset2**	**Rasp4 2G Dataset2**	**Rasp4 4G Dataset1**	**Rasp4 4G Dataset2**
Normal	252	108	360	360	360	360	360
Attack	-	2880	2880	2880	2880	2880	2880
	Rasp4 4G Dataset1	**Rasp4 4G Dataset1**	**Raps3 Dataset1**	**Raps3 Dataset2**	**Rasp4 2G Dataset1**	**Rasp4 2G Dataset2**	**Rasp4 4G Dataset2**
Normal	252	108	360	360	360	360	360
Attack	-	2880	2880	2880	2880	2880	2880

The feature used for training is the 1Gram Frequency mentioned above, and the algorithms are One-Class SVM and Autoencoder. Figure 6.1 shows the performance of the local model trained on three devices on different datasets. Models, either the ML approach or the DL approach, could not recognize the normal behaviors from other devices well. For example, when trained with Raspberry Pi 3 data, the One-class SVM model only correctly recognizes 26% of the normal data in the Raspberry Pi 4 2G dataset, while the TPR of the Raspberry Pi 4 4G dataset is even lower at about 20%; if the Raspberry Pi 4 2G dataset is used as the training set, and the Raspberry Pi 3 dataset is used for testset, for both One-class SVM or Autoencoder, their TPRs are below 50%. These two anomaly detection algorithms are sensitive to outlier data, but if the distribution of the normal data changes, these algorithms tend to classify it as abnormal.

Section 5.1 has shown that the distribution of system calls for Raspberry Pi 3 differs significantly from that of the two Raspberry Pi 4s; thus, whether the models trained by Raspberry Pi 3 data predict data from Raspberry Pi 4 devices or identify data from Raspberry Pi 3 with models trained by Raspberry Pi 4, their accuracy in identifying normal data is low. However, the distribution of system calls in two Raspberry Pi 4s are similar; therefore, their local models could successfully predict the data from each other. Since normal data on different devices do not follow the IID thus, all the local models have high TNRs, almost all reaching 100%, but fail to identify the benign behaviors from other devices.

The results of this scenario imply that the local models may work poorly when dealing with data from an unknown distribution. As clients are sensitive to the privacy of system calls data, there is no chance that clients share their data with others; therefore, a mechanism that can solve the non-IID problem is needed.

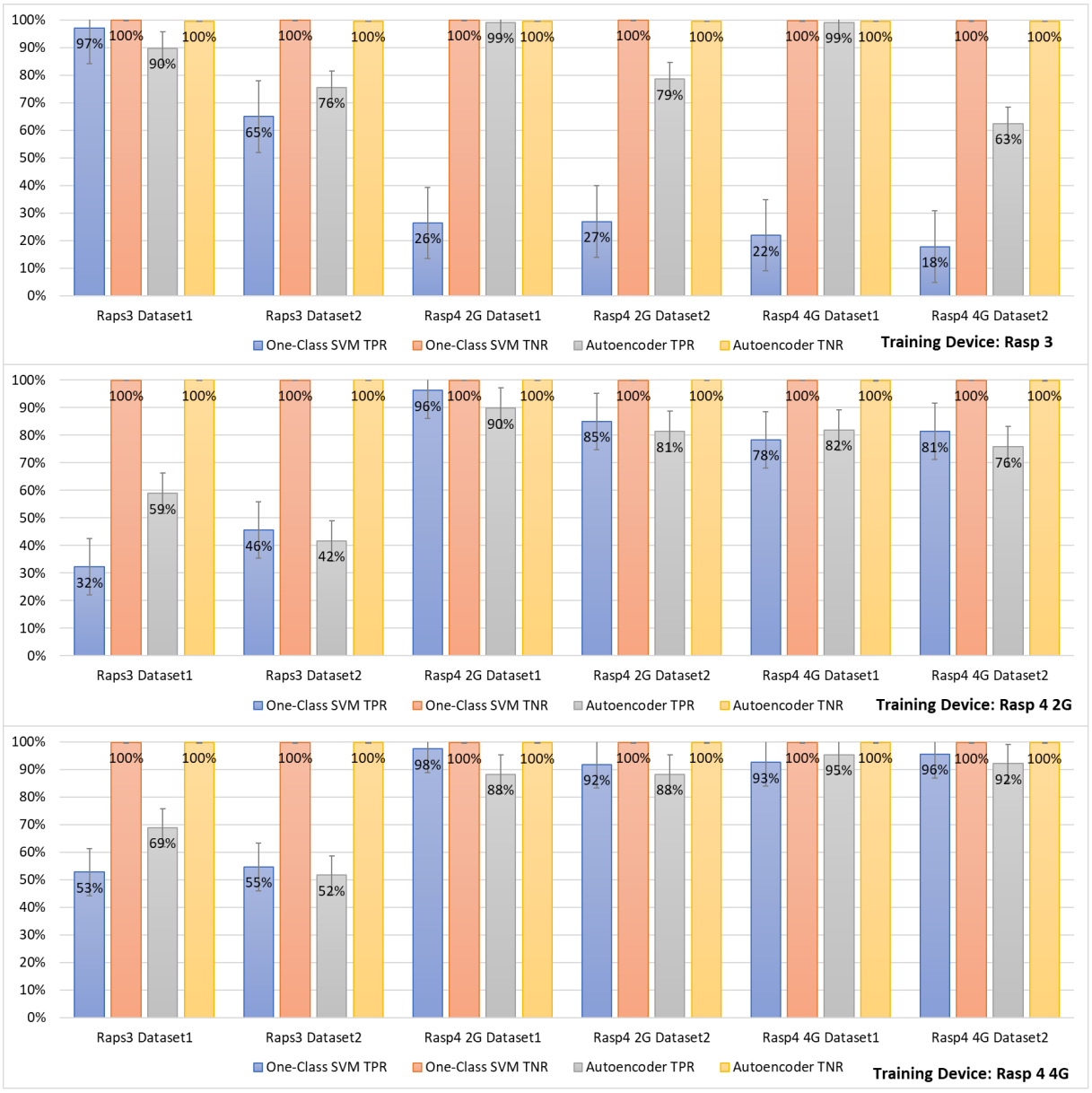

Figure 6.1: Performance of natives models in different training devices

6.2 Case Study #2: Federated Learning Approaches

From the experiments above, the problem of data in different datasets that do not follow the IID distribution needs to be solved. The FL approach is an effective tool to solve this problem. On the one hand, the client data does not need to be shared with peers or with the server, which significantly protects privacy, and on the other hand, using model aggregation, the shared model does not need the data to satisfy the assumption of following the same distribution which is a proper solution to the non-IID problem.

To verify whether the FL-based approach can handle unknown devices and unknown datasets situations, as well as to evaluate the impact of different numbers of training clients, three different FL training settings were experimented as shown in Figure 6.2, including leave-two-devices (Figure 6.2 A), leave-one-device (Figure 6.2 B), and leave-one-dataset (Figure 6.2 C).

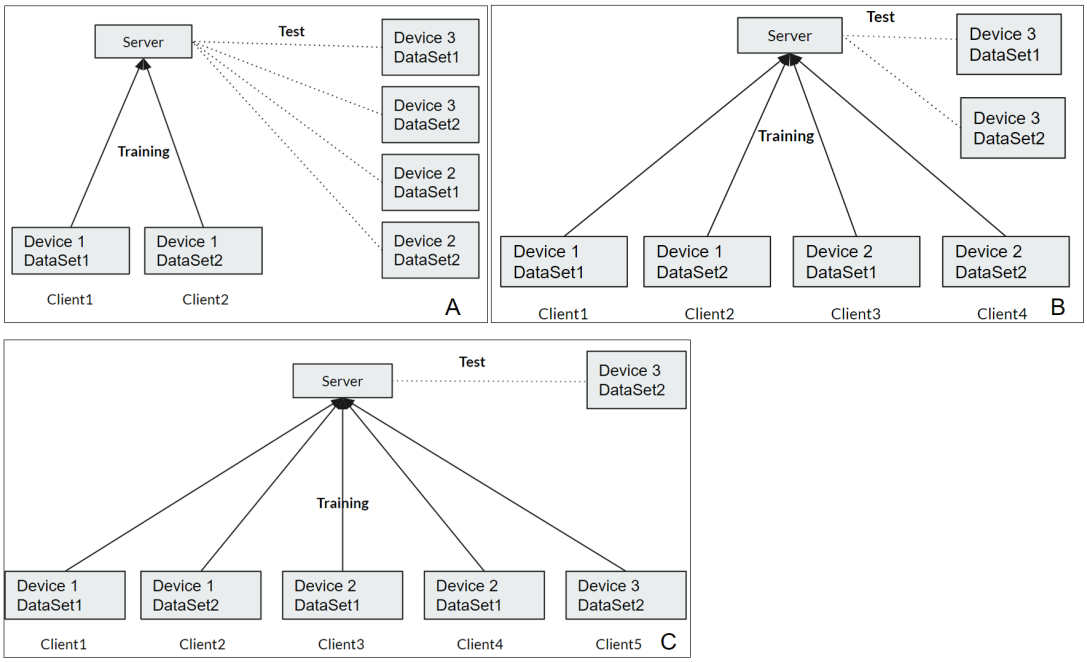

Figure 6.2: Experiments setup for FL-based approaches

In the first case, i.e., leave-two-devices, only one type of device data is used to train the FL-based model. In this setup, the percentage of known devices is 33.33%, while the other two device datasets are used to verify the generalization ability of the model. In the second case, i.e., leave-one-device, data from two different devices are used as the training set, while the percentage of known devices is 66.67%. In the third case, leave-one-dataset, data from all models are involved in the training of the FL model, thus the percentage of known devices is 100%, and only one dataset from one device is not involved in the training and is used to validate the model.

Figure 6.3 illustrates the effect of the different number of known devices on the performance of the models. Overall, the Autoencoder model can detect the attack data in

unknown devices well, even at a lower level of known devices, and it performs consistently. Particularly, the average TPR of the One-class SVM algorithm increased significantly from 81% to 85% as the number of clients involved in the computation grew. On the other hand, the performance of the Autoencoder algorithm is relatively stable, with its TPR fluctuating between 94% and 95%. At the same time, both algorithms show great ability in detecting anomalies, and their TNRs are above 99%, which can detect potential threats remarkably effectively. For One-class SVM, its recognition rate for unknown normal data is lower than Autoencoder but still better than the local models, which are shown in the previous section, and its performance keeps improving as the known devices are raised.

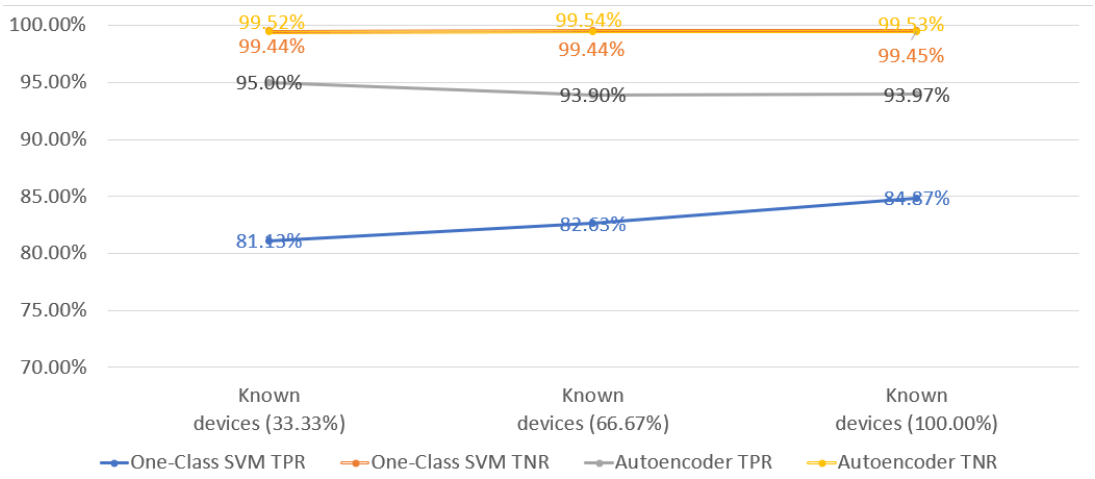

Figure 6.3: Overall performance of models with different percentage of known devices for FL-based approaches

Specifically, Figure 6.4 presents the performance of the FL models based on the set-up of leave-2-devices (i.e., 33.33% of known devices) on each dataset. From the figure, it can be seen that the cyberattack detection model using Autoencoder algorithm can fulfill the task with more than 90% TPR and nearly 100% TNR no matter which device data is used as the training dataset. As for One-class SVM, its recognition rate for normal data is lower but still higher than the baseline of 60%. On the other hand, similar to the local model, if the training dataset for known devices is small, the model still suffers from the problem that it does poorly on unknown devices than on known devices.

Compared with the scenario using only the local model, the leave-two-devices approach demonstrates the superiority of the FL method. When just using Raspberry Pi 3 data as the training set, the average TPR of the local One-class SVM algorithm is only 27% on the Raspberry Pi 4 2G dataset and even 20% on the Raspberry Pi 4 4G dataset. For comparison, the One-class SVM algorithm using the FL-based strategy improves the average TPR to 73% on the Raspberry Pi 4 2G and 79% on the Raspberry Pi 4 4G dataset. Autoencoder also performs similarly, improving the average TPR from 87% to 95% on the Raspberry Pi 4 2G and from 81% to 95% on the Raspberry Pi 4 4G datasets.

In addition, the performance of the approach with just the local dataset varies dramatically on different test datasets, and its standard deviation of TPR reaches 0.29. In contrast,

the standard deviation of TPR for the leave-two-devices approach is 0.1, indicating that it performs more consistently on different test sets.

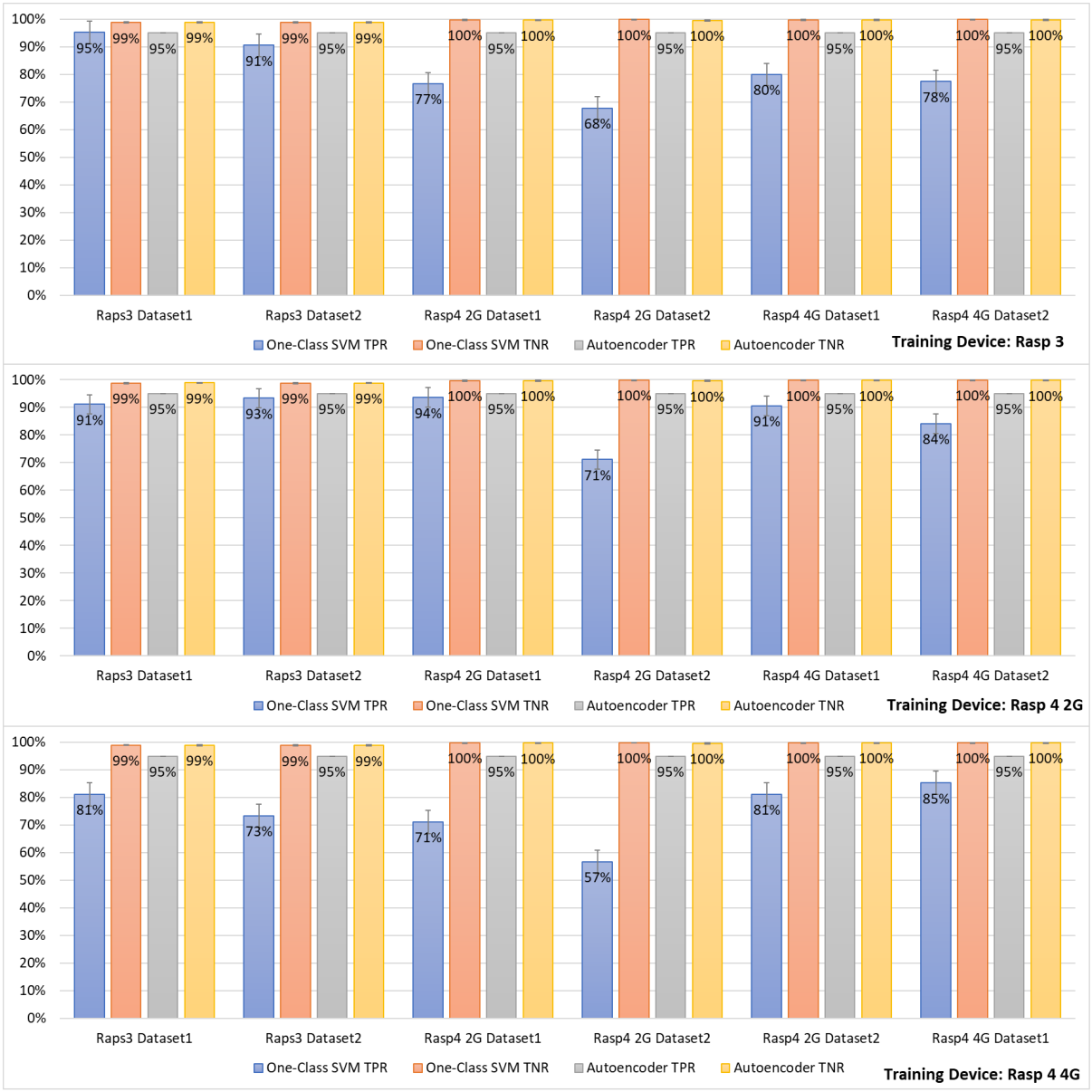

Figure 6.4: Performance of leave-two-devices models in different datasets

When the percentage of known devices is increased to 66.67%, as shown in Figure 6.5, the performance of the algorithms improved on each dataset. For the Autoencoder algorithm, its performance is quite stable, i.e., the TPR is close to 95%, while the TNR is close to 100%.

For One-class SVM, its performance also has a corresponding improvement due to the increase of training data. At the same time, the performance on different datasets is more balanced because more datasets are covered. Compared with the leave-two-devices method, the average TPR of the One-class SVM algorithm on the Raspberry Pi 3 dataset

improves from 87% to 92%, on the Raspberry Pi 4 2G dataset improves from 70% to 72%, and on the Raspberry Pi 4 4G dataset improves from 83% to 84%. Meanwhile, the standard deviation of TPR on different datasets decreases from 0.11 to 0.08, and it implies the algorithms performed more smoothly.

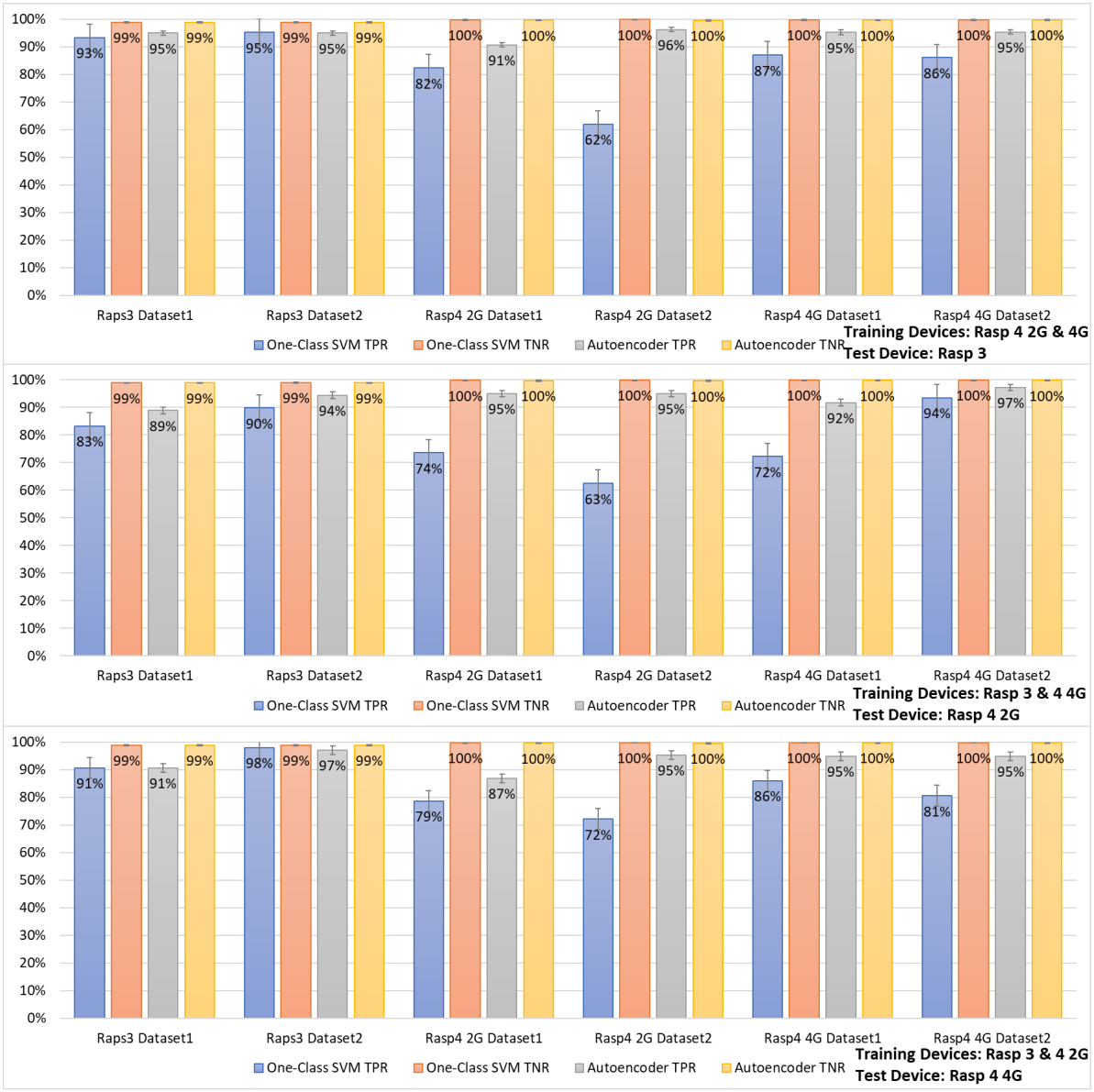

Figure 6.5: Performance of leave-one-device models in different datasets

When the training data includes all types of devices, as shown in Figure 6.6, both One-class SVM and Autoencoder algorithms are capable of identifying the attack data contained in different datasets well. On the one hand, the detection model based on Autoencoder algorithm performs well on all datasets. On the other hand, for the One-class SVM, compared with the leave-one-device approach, the average TPR on the Raspberry Pi 3 dataset is equal with 92%, but on the Raspberry Pi 4 2G dataset, it improves from 72% to 78%, and on the Raspberry Pi 4 4G dataset improves from 84% to 85%. Meanwhile,

they both perform consistently in different validation experiments.

In conclusion, these FL-based approaches are able to effectively solve the non-IID problem and also protect user privacy well due to their special model update strategies. Based on the experiments, the performance of the model rises with the increase of known devices in the training clients, while the stability of the model on each dataset also ascends. On the other hand, DL-based Autoencoder performs better than ML-based One-class SVM, especially in terms of identifying normal data.

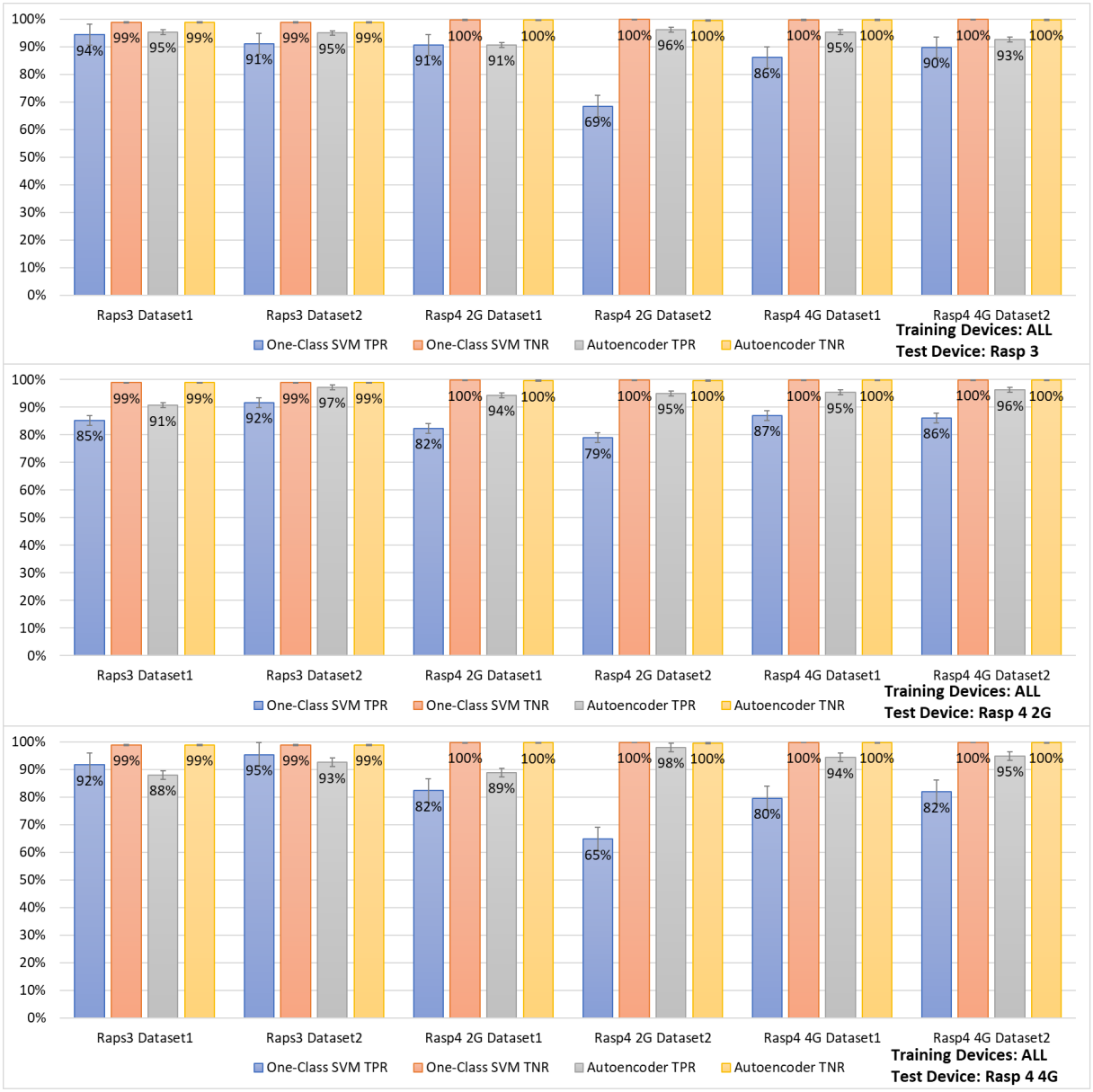

Figure 6.6: Performance of leave-one-dataset models in different datasets

6.3 Case Study #3: Central Model

How does the model perform if the user is not concerned about data privacy? In this scenario, the clients are not privacy-preserving and willing to upload the data to the server for central model training. After the training is completed, the server pushes the model to each client for cyberattack identification. The efficiency of the FL approach in attack identification is verified by comparing them with the central model.

Table 6.2: Comparison of the performance of the model under different setups

	One-Class SVM						**Autoencode**					
	TPR (%)		**TNR(%)**		**F1-score (%)**		**TPR (%)**		**TNR(%)**		**F1-score (%)**	
	Avg.	**Std.**	**Avg.**	**Std.**	**Avg.**	**Std.**	**Avg.**	**Std.**	**Avg.**	**Std.**	**Avg.**	**Std.**
Local Model	64.39	28.93	99.87	0.09	76.93	31.65	78.84	16.01	99.78	0.15	87.30	23.46
FL with Known Devices (33.33%)	81.13	10.22	99.52	0.46	87.72	15.93	95.00	0.00	99.44	0.41	95.23	1.11
FL with Known Devices (66.67%)	82.63	10.39	99.54	0.45	88.72	17.01	93.90	2.78	99.44	0.39	94.68	7.39
FL with Known Devices (100.00%)	84.87	8.01	99.53	0.45	89.98	13.10	93.97	2.73	99.45	0.39	94.74	7.53
Central Model	97.70	3.81	99.38	0.29	96.40	12.28	97.78	3.68	99.35	0.29	96.34	11.93

In this experiment, the datasets of the three devices are aggregated and trained to a central model uniformly. The performance of the models under different setups are listed in Table 6.2. By comparing the TPR, TNR and F1-score of the models in different cases, it is clear that the central model is the upper bound of the different models, while the Local model is the lower bound. Since the central model contains the most variety of data, thus both the One-class SVM algorithm and the Autoencoder algorithm achieve high performance. Several FL-based approaches, on the other hand, perform between local and central. For One-class SVM, the performance of the model increases significantly as the number of data increases. As for Autoencoder, its performance is relatively stable, and the FL-based strategy has similar results to the central approach model. Through the experiments, it also shows that although the recognition performance of the FL-based mechanism is inferior compared to the central model, it is still very effective in detecting network attacks in IoT environments.

6.4 Discussion

This chapter quantitatively evaluates the FL-based cyberattack detection approaches as documented in the previous sections. Since the distribution of system calls differs in devices types and versions, it leads to the fact that the model based on local data cannot effectively detect attacks in other devices. Besides, taking into account the privacy-preserving of the clients, the FL-based technique becomes a powerful solution. Nevertheless, the actual performance of the FL-based model is slightly inferior to that of the

central model but still far better than local models and has an excellent performance in identifying both normal and abnormal data. In general, the more clients involved in the computation and the more device models, the better the FL model performs and the more it can overcome the problem of imbalanced data distribution. It is also observed that the Autoencoder-based models perform significantly better and more consistently than the traditional ML models. The Autoencoder model with FL training strategy is only marginally less effective than the central model. Based on the above facts, this work concludes that the FL-based cyberattacks identification framework can potently detect SSDF cyberthreats and protect the spectrum data integrity in the RF spectrum monitoring platform.

Chapter 7

Summary, Conclusions and Future Work

This chapter summarizes the entire project and draws the main conclusions of this work, together with an outlook for future work. The first section summarizes and concludes the main work of this project and then draw some conclusion about the proposed system. An overview of the possible future directions of this work is presented in the second part.

7.1 Summary and Conclusion

The main goal of this work is to design and implement a privacy-preserving SSDF cyberattacks detection system by exploiting the FL-based AI approach with system calls behavior fingerprinting. To achieve this goal, several works have been done.

First, the background knowledge of IoT RF spectrum monitoring networks and device behavior fingerprinting are introduced. Second, this work has systematically summarized the state-of-the-art works that adopt the system calls behavioral fingerprinting to detect cyberattacks. Third, a set of use cases have been analyzed to extract the requirements of the cyberattack detection system in IoT RF spectrum monitoring networks. After that, an intelligent detection system architecture has been sketched, which is used for defining the structure, designing the functionality, and researching the feasibility of the system. Following, this work analyzes the system calls distribution among different types of sensors. Data exploration and visualization revealed the problem that different system calls are not equally distributed among devices, which became necessary to use the FL-based approach. Furthermore, this work evaluates the feature extraction methods and detection algorithms in terms of both computational cost and detection performance. Finally, a quantitative analysis of the FL-Based system is performed using different scenarios to test and validate the effectiveness of cyberattacks affecting RF spectrum data integrity.

To conclusion, this thesis has the following findings:

- First, by comparing various types of feature extraction approaches from the aspects of computational cost and performance, this work finds that in the IoT environment

with resources-limited sensors, both sequential-based and graph-based feature extraction approaches consume a large number of system resources. Therefore, these two approaches are not practical for detecting SSDF cyberattacks in real IoT devices. Meanwhile, since there are obvious differences between normal data and attack data in terms of the distribution of system calls behavior, the feature extraction method using 1Gram Frequency is well suited for the task of cyberattack identification.

- Second, the distribution of system calls on different devices does not follow IID, and therefore the model using local data cannot be generalized to data from other devices.
- Third, considering users' privacy, the FL-based approach shares only the model but not the user data and thereby is particularly suitable for the context to which this article applies. Collaboration and sharing can effectively improve the performance of detection models. Experiments have proved that even only the model is shared, the performance of the system can be significantly improved. Compared with the local model, the FL system can effectively overcome the non-IID problem and achieve comparable performance to the central model.
- Fourth, DL-based Autoencoder algorithms are more robust than traditional ML techniques, and it is more adaptable to the case of unknown devices than ML methods.
- Lastly, The integration of more diverse clients into the collaborative FL approach can significantly boost the performance of the detection system.

In conclusion, the proposed FL-based cyberattack detection method is capable of accurately identifying SSDF cyberattack data while protecting data security.

7.2 Future Work

Although the proposed system has been optimized in many aspects considering the computational power of Raspberry Pi devices, there are still some tasks that cannot be done directly on sensors, such as the fitting of feature extraction approaches and the FL part is also simulated in the server. Therefore, this work hopes to further improve the method to transform the proposed system into an end-to-end solution without processing data in the server.

On the other hand, the main work of this article identifies whether the system suffers from a cyberattack by means of unsupervised learning, but not to identify the types of specific attacks. A supervised multi-label classification approach based on the FL framework will be made to identify the type of attacks in the following work.

Bibliography

[1] Muhamed Fauzi Bin Abbas and Thambipillai Srikanthan. "Low-complexity signature-based malware detection for IoT devices". In: *International Conference on Applications and Techniques in Information Security*. Springer. 2017, pp. 181–189.

[2] Abada Abderrahmane et al. "Android malware detection based on system calls analysis and CNN classification". In: *2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)*. IEEE. 2019, pp. 1–6.

[3] Charu C Aggarwal. "An introduction to outlier analysis". In: *Outlier analysis*. Springer, 2017, pp. 1–34.

[4] ASM Ahsan-Ul-Haque, Md Shohrab Hossain, and Mohammed Atiquzzaman. "Sequencing system calls for effective malware detection in android". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2018, pp. 1–7.

[5] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[6] Géron Aurélien. "Hands-on machine learning with scikit-learn & tensorflow". In: *Geron Aurelien* (2017).

[7] Dominik Breitenbacher et al. "HADES-IoT: A practical host-based anomaly detection system for IoT devices". In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 2019, pp. 479–484.

[8] Gerardo Canfora et al. "Detecting android malware using sequences of system calls". In: *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*. 2015, pp. 13–20.

[9] Alberto Huertas Celdran et al. "CyberSpec: Intelligent Behavioral Fingerprinting to Detect Attacks on Crowdsensing Spectrum Sensors". In: (2021).

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[11] Andrea De Lorenzo et al. "Visualizing the outcome of dynamic analysis of Android malware with VizMal". In: *Journal of Information Security and Applications* 50 (2020), p. 102423.

[12] Prachi Deshpande et al. "HIDS: A host based intrusion detection system for cloud computing environment". In: *International Journal of System Assurance Engineering and Management* 9.3 (2018), pp. 567–576.

[13] Marko Dimjašević et al. "Evaluation of android malware detection based on system calls". In: *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*. 2016, pp. 1–8.

[14] Manuel Egele et al. “A survey on automated dynamic malware-analysis techniques and tools”. In: *ACM computing surveys (CSUR)* 44.2 (2008), pp. 1–42.

[15] *Enabling abundant wireless connectivity for everyone*. URL: `https://www.google.com/get/spectrumdatabase/`.

[16] Simone Facchini et al. “Multi-level Distributed Intrusion Detection System for an IoT based Smart Home Environment.” In: *ICISSP*. 2020, pp. 705–712.

[17] Yoav Goldberg. “Neural network methods for natural language processing”. In: *Synthesis lectures on human language technologies* 10.1 (2017), pp. 1–309.

[18] Martin Grimmer et al. “Intrusion Detection on System Call Graphs”. In: *Sicherheit in vernetzten Systemen, pages G1–G18* (2018).

[19] Dang Kien Hoang, Duy Loi Vu, et al. “IoT Malware Classification Based on System Calls”. In: *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*. IEEE. 2020, pp. 1–6.

[20] Shifu Hou et al. “Hindroid: An intelligent android malware detection system based on structured heterogeneous information network”. In: *Proceedings of the 23rd ACM SIGKDD International conference on knowledge discovery and data mining*. 2017, pp. 1507–1515.

[21] Bojan Kolosnjaji et al. “Deep learning for classification of malware system call sequences”. In: *Australasian joint conference on artificial intelligence*. Springer. 2016, pp. 137–149.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[23] Jianwu Li et al. “A survey of security issues in cognitive radio networks”. In: *China Communications* 12.3 (2015), pp. 132–150.

[24] Yingbin Liang, H Vincent Poor, and Shlomo Shamai. *Information theoretic security*. Now Publishers Inc, 2009.

[25] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation forest”. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation-based anomaly detection”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), pp. 1–39.

[27] Zhen Liu et al. “A statistical pattern based feature extraction method on system call traces for anomaly detection”. In: *Information and Software Technology* 126 (2020), p. 106348.

[28] Brendan McMahan et al. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 20–22 Apr 2017, pp. 1273–1282. URL: `https://proceedings.mlr.press/v54/mcmahan17a.html`.

[29] Francesca Meneghello et al. “IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices”. In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8182–8201.

[30] Preeti Mishra et al. "Vmguard: A vmi-based security architecture for intrusion detection in cloud environment". In: *IEEE Transactions on Cloud Computing* 8.3 (2018), pp. 957–971.

[31] Anand Mudgerikar, Puneet Sharma, and Elisa Bertino. "E-spion: A system-level intrusion detection system for iot devices". In: *proceedings of the 2019 ACM Asia conference on computer and communications security*. 2019, pp. 493–500.

[32] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. "Anomaly detection from system tracing data using multimodal deep learning". In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE. 2019, pp. 179–186.

[33] Thien Duc Nguyen et al. "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.

[34] Guansong Pang et al. "Deep learning for anomaly detection: A review". In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–38.

[35] *Perf.* URL: `https://perf.wiki.kernel.org/index.php/Main_Page`.

[36] Alexandru RADOVICI, RUSU Cristian, and Rãzvan ŞERBAN. "A survey of iot security threats and solutions". In: *2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE. 2018, pp. 1–5.

[37] Sreeraj Rajendran et al. "Electrosense: Open and big spectrum data". In: *IEEE Communications Magazine* 56.1 (2017), pp. 210–217.

[38] Sreeraj Rajendran et al. "Unsupervised wireless spectrum anomaly detection with interpretable features". In: *IEEE Transactions on Cognitive Communications and Networking* 5.3 (2019), pp. 637–647.

[39] Valerian Rey et al. "Federated learning for malware detection in iot devices". In: *arXiv preprint arXiv:2104.09994* (2021).

[40] Khaled Riad, Teng Huang, and Lishan Ke. "A dynamic and hierarchical access control for IoT in multi-authority cloud storage". In: *Journal of Network and Computer Applications* 160 (2020), p. 102633.

[41] Peter J Rousseeuw and Katrien Van Driessen. "A fast algorithm for the minimum covariance determinant estimator". In: *Technometrics* 41.3 (1999), pp. 212–223.

[42] Pedro Miguel Sánchez Sánchez et al. "A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets". In: *IEEE Communications Surveys & Tutorials* (2021).

[43] Bernhard Schölkopf et al. "Estimating the support of a high-dimensional distribution". In: *Neural computation* 13.7 (2001), pp. 1443–1471.

[44] Bernhard Schölkopf et al. "Support vector method for novelty detection." In: *NIPS*. Vol. 12. Citeseer. 1999, pp. 582–588.

[45] M Shobana and S Poonkuzhali. "A novel approach to detect IoT malware by system calls using deep learning techniques". In: *2020 International Conference on Innovative Trends in Information Technology (ICITIIT)*. IEEE. 2020, pp. 1–5.

[46] Nathan Shone et al. “Misbehaviour monitoring on system-of-systems components”. In: *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS)*. IEEE. 2013, pp. 1–6.

[47] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. “Detecting behavioral change of IoT devices using clustering-based network traffic modeling”. In: *IEEE Internet of Things Journal* 7.8 (2020), pp. 7295–7309.

[48] Arunan Sivanathan et al. “Classifying IoT devices in smart environments using network traffic characteristics”. In: *IEEE Transactions on Mobile Computing* 18.8 (2018), pp. 1745–1759.

[49] Alex J Smola and Bernhard Schölkopf. “A tutorial on support vector regression”. In: *Statistics and computing* 14.3 (2004), pp. 199–222.

[50] Petre Stoica, Randolph L Moses, et al. “Spectral analysis of signals”. In: (2005).

[51] Benjamin Berry Thompson et al. “Implicit learning in autoencoder novelty assessment”. In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*. Vol. 3. IEEE. 2002, pp. 2878–2883.

[52] Fei Xiao et al. “Malware detection based on deep learning of behavior graphs”. In: *Mathematical Problems in Engineering* 2019 (2019).

[53] Qiang Yang et al. “Federated machine learning: Concept and applications”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[54] Jixin Zhang et al. “Sensitive system calls based packed malware variants detection using principal component initialized MultiLayers neural networks”. In: *Cybersecurity* 1.1 (2018), pp. 1–13.

[55] Mariya Zhivkova Zheleva et al. “Enabling a Nationwide Radio Frequency Inventory Using the Spectrum Observatory”. In: *IEEE Transactions on Mobile Computing* 17.2 (2018), pp. 362–375. DOI: 10.1109/TMC.2017.2716936.

Abbreviations

AI	Artificial Intelligence
AUC	Area under the Curve
CNN	Convolutional Neural Networks
CRN	Cognitive Radio Networks
CPU	Central Processing Unit
CSG	Communication Systems Group
DL	Deep Learning
DNN	Deep Neural Networks
DDoS	Distributed Denial of Service
DoS	Denial of Service
FL	Federated Learning
GB	Gigabyte
IID	Independent and Identically Distributed
IoT	Internet-of-Things
KB	Kilobyte
k-NN	k-Nearest Neighbor
LASSO	Least Absolute Shrinkage and Selection Operator
LOF	Local Outlie Factor
LSTM	Long-Short Term Memory
MB	Megabyte
ML	Machine Learning
MSE	Mean Squared Error
NLP	Natural Language Processing
NN	Neural Networks
RF	Radio Frequency
OS	Operating System
PUEA	Primary Users Emulation Attack
RNN	Recurrent Neural Networks
SSDF	Sensing Data Falsification Attack
SVM	Support Vector Machines
TF-IDF	Term Frequency-Inverse Document Frequency
TNR	True Negative Rate
TPR	True Positive Rate
UZH	University of Zurich

List of Figures

List of Tables

Appendix A

Installation Guidelines

This part provide a guideline for install this system to the clients and the server. This installation guideline contains three parts, the first part is to install system calls monitoring module to the sensors, the second part is to extract the features from the monitored raw data, and the last is to use the ML-based, DL-based, and the FL-based approaches to detect SSDF attacks from the data. The following shows the outline structure of the system:

```
|—— sensor
    |—— get_system_perf.sh
    |—— monitoring.sh
    |—— run_background .sh
|—— datamodule
    |—— preprocessing_pref.py
    |—— get_features.py
    |—— normalization.py
    |—— get_pac.py
|—— detectionmodule
    |—— ml
    |—— fl
```

A.1 System calls Monitoring Module

The *sensor* folder contains all required scripts for monitoring the system calls from the Raspberry Pis.

Initial setup

First the Perf tool needs to be installed to the Raspberry Pis:

```
sudo apt-get install perf
```

Then type *perf* to the terminal to test whether it has been successfully installed, if not, then the exec file of perf need to be modified:

```
sudo nano /usr/bin/perf
#exec "perf_$version" "$@"
exec "perf_4.9" "$@"
```

Clone GitHub Repository

The GitHub repository of this project could be cloned as:

```
git clone git@github.com:luke-feng/IoT_Sensors_Security_Analysis.git
```

Data Monitoring

For each script within this folder, the functionality is:

- *get_system_perf.sh* is responding for finding the desired PID, and monitoring it system calls. There are several parameters that can be modified to adopt different tasks.
  - *time_window* means how many seconds that need to be monitored each time, by default it's 60s.
  - *total_loop* how many loops that need to be monitored for this process, by default it's 360 loops.
- *monitoring.sh* is responding for copy the attack files to the service, and automatically start the *get_system_perf* service to monitor the system calls of attack/normal services.
- *run_background.sh* could run this system calls monitoring script in background.

If the attack files are contained with these three scripts in a same folder, then the monitoring module could be easily start with:

```
mkdir dataset
./run_background.sh
```

A.2 Data Module

Data module is used to extract the features from the monitored system call files. For each script within this folder, the functionality is:

- *preprocessing_pref.py* is used to remove the non-relevant data from the raw data, and keep the system calls only.
- *get_features.py* is used to convent the raw data to the features, and several feature extraction approaches are used, including the frequency, TF-IDF, One-hot, Dict-index, Dependency-graph
- *normalization.py* is used to normalize the features to [0,1] space.
- *get_pca.py* is used to fit and transform the PCA from high-dimension features.

A.3 Detection Module

Detection module is responding for using AI techs to identify the cyberattacks from the data.

Machine Learning Based Approaches

Four ML-based and one DL-based algorithms are used in ML-based approaches, including: One-class SVM, Isolation Forest, Robust Covariance, SGD One-class SVM, as well as the Autoencoder.

- $ml - based.py$ is used to train and test the ML-based models.
- $dl - based.py$ is used to train and test the DL-based models.

Federated Learning Based Approaches

Project dependencies (such as $scikit - learn$ and $flwr$) are defined in *pyproject.toml*. It can be install by:

```
poetry install
poetry shell
```

Afterwards it is ready to start the Flower server as well as the clients. Simply start the server in a terminal as follows:

```
poetry run python3 server.py
```

For clients, simply open two more terminals and run the following command in each:

```
poetry run python3 client$k$.py
```

Appendix B

Contents of the CD

The CD contains the all the documents, project source code, and the installation instructions for this project.