Passive IoT Device-Type Identification Using Few-Shot Learning

Zineb Meriem Ferdjouni

A Thesis
in
The Concordia Institute
for
Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Applied Science (Information Systems Security) at
Concordia University
Montréal, Québec, Canada

June 2023

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Zineb Meriem Ferdjouni

Entitled: Passive IoT Device-Type Identification Using Few-Shot Learning

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair
Dr. Suryadipta Majumdar

______________________________ External Examiner
Dr. Othmane Ait Mohammed

______________________________ Internal Examiner
Dr. Jun Yan

______________________________ Thesis Supervisor
Dr. Mourad Debbabi

Approved by ______________________________
Dr. , Graduate Program Director

____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract for Masters

Passive IoT Device-Type Identification Using Few-Shot Learning

Zineb Meriem Ferdjouni

The ever-growing number and diversity of connected devices have contributed to rising network security challenges. Vulnerable and unauthorized devices may pose a significant security risk with severe consequences. Device-type identification is instrumental in reducing risk and thwarting cyberattacks that may be caused by vulnerable devices. At present, IoT device identification methods use traditional machine learning or deep learning techniques, which require a large amount of labeled data to generate the device fingerprints. Moreover, these techniques require building a new model whenever a new device is introduced. To address these limitations, we propose a few-shot learning-based approach on siamese neural networks to identify IoT device-type connected to a network by analyzing their network communications, which can be effective under conditions of insufficient labeled data and/or resources. We evaluate our method on data obtained from real-world IoT devices. The experimental results show the effectiveness of the proposed method even with a small amount of data samples. Besides, it indicates that our approach outperforms IoT Sentinel, the state-of-the-art approach for IoT fingerprinting, by a margin of 10% additional accuracy.

Dedication

To my beloved family, I dedicate this thesis to each and every one of you, for your unwavering love, encouragement, and support throughout my academic journey.

To my father, thank you for instilling in me a passion for learning and for always believing in me, even when I doubted myself. Your unwavering belief in me and my abilities has given me the confidence to pursue my dreams and has inspired me to push myself beyond my limits. Your support, guidance and wisdom have been invaluable to me. Your hard work and sacrifices have allowed me to pursue my dreams, and for that, I appreciate you and I will be forever grateful.

To my mother, thank you for being my rock. Your unwavering support and encouragement have helped me to overcome even the toughest of challenges. I will never forget the countless sacrifices you have made to ensure our happiness and success. Thank you for always being there for us.

To my brother, thank you for being my sounding board and my confidante. Your unwavering support and encouragement have been a source of strength to me.

To my husband, thank you for always standing by my side, even when the demands of academia made it difficult. Your love, patience, and understanding have been a constant source of comfort to me.

To my son, thank you for being the light of my life and the reason why I strive to be the best version of myself every day. Your presence reminds me of the joy and wonder of life, and I am forever grateful for the opportunity to be your mother. I am blessed to have such

an amazing family, and I dedicate this thesis to each and every one of you.

To my supervisor, thank you for your guidance and support throughout my academic journey. From our first meeting, you have been a constant source of encouragement and support. Your commitment to my success has gone above and beyond what I could have ever expected from a supervisor. Your passion for research and dedication to your students have inspired me to strive for excellence in my work. I am so grateful for the time and effort you have invested in my development as a researcher. As I complete my thesis, I want to take this opportunity to thank you for your guidance, support, and mentorship throughout this journey. Your impact on my academic and personal growth has been immeasurable, and I will always be grateful for your contribution to my success. It has been an honor and a privilege to work with you.

With deepest gratitude and appreciation,

Meriem

Acknowledgments

I would like to express my appreciation and gratitude to everyone who has contributed to the successful completion of this thesis. Foremost, I would like to express my heartfelt appreciation and gratitude to my supervisor, Dr. Mourad Debbabi who has guided and supported me throughout my academic journey. His mentorship and expertise have been invaluable in shaping my research and helping me to achieve my goals.

I am also grateful to the members of my thesis committee, Dr. Jun Yan, Dr. Suryadipta Majumdar and Dr. Otmane Ait Mohamed for their time and efforts in reviewing and providing feedback on my work.

I would like to extend my appreciation to my colleagues and friends in the Security Research Centre (SRC), Dhiaa Rebbah, Badis Racherache, Abdullah Qasem, ElMouatez Billah Karbab, and Pratyusha Bhattacharya, who have supported me throughout my academic journey. Their encouragement and support have been a source of strength and motivation during these challenging times.

Finally, I would like to acknowledge the financial support provided by NSERC, which has made this research possible.

To everyone who has contributed to this thesis in one way or another, I offer my sincere thanks and appreciation. Your support has been instrumental in helping me achieve this significant milestone in my academic journey.

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of the research study, presenting the motivation, problem statement, contribution, and thesis organization.

1.1 Motivation

Nowadays, the number of connected devices in modern homes, industrial systems, and organizations continues to expand with the rapid growth of Internet of Things (IoT) devices. The increasing convergence of Information Technology (IT) and Operational Technology (OT) towards IoT also led to the rise of Industrial IoT (IIoT) as companies are continuously engaging in this new paradigm. IHS claims that more than 75 billion IoT devices will be connected in 2025 [8]. Moreover, to improve productivity and optimize expenses and profits, organizations are widely accepting and incorporating the Bring Your Own Device (BYOD) policy in the workplace, which is also creating a massive spike of interconnected devices. Unfortunately, this technological advancement is a double-edged sword, while it can be used to enhance productivity and solve problems, it can present serious security issues and challenges as these devices can be accompanied by a number of vulnerabilities and security risks whether the network is based on wireless or wired technologies. However, many of these vulnerabilities remain unpatched [9] and thus leave devices open to identity theft and can be exploited to connect a malicious device or impersonate a legitimate device, in many cases, by spoofing the Media Access Control (MAC) address or/and Internet Protocol (IP) address, that would eventually allow an adversary or an outsider to gain access to the targeted network. Once the access is gained, the adversary can perform several attacks. If these attacks are successfully launched, it may be possible and relatively easy to compromise a security aspect such as confidentiality, data integrity, origin integrity, or availability. For instance, confidentiality may be compromised when a rouge access point attack is successfully launched. Adversaries can install a malicious AP to impersonate a legitimate AP, and thus a Man-In-The-Middle (MITM) attack can be launched. Also, over time, attackers can collect confidential information that they can use for malicious purposes. As stated in [10], Mirai malware infected more than 600,000 IoT-vulnerable devices that were used to launch distributed denial-of-service (DDoS) attacks. Moreover, ZDNet and Cynerio reported that healthcare systems could be one of the most impacted by security risks. The latter found that more than half of IoT and the Internet of Medical Things (IoMT) are vulnerable, potentially impacting healthcare organizations and patients.

To protect the network against such attacks, an authentication mechanism needs to be applied. Unfortunately, traditional cryptographic-based solutions alone may enable the attacker to gain network access using broken encryption keys or by spoofing a legitimate device's identity due to authentication weakness [11, 12], while security protocols are also prone to some attacks [13]. In addition to existing cryptographic-based approaches, researchers have proposed authentication solutions based on fingerprinting to add a security layer to enhance network security. Nevertheless, most of the existing fingerprint-based authentication systems support user authentication only. To build a robust authentication system and keep the attack surface as small as possible, device authentication is essential before access to resources, services, or networks can be granted. However, authenticating devices based on their MAC address to grant devices access to the network is also limited since MAC addresses are visible and prone to theft. Therefore, there is a strong need to extend the fingerprinting approach to devices for identification and access management that can help organizations manage and secure the network without considering the IP or MAC addresses. Moreover, device fingerprinting techniques may be leveraged for other security purposes, such as anomaly detection and forensics (a detailed application overview will be provided in Section 2. Motivated by the above, it was of interest to investigate device fingerprinting in order to prevent such attacks for achieving network security. In this research, we focus on IoT devices, and the main goal is to develop a passive identification framework using machine learning techniques. Motivated by the above, it was of interest to investigate device fingerprinting in order to prevent such attacks for achieving network security. In this research, we focus on IoT devices, and the main goal is to develop a passive identification framework using machine learning techniques.

1.2 Problem Statement

IoT devices are rapidly permeating modern society, from personal to industrial usage. With the increasing number of connected devices, security issues have also been an increasing concern. A vulnerable device may be maliciously exploited and may eventually result in significant damage; hence IoT device-type identification is a crucial matter to investigate. In this thesis, we aim to design and implement a framework that allows us to correctly and automatically detect and distinguish IoT devices from non-IoT, and if detected as IoT; we identify the IoT device-type. Particularly, we leverage deep learning techniques to build a system capable of identifying the types of IoT devices based on network traffic analysis, even in the presence of encryption.

In this thesis, we seek to answer the following research questions:

- How to identify the best set of features that can correctly identify a device?
- How to distinguish between IoT devices and non-IoT devices?
- How can we leverage machine learning techniques to identify IoT device types?
- When an IoT device connects to a network, can we accurately identify the device-type?

For a network-connected device, our problem can be defined into two sub-problems as follows:

IoT Detection

In this phase, we seek to detect the presence of an IoT device by deciding whether the traffic is generated from an IoT or non-IoT, which helps us exclude non-IoT devices. This is a binary classification problem. Upon IoT device detection, IoT device-type identification is conducted.

IoT Device-Type Identification

Focusing on IoT devices, we seek to determine the device type that has likely generated a given traffic. This is One-to-Many mapping of a given device against a list of known devices.

1.3 Contributions

In this thesis, the contributions are as follows:

1. We design and provide an automatic solution for feature selection to identify the best set of features.
2. We design and implement an IoT device detection model based on deep learning.

3. We design and implement a realistic and effective novel deep learning based model using network traffic flows for device-type identification.

4. Models performance is evaluated on a real IoT dataset.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides background on IoT security, device fingerprinting, and presents an overview of different cyberattacks that may be prevented using device fingerprinting and its different applications. Chapter 3 summarizes and highlights recent related work. Chapter 4, describes the methodology, including the research design and deep learning techniques used for IoT identification.Chapter 5 presents the description of the dataset, implementation details, and illustrates the experiments and results. Finally, Chapter 6 concludes this thesis and discusses future work.

Chapter 2

Background

This chapter introduces the concepts discussed in this thesis. We also discuss IoT security and its implication for IoT device fingerprinting. For the rest of this thesis, we use the terms "device identification" and "device fingerprinting" interchangeably.

2.1 Internet of Things

2.1.1 Overview

The term "Internet of Things" (IoT), introduced by Kevin Ashton in 1999, refers to a network of interconnected devices, vehicles, buildings, and other objects embedded with sensors, actuators, and software that enable them to transfer and exchange data [14]. These devices can range from simple household appliances like thermostats and light bulbs to complex industrial machinery and autonomous vehicles. IoT devices play an important role in the digital transformation of various industries and the evolution of the information society. The prevalence of IoT devices is rapidly growing, and such devices are deployed in a wide range of applications and permeated various sectors, including smart homes, smart cities, smart grids, automobiles, healthcare, transportation and manufacturing. As depicted in Figure 1, the IoT market value is projected to grow from 157 billion USD in 2016 to 771 billion USD by 2026 [1]. McKinsey [15] estimates that 127 new IoT devices connect to the

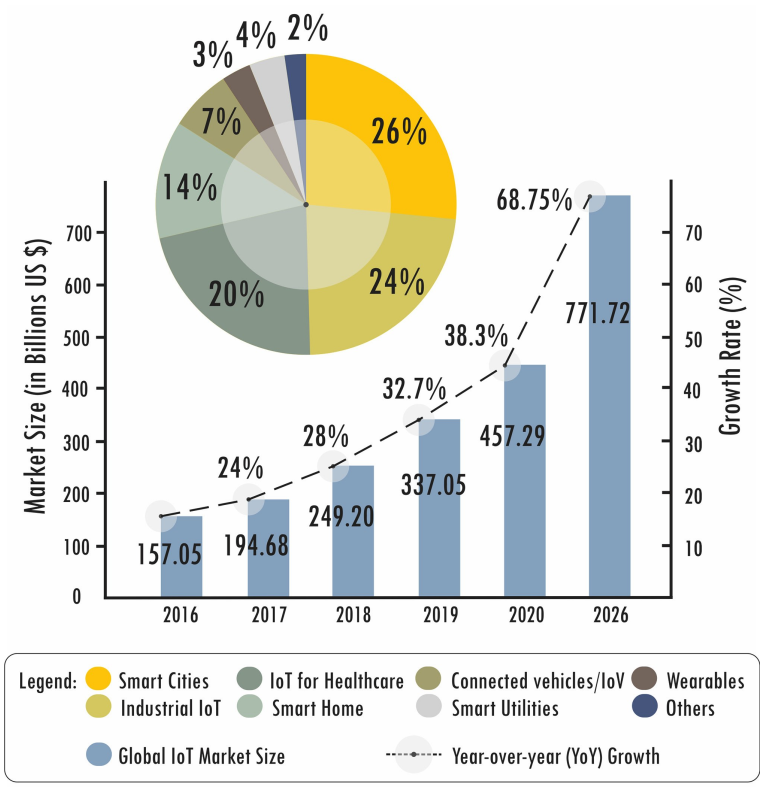


Figure 1: Global IoT Market Size by 2026 [1].

internet every second. Experts predict that the interconnected devices will reach 43 billion by 2023 [16], more than 75 billion in 2025 [17], and up to 500 billion by 2030 [1]. IoT devices can be broadly categorized into three classes based on their scope and functionality:: 1) Consumer IoT, 2) Industrial IoT, and 3) Commercial IoT.

Consumer IoT

Consumer IoT refers to inter-connected devices designed for personal use to enhance convenience, comfort, and efficiency in daily life. Examples range from personal and wearable devices, such as smartwatches, to home devices and appliances (e.g., cameras, smart lamps, refrigerators, heaters, etc.).

Commercial IoT

Commercial IoT focuses on integrating IoT technologies into businesses and various commercial environments. This sector includes retail, healthcare, transportation, hospitality, and other industries where IoT devices and systems are leveraged to optimize operations, enhance customer experiences, and drive innovation. Commercial IoT applications range from inventory management and supply chain optimization to intelligent surveillance systems, smart retail solutions, and real-time asset tracking. By harnessing the power of IoT, commercial enterprises can achieve increased operational efficiency, cost savings, and improved decision-making capabilities.

Industrial IoT

The Industrial Internet of Things (IIoT), also known as the Industrial Internet, represents the application of IoT in industrial settings, such as manufacturing plants, energy grids, logistics, supply chain management, and critical infrastructure. This integration of IoT devices and industrial systems enables interconnection among industrial devices and processes to analyze, automate, monitor, and exchange system information, providing the ability to enhance industrial productivity and manufacturing across several industries such as healthcare, transportation, oil and gas, aviation, energy/utilities, etc.

2.1.2 Internet of Things Security

As the number of IoT devices continues to grow, so do the opportunities and challenges that come with them. IoT has ushered in a new era of connectivity and automation, where devices seamlessly communicate and collaborate to enhance various aspects of our lives. However, the widespread adoption of IoT devices has also introduced significant security challenges. Many of these devices are often resource-constrained and have weak built-in security, which poses challenges for implementing strong security measures in IoT devices, making them susceptible to attacks. In addition, IoT networks expand the attack surface by connecting multiple devices, creating potential entry points for adversaries, which can then expose the network they are connected to. For example, the fish tank attack that helped hack and access a secured computer. The attackers attempted to obtain data from a computer in a casino by using a fish tank, and they managed to access 10 GB of data [18]. Furthermore, in the race to bring innovative IoT devices to market quickly and at a competitive price. This approach can leave IoT devices vulnerable to a wide range of attacks

IoT Vulnerabilities, Threats and Risks

An IoT vulnerability is a weakness or flaw in the design, implementation, or deployment of IoT devices, systems, or networks that can be exploited. These vulnerabilities can range from simple design flaws to more complex technical issues and have severe consequences for individuals and organizations. From hacking and data breaches to device manipulation and network disruption, the potential consequences of IoT security threats are significant and far-reaching. The increasing number of security threats to IoT devices has already had significant economic and privacy impacts for companies and users. In 2015, IBM Security Intelligence reported a vehicle hack. By exploiting vulnerabilities, researchers managed to control the vehicle, and they could send commands remotely through the CAN bus. Consequently, They could control the steering wheel, engine, transmission, and braking system. In 2016, Users reported unavailable websites, including Twitter, Netflix, Spotify, Airbnb, Reddit, Etsy, SoundCloud, and The New York Times, as a result of distributed denial of service (DDoS) attacks originating from vulnerable IoT devices, using Mirai botnet [19]. Figure 2 illustrates an overview of the Mirai attack.

In 2017, there was a 600 percent increase in IoT attacks [20]. According to [21], "The FDA confirmed that St. Jude Medical’s implantable cardiac devices have vulnerabilities that could allow a hacker to access a device. Once in, they could deplete the battery or administer incorrect pacing or shocks. The devices, like pacemakers and defibrillators, are used to monitor and control patients’ heart functions and prevent heart attacks." The

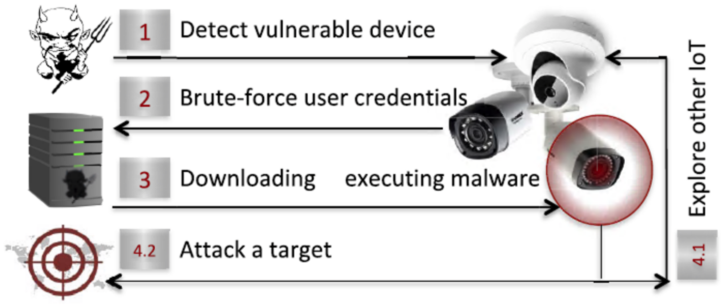

1 Detect vulnerable device
2 Brute-force user credentials
3 Downloading executing malware
4.2 Attack a target
Explore other IoT
4.1

Figure 2: Mirai Attack Process [2].

article continued, "The vulnerability occurred in the transmitter that reads the device's data and remotely shares it with physicians. The FDA said hackers could control a device by accessing its transmitter". Furthermore, HP revealed that 90 percent of IoT devices collect personal information and 70 percent of the most commonly used devices contain vulnerabilities [22].

Furthermore, experts expect the number of IoT-based attacks to grow as connected devices continue to rise. In 2020, Zscaler [23] released a report about the emerging threats of IoT. They blocked 14,000 IoT-based malware attempts per month, which increased by seven times in less than a year [23].

Types of Attacks on IoT

Generally, cyber attacks on IoT can be classified into three main categories: 1) attacks that compromise the confidentiality of the data, 2) attacks that compromise the integrity and authentication of connections, and 3) attacks that compromise access control and availability. The order in which security objectives are prioritized depends on the IoT sector, the severity of the attack, the compromised asset, and the sensitivity of the information stored. A non-exhaustive list of common attacks in IoT networks, along with their definition, is presented in Table 1.

Table 1: Common Attacks in IoT Network

Attack	Definition	Compromised Security Objective
Identity Spoofing	A computer has two traditional network identities, Internet Protocol (IP) and Media Access Control (MAC). Identity spoofing is computer identity theft that refers to the act of modifying the source IP/MAC address and imitating the legitimate addresses to mask the attacker's identity [24]. These addresses are usually spoofed to launch more sophisticated attacks such as DoS.	Authentication
Sybil Attack	Sybil attack is an impersonation attack where the attacker (the malicious node) attempts to gain control by forging multiple nodes' identities. Sybil attacks are easy to launch in wireless sensor networks (WSN) [25]. The threat of this type of attack to routing mechanisms was shown in [26]. The adversary can use the fake identities to disrupt a system or to fake data such as voting results and reputation evaluation.	Data Integrity and Availability
Denial of Service (DoS)	DoS attack seeks to make a system, a machine, a network, or a service unavailable to legitimate users. This attack can be launched using spoofed source IP addresses [27]. The malicious actor sends packets to a destination with the forged IP address of the target system; if the receiver responds, the response will be directed to the target. The ability to spoof the addresses is a core vulnerability exploited by many DDoS attacks [27].	Availability
Man-In-The-Middle (MITM)	MITM attack is a common attack in the context of network security. It allows the attacker to intercept a communication between two devices and lurk on the network silently and undetected making him able to eavesdrop and thus result in leaking valuable information and altering data [28].	Confidentiality and integrity (Origin and Data)
Malware	Malware is an attack that infects networked devices (IoT) turning them into botnets running Linux. This malware is often used to launch DDoS attacks [28].	Availability
Wormhole Attack	Wormhole attack is defined as a severe attack that can be launched without compromising any network host even if the network communication is encrypted [29]. This attack occurs when an attacker records a network packet in one location within a network and transmits it to another location where it will be replayed on both wired tunnels or wireless. The wormhole attack poses a serious threat against location-based security system.Wormhole attack make the attacker perform further attacks such as MITM, routing disruption, or DoS attacks.	Confidentiality, Integrity, and Availability
Replay Attack	Replay attack occurs when an unauthorized eavesdropper intercepts a data transmission, captures the network traffic, and maliciously or fraudulently retransmits it to the receiver impersonating the original sender [30]. Replay attacks may aid attackers in gaining access to the network or lead to completing a duplicate transaction. In this attack, the adversary is not required to decrypt a message.	Integrity
Fraud Attack	Cyberfraud may cover a huge range of activities. With the link between computers and the internet, we define online Fraud (Also known as Internet Fraud) is a type of cybercrime fraud that is committed using devices such as computers and the internet to defraud victims including businesses and individuals; it can involve identity theft and financial fraud [30]. Fraudsters can exploit poor security measures in IoT devices.	Integrity
Unauthorized Access	Unauthorized access attack occur when an unauthorized entity gains access to a system or data without permission. This can be done by exploiting vulnerabilities in systems/devices. IoT devices that are misconfigured or have weak passwords can also be exploited by attackers to gain unauthorized access [31].	Confidentiality, Integrity, and Availability

IoT Threats vs. IT Threats

While both IT and IoT networks face security threats that can compromise the confidentiality, integrity, and availability of sensitive information and systems, IoT threats introduce unique challenges due to the characteristics of interconnected devices and resource limitations. IT threats primarily target traditional information technology systems, including networks, servers, and software applications. These threats encompass familiar risks such as malware infections, phishing attacks, network breaches, and data theft. For example, a ransomware attack targeting an IT network can encrypt critical data and demand a ransom for its release, disrupting business operations and compromising data integrity. Although most attacks targeting IoT devices are similar to those targeting IT networks, IoT networks face unique security challenges. IoT networks are characterized by a diverse array of interconnected devices, each with its own vulnerabilities and communication protocols. This complexity creates a larger attack surface for potential exploitation. Moreover, IoT threats can have direct physical consequences. Compromised IoT devices may affect critical infrastructure, public safety, or personal well-being. For example, an attack on an industrial IoT system controlling power grids could lead to widespread power outages. In contrast, most IT threats primarily affect digital assets and services. For example, in 2021, an attacker tried to poison the water supply of a Florida city by gaining unauthorized access to the city’s water treatment systems and attempting to manipulate the levels of chemicals used in water treatment to dangerous levels [32].

2.2 Device Fingerprinting

2.2.1 Definition

Device fingerprinting (also referred to as device identification) is the process of identifying and distinguishing devices based on their unique characteristics and attributes. It involves collecting and analyzing various data points associated with a device, such as its hardware, network behaviour, and other identifying features. By analyzing these characteristics, a distinct "fingerprint" or digital signature is created for each device, enabling its identification and differentiation from other devices.

IoT Device-Type Fingerprinting

In the context of IoT, device fingerprinting refers to identifying and profiling IoT devices within a network. IoT device-type fingerprinting specifically focuses on identifying and categorizing IoT devices based on their device types or classes. It involves analyzing unique attributes associated with different device types. By employing IoT device-type fingerprinting techniques, organizations can classify IoT devices based on their specific types, allowing for targeted management, security policies, and compatibility considerations for each device category.

IoT Device Fingerprinting

IoT device fingerprinting is a process that involves uniquely identifying and profiling individual IoT devices within a network or system. It encompasses the collection and analysis of device-specific attributes. By creating a distinctive digital signature or "fingerprint" for each IoT device, IoT device fingerprinting enables accurate device identification, tracking, and management. It plays a crucial role in enhancing security and enabling effective device-specific access controls within IoT ecosystems.

Device fingerprinting can be classified into active and passive fingerprinting. Active device fingerprinting involves actively probing and interacting with a device to gather information and identify its unique characteristics. This technique typically requires sending specific requests or commands to the device and analyzing the responses received. In contrast, passive device fingerprinting does not involve any direct interaction with the device.

Instead, it relies on passive observing and analyzing network traffic or data generated by the device to gather information about its unique attributes.

The main difference between these methods is that passive fingerprinting can be used in conjunction with firewall systems, as passive sniffing will not alert or disturb the system.

2.2.2 Significance and Applications

IoT device fingerprinting is important in monitoring and improving network security. It enables us to identify unseen device types. In other words, when a new device connects to the network, fingerprinting helps analyze and detect its type. We can then grant or deny access and apply security measures based on known device vulnerabilities. According to [33], "Profiling tools are a requirement in the evolution of the IoT ecosystem. Their significance must not be downplayed, as IoT devices can be the point of access an attacker needs to initiate a large-scale attack."

Device Identity Validation

Identity spoofing or device forgery is when one device uses another device’s identifiers to gain authorized access or privilege. According to [34], "A naive solution to defending against node forgery is to verify the MAC address of the device against the legitimate ones." However, such traditional identifiers are easy to spoof. Once a device identity is forged, different attacks can be launched. Identity validation using device fingerprinting tools may reduce the risk of identity spoofing and identity fraud. Fingerprinting a device is crucial to enhance network security and protect it from existing threats, as it also helps distinguish whether a device is legitimate or malicious.

Authentication

Authentication is the process that verifies the identity of a user or a device and, therefore, the process of allowing or denying access to a system or a network. As discussed earlier, a traditional cryptographic solution such as encryption alone is insufficient to secure networks. Moreover, most organizations or networks focus only on authenticating users, not devices. Therefore, researchers have employed authentication mechanisms based on device fingerprinting methods to identify devices such as APs and IoT to enhance network security.

Unauthorized device Detection

Bring Your Own Device (BYOD) is a policy that allows employees to bring personal devices to an organization and connect to its network for work-related activities, which organizations increasingly accept. The IoT devices that employees bring and connect to the organization's network without authorization are called shadow IoT. BOYD and shadow IoT pose a security risk by increasing the attack surface. Another application of device fingerprinting is detecting unauthorized or rogue devices. It enables detecting connections to unauthorized servers or rogue devices attempting to gain network access; once detected, we can enforce security policies regarding the type of the device and isolate or block unauthorized devices.

Digital Forensics

The data source is becoming more important with the growing complexity of computer/device networks. In digital forensics investigation, device fingerprinting aims at identifying the device using the content/data it produces or/and receives. The device fingerprinting technique is of great importance in the field of digital forensics investigations. It can be used for forensic data collection in order to determine the content origin and recognize the device and/or its type used in criminal activity. Furthermore, a device fingerprint can be used as cybercrime evidence. For instance, fingerprinting can be used to identify the source device that has acquired a digital image or video [35]. According to [1], "However, unlike IoT security practices, forensics techniques do not aim to minimize the damage but to identify the attack/deficit origin or the liabilities of the different parties.".

Tracking and Digital Advertising

Cookies are the traditional and most common way of tracking users across multiple websites, "mostly for advertising purposes". Digital advertising or digital marketing is the act of delivering promotional content to users. Marketers need to be able to understand the behaviour of their users and identify their platforms; thus, most advertising tracking practices rely on cookies. However, cookies have evolved through the years and resulted in the rise of users' privacy concerns. Also, the reliability of cookies did not remain constant as, in many cases, users delete or block them. In addition, internet browsers provide the offer of limiting their use, and some enable tracking rejection by default. Consequently, device fingerprinting was explored to identify and track the devices of users online. For example, authors in [36] used fingerprinting techniques to track users. They collected data from device sensors to generate a device fingerprint. Their technique can be used to supplement other privacy tracking technologies, such as cookies or canvas fingerprinting. Device fingerprinting is also emerging in the advertising industry [37], which allows advertisers to create device and user profiles. It uses the data and parameters that a browser on a device sends to a website, such as screen resolution, device vendor, device operating system, etc. to develop a device's digital fingerprint. They can use the collected information and catered profiles to target users with personalized advertisements or to sell them to marketers.

Fraud Prevention

Device fingerprinting can also prevent several types of fraud in different areas, such as device fraud in online activity, credit card fraud in the financial industry or fraudulent customers in eCommerce fraud. The uniqueness of a device fingerprint can predict the fraudulent device even if the identity is altered through the use of proxies. According to [38], "Constructively, a correctly identified device can be used to combat fraud, e.g., by detecting that a user who is trying to login to a site is likely an attacker who stole a user's credentials or cookies, rather than the legitimate user."

Industrial Security

As stated in [39], "There is an increasing security concern that ICS devices are being vulnerable to malicious users/attackers, where any subtle changing or tampering attack would cause significant damage to industrial manufacturing.". Industrial Control Systems (ICS) manage critical infrastructures ranging from oil and gas refining, wastewater treatment and power grid. ICS may be composed of a vast number of devices connected with each other. While modern technologies, such as automation, are emerging in factories to facilitate and automate industrial process control, vulnerabilities in industrial systems increase cyber risk. Moreover, the growing number of IIoT devices increases the attack surface. The security of the electrical grid and other industrial systems can also be improved by device fingerprinting. It can be combined with other cybersecurity capabilities, such as security monitoring, to enhance the detection and mitigation of cyberattacks in the ICS network environment. For example, attackers can replace a real device with a fake device [39], they can inject false data and commands and result in unsafe consequences [40], such as blackouts in the power grid. They can also manipulate data in control components like data recorded in power metering equipment, which may lead to economic losses and environmental disasters [40, 41].

2.3 Machine Learning and Deep Learning Techniques

Machine Learning (ML) and Deep Learning (DL) are two rapidly growing fields within Artificial Intelligence (AI) that aim to give computers the ability to learn and make decisions without explicit programming. ML focuses on developing algorithms that enable computer systems to identify patterns and relationships in data, make predictions, and make decisions based on that information. On the other hand, DL takes ML a step further by using artificial neural networks to model complex relationships between input and output data. ML falls into two primary categories, namely supervised learning and unsupervised learning.

2.3.1 Supervised Machine Learning

Supervised machine learning, also known as supervised learning, is a type of machine learning in which the algorithm is trained using labeled data to make predictions about unseen data. The labeled data provides the algorithm with information about the correct output for a given input, so it can learn how to map inputs to outputs. In supervised learning, the algorithm is given a set of input-output pairs, and the goal is to learn a mapping function that can be used to predict the output for new inputs. The learning algorithm iterates through the training data, adjusting its parameters based on the errors it makes in until it reaches a level of accuracy that meets the desired performance criteria. Common applications of supervised learning include image classification, speech recognition, and predictive modeling. Decision Trees, Random Forest, Support Vector Machine (SVMs), Gradient Boosting, Naïve Bayes, and K-Nearest Neighbours (KNN) are common supervised techniques.

Decision Tree

Decision Tree is a non-parametric supervised learning model that can be used for classification and regression problems [42]. The model is composed of a set of questions based on a hierarchical tree and leaves representation. Each node represents a question/test, and each node output.

Random Forest

Random Forest [43] algorithm is widely used in supervised machine learning problems, including regression and classification. It combines the multiple decision trees where each tree is trained on a random subset of the training set. The prediction of each tree is then aggregated to reach a single result.

Support Vector Machine

SVM is a supervised learning algorithm used for classification and regression analysis [44]. It is a linear classifier that seeks to find the hyperplane that best separates the data into different classes. SVM is particularly well-suited for high-dimensional data where the number of features is much larger than the number of samples.

Gradient Boosting

Gradient Boosting is an ensemble learning method that combines the predictions of multiple simple models, such as decision trees, to produce a more accurate prediction. The algorithm works by training weak models, such as decision trees, one after the other. After each model is trained, the algorithm updates the weights of the training instances based on the accuracy of the previous model's predictions.

Naïve Bayes

Naive Bayes is a simple probabilistic classifier that is based on Bayes' theorem. Naive Bayes is well-suited for device identification tasks that involve a large number of features, as it can handle a high-dimensional feature space.

K-Nearest Neighbours

KNN is a simple classification algorithm that classifies a point based on the classes of its K nearest neighbors in the feature space [45]. KNN can be used for device identification tasks, especially if the data is well-separated into different classes.

2.3.2 Unsupervised Machine Learning

Unsupervised machine learning, also known as unsupervised learning, is a type of machine learning in which the algorithm is not given any labeled data, and the goal is to find patterns and relationships within the data on its own. Unlike supervised learning, unsupervised learning algorithms are not given any specific output to predict, but rather the goal is to extract meaningful insights and representations of the data. Unsupervised learning is often used in applications such as market segmentation, customer behaviour analysis, and fraud detection. Despite its benefits, unsupervised learning can be challenging because the results can depend on the choice of algorithm, the representation of the data, and the number of clusters or dimensions used. Some of the most common unsupervised learning techniques include:

Clustering: Clustering is a technique for grouping similar data points together. Common clustering algorithms include k-means and C-means (also known as fuzzy C-means). The main difference between the two algorithms is the way they assign points to clusters. In K-means, each point is assigned to exactly one cluster based on the distance to the cluster centroid. The assignment is binary, meaning that a point is either in one cluster or another. In contrast, in C-means, each point can belong to multiple clusters to a certain degree based on the membership values. The membership values are continuous values that represent the degree of membership of a point in a cluster, ranging from 0 to 1.

Principal Component Analysis (PCA): PCA is a dimensionality reduction technique that seeks to project the data onto a lower-dimensional space while preserving as much of the variance in the data as possible.

2.3.3 Deep Learning

Deep learning (DL) is a subfield of machine learning that is based on artificial neural networks with multiple layers. It is inspired by the structure and function of the human brain, and the goal is to build algorithms that can automatically learn and improve their performance on a task through experience. Deep learning algorithms can process large amounts of data, automatically identify patterns, and make predictions or decisions. Unlike traditional machine learning algorithms, deep learning algorithms can automatically learn a hierarchy of features from the raw data, and they are particularly well-suited for tasks that require the processing of unstructured data, such as images, audio, and text. Deep learning has achieved state-of-the-art results in a wide range of applications, including image and speech recognition, natural language processing, and cybersecurity such as intrusion detection, malware detection, and spam detection. There are several different types of neural networks, each with its own unique architecture and strengths:

Feed-Forward Neural Networks (FFNNs)

Feed-forward neural networks (also referred to as Multi-Layer Perceptron (MLP)) are one of the most widely used types of artificial neural networks, which have been successfully applied to a wide range of problems in various fields, including cybersecurity, natural language processing, and finance. An FFNN is composed of multiple layers of interconnected nodes or neurons, where information flows in one direction, from the input layer to the output layer, and the network learns to make predictions based on the relationships between the input and output variables [46]. MLPs are capable of learning complex non-linear relationships between the input and output, making them suitable for a wide range of tasks, including classification and regression. Figure 3 illustrates a simple example of an FFNN with one input layer, two hidden layers, and one output layer. The input layer receives the input data, which is then processed by the hidden layers, and the output layer generates the final output. Each neuron in the hidden layers and the output layer receives weighted inputs from the previous layer, applies an activation function to the weighted sum of inputs, and generates an output. The weights and biases of the neurons are learned during the training process using backpropagation, which is an iterative optimization algorithm that minimizes the error between the predicted output and the actual output. The simplicity and effectiveness of FFNNs have made them a popular choice for many real-world applications

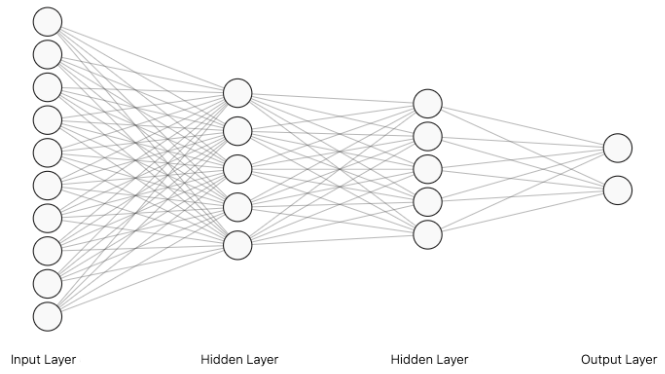


Figure 3: A Simple FFNNs Architecture with Two Hidden Layers.

Convolutional Neural Networks (CNNs)

CNNs are a type of deep neural network that has achieved remarkable success in various fields, including computer vision, speech recognition, and natural language processing [47]. CNNs are particularly useful for processing high-dimensional inputs, such as images or audio signals, by leveraging shared weights and local connectivity to extract spatial and temporal features. In the context of cybersecurity, CNNs have been applied to tasks such as intrusion detection, malware detection, and network traffic classification. The model 1D-CNNs, which are a variant of CNNs, have also been used to process one-dimensional signals, such as time series data, audio signals, or textual data. The architecture of a 1D-CNN is similar to a standard CNN, but the input and the filters are one-dimensional instead of two-dimensional. Figure 4 illustrates a 1D-CNN with one input layer, two convolutional layers, and one output layer. At the input layer, the one-dimensional signal, which is usually represented as a sequence of values, is fed into the network. The first convolutional layer applies a set of filters to the input in order to extract local patterns or features.

The second convolutional layer applies another set of filters to the feature maps produced by the first layer in order to capture higher-level features. The filters in the second layer have a larger receptive field, which means that they can capture more complex patterns that involve multiple features from the previous layer.

The output layer generates the final output, which can be a classification or regression result, depending on the task.

In summary, 1D-CNNs with multiple convolutional layers are powerful tools for processing one-dimensional signals and can learn to extract meaningful features from sequential data, which is useful for various applications, including cybersecurity.

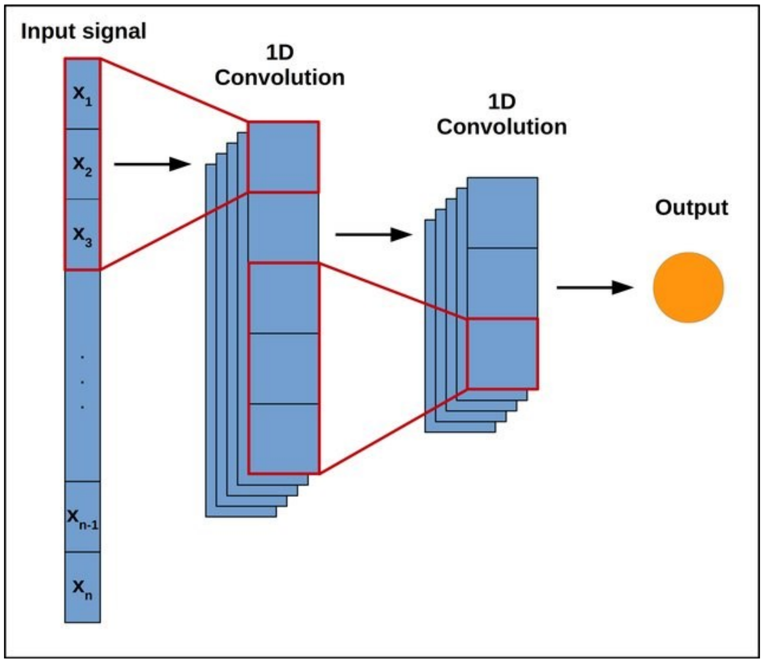


Figure 4: 1D-CNN Architecture with Two Convolutional Layers [3].

Recurrent Neural Networks (RNNs)

RNNs are a type of neural network architecture that has gained significant attention in recent years due to their ability to process sequential data such as natural language, speech, and time-series data. RNNs are characterized by their ability to maintain a hidden state that depends not only on the current input but also on all previous inputs. This makes RNNs particularly useful for tasks that require the processing of sequential data with variable-length inputs. RNNs have been successfully applied in a wide range of domains, including cybersecurity, such as intrusion detection, malware detection and fraud prevention. However, training RNNs with Backpropagation Through Time (BPTT) can be challenging due to the issue of vanishing or exploding gradients. Researchers have proposed various solutions to address this issue, including Long Short-Term Memory (LSTM) networks, which have shown significant improvements in the performance of RNNs on a variety of tasks.

Long Short-Term Memory (LSTM) networks

LSTM [48] networks are a type of RNN architecture that has become increasingly popular due to their ability to handle long-term dependencies in sequential data. LSTMs were designed to address the problem of vanishing or exploding gradients in standard RNNs. LSTMs introduce memory cells and gates that selectively control the flow of information within the network, allowing the network to selectively forget or remember information based on its relevance. LSTMs have been successfully applied in a wide range of domains, such as speech recognition, language modeling, and video analysis. The effectiveness of LSTMs in handling long-term dependencies has led to significant improvements in the performance of RNNs on a variety of tasks, and they continue to be an active area of research in the deep learning community. The architecture of a simple LSTM cell is shown in Figure 5, which is typically composed of a logistic sigmoid (σ), an input gate (i), a forget gate (f), an output gate (o) and a cell state (c) [4]. The input gate determines which information from the current input should be stored in the memory cell. The forget gate determines which information from the previous time step should be forgotten, the output gate determines which information should be output from the memory cell, and the state cell is responsible for storing information over time.

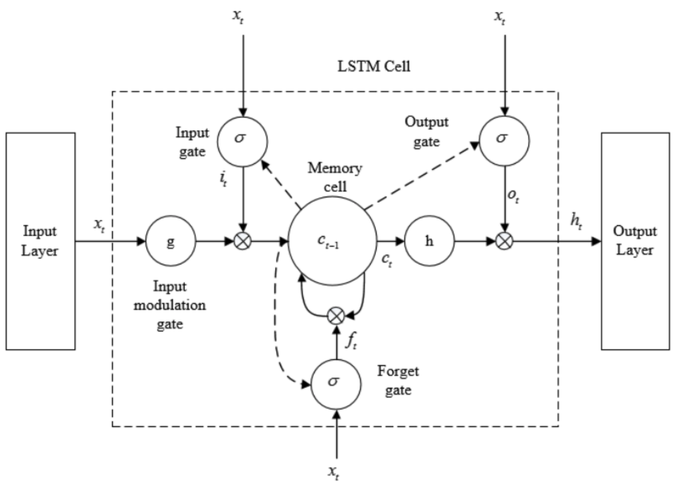


Figure 5: LSTM Cell [4].

Transformer Neural Network (TNN)

Transformer is a deep-learning model introduced in [5] as an alternative to RNNs and CNNs for processing sequential data. It is a type of neural network architecture that relies on a self-attention mechanism, which allows the model to weigh the importance of each input element when producing the output. This is in contrast to traditional RNNs or CNNs, which rely on fixed-size context windows to determine the impact of each element. The transformer architecture is well suited to parallelization, allowing for much faster training times and lower memory requirements compared to RNNs and CNNs, which are typically sequential in nature. This model is also used for sequential data. The Transformer architecture consists of an encoder and a decoder, both of which are composed of several layers of self-attention and feed-forward neural networks. In the encoder, the input sequence is processed by a series of identical self-attention layers in parallel, followed by a feed-forward neural network. Each self-attention layer takes the input sequence and computes a set of attention weights for each position in the sequence based on the relationships between all the positions. The attention weights determine how much importance should be given to each position when computing the output representation. The decoder is similar to the encoder but includes an additional masked self-attention layer that allows the network to attend only to the previously generated output sequence during training and to the previously generated tokens during inference. This is necessary to prevent the model from cheating by looking ahead in the output sequence. A detailed description of the model is available in [5]. The Transformer has gained popularity in natural language processing tasks, such as language translation, language modeling, and text classification, due to its ability to handle long-range dependencies, capture semantic relationships between words, and achieve state-of-the-art performance on several benchmark datasets. Moreover, the Transformer has shown promising results in other domains, such as image captioning, speech recognition, and recommendation systems. Therefore, the Transformer is an important area of research in machine learning and has numerous potential applications in cybersecurity, particularly in analyzing natural language text data for threat detection and prevention. Figure 6 shows the transformer architecture consisting of the encoder and decoder components, as well as the connections between them [5].

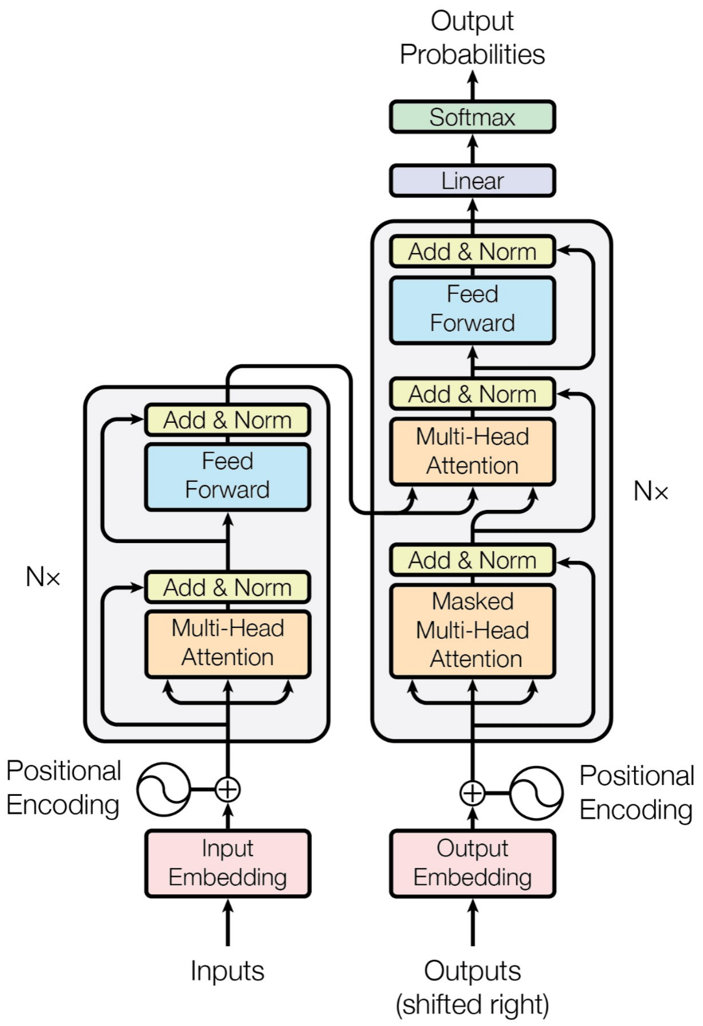


Figure 6: The Encoder-Decoder Architecture of the Transformer [5].

Chapter 3

Literature Review

In this chapter, we present a review of recent literature on IoT device fingerprinting. First, we provide a taxonomy of IoT fingerprinting techniques proposed in literature based on whether they are for fingerprinting an individual device or device-type. Furthermore, we introduce a taxonomy of machine learning and deep learning algorithms used for IoT device fingerprinting.

3.1 IoT Device Fingerprinting

There are two common methods of performing device fingerprinting: passive and active. The passive approach acquires data from a device through observation of its normal behaviour on the network without interacting with the devices. Differently, the active approach involves sending a probe request to a device and extracting features from its response. For instance, the method introduced by [49] utilizes active fingerprinting to collect packets of live hosts and uses the device fingerprint to detect IoT devices. In [50], active device fingerprinting based on protocol analysis was explored. The authors use active probing to investigate ICS devices and leverage the information encapsulated in the packet header of these protocols, such as device vendor, type, and function. They analyzed seventeen industrial protocols that operate over TCP or UDP to identify industrial control system devices. Relevant work in the scenario of passive fingerprinting was proposed by Marchal et al. [51]. The authors present AuDI (Autonomous IoT Device-Type Iidentification), an IoT device-type identification system by analyzing devices' network communications. Similarly, [7, 52] propose a passive analysis of traffic for device identification. As the work in this thesis relies on the passive approach, our review focuses on passive approaches. In this section, we present a taxonomy of fingerprinting techniques, as shown in Figure 7. There are three approaches currently being adopted in IoT device fingerprinting, including device-type fingerprinting (e.g. whether a device is an IoT or not), IoT device-type fingerprinting, and individual IoT device fingerprinting.

3.1.1 IoT or non-IoT

IoT enables different objects such as sensor nodes, embedded systems, and intermediate devices to collect and exchange data over the internet [53]. In the literature [54], more specifically the field of IoT device fingerprinting, IoT devices consist of devices that need to be connected to the internet to work properly, such as smart cameras, smart bulbs, smart fire alarms, etc. While non-IoT involves standard or general-purpose devices such as laptops, personal computers, tablets, and mobile phones.

Bremler-Barr et al. [55] suggest identifying whether a device is an IoT device or not whenever a new device connects to the network. In their work, 22 features from standard protocols (Link-Layer, IP, TCP, DNS, HTTP) are tested for feature selection. Then, they used three ML classifiers for identification. The first classifier is a Logistic Regression (LR) using traffic features, the second classifier is based on a Decision Tree on DHCP protocol information, and the third classifier is a unified classifier that leverages the advantages of the first and second classifiers. The unified classifier achieves 98% of F-score, precision, and recall. The authors of [56] propose ProfilIoT, a method to identify devices within a network by analyzing traffic data. The goal is to determine whether the traffic belongs to a non-IoT device category (computer or smartphone) or a specific IoT type. In [57], the authors trained a binary classifier to distinguish between IoT and non-IoT devices. The dataset is composed of 21 IoT devices and seven non-IoT devices, and the traffic was divided into two classes, IoT and non-IoT. They achieved 99% precision in distinguishing between IoT and non-IoT devices. In [58], the authors propose EvoIoT, an IoT and non-IoT identification model based on features from encrypted network traffic. They evaluate EvoIoT on two public datasets and a private dataset collected from a laboratory setting. Similarly, [59], [60] and [61] also focused on identifying new devices and predicting the kind of device that is being observed (IoT vs. Non-IoT).

3.1.2 IoT Device-Type Fingerprinting

device-type fingerprinting aims at identifying the device category such as camera, smart TV, smart doorbells, etc. Identifying the type of device connected to the network can help to reduce the attack surface. For example, knowing that a device-type X of model Y is vulnerable, the identification of this type helps in applying the necessary security measures. There are several research works that address the problem of identifying device-types. Miettinen et al. [7] propose IoT Sentinel, a system capable of identifying the types of devices being connected to a network using machine learning. The authors collected data from 27 IoT device-types during the setup of network communication. The authors collected data from 27 IoT device-types during the setup process and analyzed packet headers to extract 23 features resilient to encrypted traffic to generate device fingerprints. In the same line, Marchal et al. [51] presented AuDi, an autonomous system to identify IoT device-type based on its periodic network communications in SOHO (Small Office and Home) network. The system AuDi models the periodic communication traffic of IoT devices using an unsupervised learning method to perform identification. Ammar et al. [62] also presented an autonomous identification approach to identify new connected IoT device type to a home network. This method is based on network traffic extracted from network flow features and payload features. In [63], the authors present an IoT device-type behavioral fingerprinting using features extracted from the network traffic of the device. They have used a subset of the features from the set outlined by Miettinen et al in [7]. Additionally, they have considered payload features, i.e., payload size, entropy and TCP window size. These features are then used to train several ML classifiers, such as k-nearest-neighbors, Decision trees, and Gradient boosting. Yantian Luo et al. [64] propose an IoT device-type identification method based on the Transformer. The authors use two transformer-based models, one to classify the traffic from IoT devices into normal and abnormal; then, the second model is adopted on the normal traffic to identify the IoT device-type. Finally, they designed a results-ensemble algorithm to improve the accuracy of the IoT device-type identification model. IoT device-type has also been explored in prior studies by [56, 65–67].

3.1.3 IoT Device Fingerprinting

IoT device fingerprinting focuses on identifying the device itself individually. In this category, Sivanathan et al. [68] propose a framework for IoT device classification to uniquely identify a specific IoT device. Their approach uses statistical features derived from network traffic collected from 28 IoT devices. Using radio signals, Jafari et al. [69] propose a passive model to identify IoT devices and distinguish among devices from the same manufacturer. The authors exploit differences and imperfections that exist in devices that occurred during the manufacturing process. In the same line, IoT-ID [70] is another system that can be used for device identification that uses the variations in the manufacturing process to derive a unique fingerprint for devices. It is based on Physically Unclonable Functions (PUFs). In [71], the authors present an IoT fingerprinting technique based on the analysis of the entire physical signals emitted by the devices. The approach is designed to provide complementary protection against spoofing attacks in smart buildings, factories, or homes.

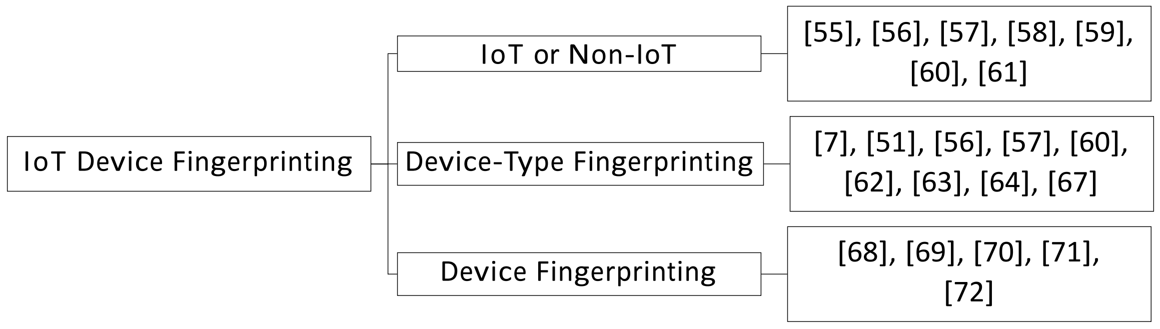


Figure 7: Taxonomy of IoT Device Fingerprinting Techniques

3.2 Machine Learning for IoT Fingerprinting

In recent years, researchers have applied Machine Learning (ML) and Deep Learning (DL) techniques in almost every research area. In this section, we focus on ML and DL techniques for IoT device-type identification. For each method, a description is provided, and a set of publications is presented. The most common types of machine learning taxonomy, as shown in Figure 8, include supervised learning, unsupervised learning, semi-supervised learning, and deep learning.

3.2.1 Supervised Machine Learning

Supervised learning relies on labelled data where given a set of known devices, the task is to find the function or the model that can accurately and uniquely determine whether the device is one of the known devices. broadly speaking, Supervised Learning can be mainly divided into two classes: classification problems and regression problems. To our knowledge, no studies have relied on regression algorithms for device identification problems.

Classification is a predictive modeling problem that assigns a class to a data sample. A number of existing studies in the literature have examined identification using multi-class classification. Common classification algorithms in IoT and device fingerprinting are Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), Gradient Boosting (GB), Naïve Bayes (NB), and k-nearest neighbours (kNN) [73], [74].

Supervised Artificial Neural Networks

Ruizhong Du et al. [75] propose a lightweight scheme to build an accurate IoT device identification based on flow statistical features. The authors then applied feature selection to select a valid subset of attributes. They evaluate the proposed method using three ML algorithms in the experiments, including the K-Nearest Neighbors algorithm (KNN), Random Forest and Extremely Randomized Trees (ET). Experimental results showed that the proposed scheme could achieve an accuracy of 99.3%. Identifying IoT devices and events based on packet length from encrypted traffic is proposed in [57]. In addition to the Majority Voting algorithm, the authors also evaluated the proposed solution using the common ML algorithms mentioned above. The results show that the Random Forest algorithm can achieve up to 96% accuracy in the identification of devices. Kostas et al. [76] presented IoT-DevID, a supervised machine learning system that recognizes IoT devices based on their network packets. They used six multi-class classifiers to evaluate the proposed solution. Similarly, several methods followed the same line of evaluation [51], [68], [52], [77], [78]. In [79], a practical IoT device identification system was presented, namely, ByteIoT, based on a simple but well-organized traffic feature, i.e., the frequency distribution of bidirectional packet lengths. The system ByteIoT applies the k-nearest-neighbors algorithm as the classifier.

Some authors suggest using One-vs-All for multi-class classification for device and device-type identification. Given a set of N known devices (classification problem with N classes), a one-vs-all classifier consists of N binary classifier, one binary classifier for each class/device. This approach is effective in detecting and classifying unknown class/device.

If a new device connects to the network, it should not match any existing binary classifier. Miettinen et al. [7] extracted network features based on setup communications to generate a fingerprint for each device. They used one-vs-all classifiers to identify the IoT device-type when a device connects to the network. Similar classification approach is presented in [56], [62], [80].

3.2.2 Unsupervised Learning

Previous studies have shown that supervised approaches are effective, and the majority of prior research has applied them. However, unknown devices (devices not included in the dataset) can not be recognized. Unlike supervised learning, the unsupervised method does not rely on labelled data. The goal is to find patterns in unlabeled data. Several works propose the application of unsupervised learning in device identification. The task is to map devices' fingerprints or profiles into groups based on their similarities. In [51], the authors propose AuDI (Autonomous IoT Device-Type Identification), an IoT device-type identification using an unsupervised machine learning method without labeled data to identify previously unseen device-types. This system uses a clustering algorithm to group device-types fingerprints into clusters. Thangavelu et al. [9] also presented IoT fingerprinting technique based on unsupervised machine learning. In this approach, network gateways perform device monitoring and classification while a clustering algorithm is applied to identify new device-types. In IoT-KEEPER [81], the authors also used unlabeled data to identify device-types. They employed an unsupervised approach based on fuzzy C-means clustering.

3.2.3 Semi-Supervised Learning

Semi-supervised learning is a type of machine learning that combines elements of both supervised and unsupervised learning. In this approach, the algorithms are trained on both labeled and unlabeled data, with the goal of leveraging the structure learned from the unlabeled data to improve the performance of the labeled data. Fan et al. [60] propose an IoT identification model based on semi-supervised learning. The model can classify specific IoT devices based on time interval features, traffic volume features, protocol features and TLS-related features.

3.2.4 Deep Learning

DL is widely considered to be a good way for classification problems. The main DL models that are frequently used in device identification are FFNNs, CNNs, and RNNs. In [69], the authors considered FFNN, CNN, and RNN models for IoT identification using radio frequency signal data. Better results are achieved using CNN, followed by FFNN and LSTM.

In Smart Recon [82], the authors used Locality Sensitive Hashes to generate a feature vector which will be trained by a neural network classifier (MLP) in fingerprinting IoT devices based on their generated network traffic. They claim that The Smart Recon method is able to identify known IoT devices with 98% accuracy using only a single packet sniffed from the network traffic flow.

Merchant et al. [83] used deep CNN to solve the identification and verification problems using time-domain complex base-band error signals. The model CNN was trained in two experiments using different data collection setups. In [77,84], the authors propose to define a device-specific unique fingerprint by analyzing solely the inter-arrival time of packets as a feature to identify a device. The model CNN is used on images of inter-arrival time (IAT) signatures for device fingerprinting of 58 non-IoT devices of 5-11 types. To evaluate the performance, they compared the ResNet-50 layer and basic CNN-5 layer architectures.

In [85], the authors propose an LSTM-CNN model to automatically identify the semantic type of a device. They used network traffic flows to classify new and unseen IoT devices. They have cascaded LSTM and CNN layers to conduct cross-device classification. They finally compare their model with different ML and DL classification techniques. They evaluate their approach by classifying 15 IoT devices into four types with real-world collected network traffic data and achieving an accuracy of 74.8% Similarly, Feihong Yin et al. present CBBI in [86], an IoT device identification approach based on Conv-BiLSTM using spatial and temporal features of the network traffic generated by the IoT devices. The system CBBI is a hybrid DL model exploiting CNN to learn the spatial characteristics of network communication traffic and bidirectional LSTM that can extract the time-domain characteristics of the network communication traffic. Furthermore, to tackle the problem of data imbalance, the authors use the GAN-based data augmentation module FGAN. Ortiz et al. propose DeviceMien, network device behavior modeling for identifying unknown IoT devices which uses the raw TCP payload as the input [59]. DeviceMien employs a stacked deep LSTM-Autoencoder to learn a set of representative features from the data itself. Then, they train a classifier on the labels assigned to the clusters. In [65], Jiaqi Bao et al. propose a hybrid supervised and unsupervised learning method combining deep neural networks with clustering to enable both seen and unseen device classification. The method also employs the autoencoder technique to reduce the dimensionality of data.

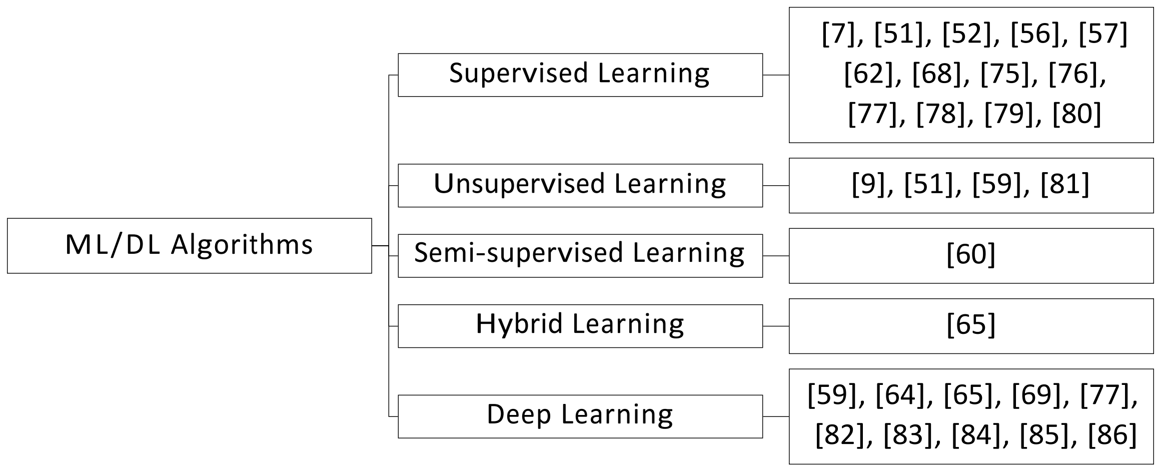


Figure 8: Taxonomy of ML/DL for IoT Device Fingerprinting

3.2.5 Conclusion

In this chapter, we presented a review of related work on the topic of IoT device identification and IoT device-type identification. We provided a review of the existing technologies that use network traffic traces and wireless signal patterns. We discuss existing non-cryptographic IoT device/device-type identification mechanisms from the perspective of machine learning and deep learning.

Chapter 4

Methodology

Having reviewed related work, we now present the main body of our research. In this chapter, we outline the specific methods used within this research. Firstly, in Section 4.1, we provide an overview of our approach. We then present the feature extraction process in Section 4.2 and the feature selection methods in Section 4.3, followed by details of the machine learning/deep learning models and techniques we used in Section 4.4. Finally, we present the conclusion of this chapter in Section 4.5.

4.1 Approach Overview

In this thesis, we develop a multi-stage framework consisting of a set of models that use network flow-based features. To achieve this objective, we will follow a specific methodology consisting of several steps that can be restated as follows: i) Feature Extraction and Data Pre-Processing, ii) Optimized Feature Selection, and iii) Device identification. In what follows, we elaborate on each of these steps. An overview of our approach is illustrated in Figure 9.

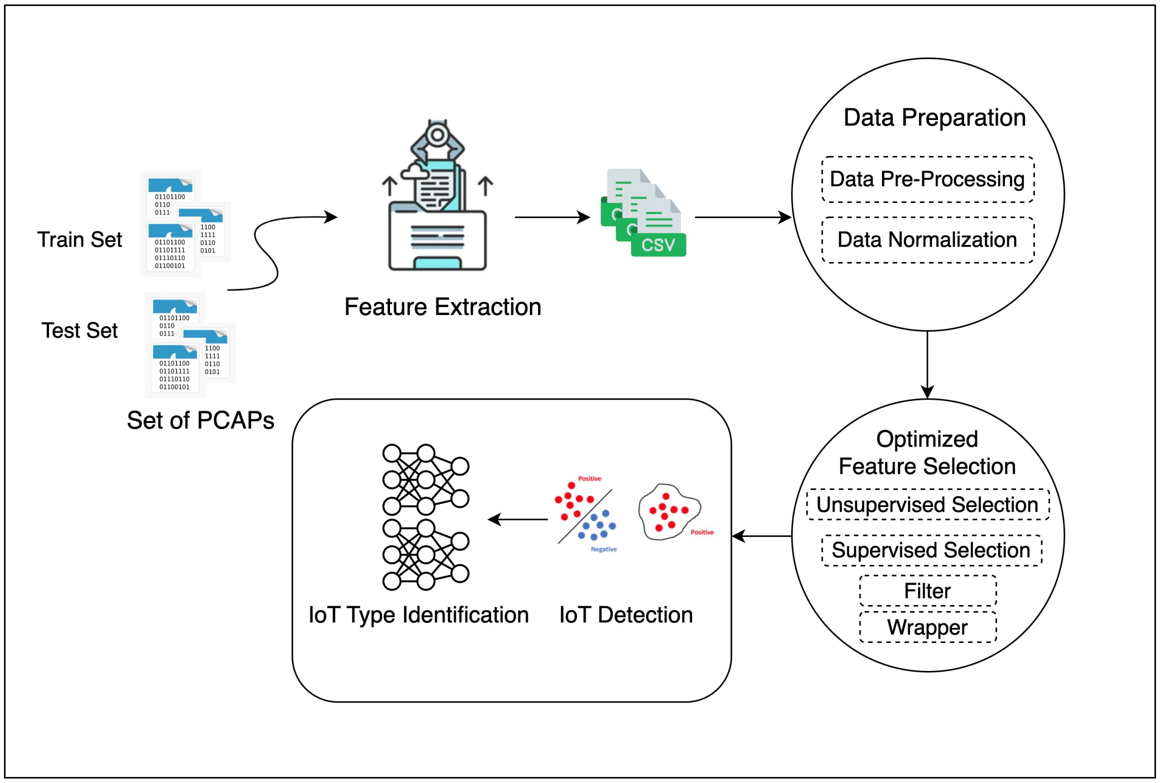


Figure 9: Approach Overview

4.2 Feature Extraction

The first step of identification involves extracting relevant information or attributes from network data. By extracting these features, a device fingerprint can be created, which serves as a digital signature or unique identifier for the device. Feature extraction involves transforming raw data into quantifiable features (structured data) that can be used for analysis and modeling. This step is crucial in building an efficient and accurate identification model. Network communications offer rich device-related information that can be utilized for device-type identification. In this regard, the features are extracted through raw passive observations from the network traffic of IoT devices during the communication setup phase. However, encryption is a challenge for traffic analysis. Since the payload content is encrypted, the characteristics are mainly reflected in the data flow. In this context, we work with bidirectional flow-based features, which include information about both incoming and outgoing traffic between devices since network traffic flows are generated from a sequence of aggregated packet header data. By considering bidirectional flows, we can gain insights into the complete communication between devices, capturing the interactions in both directions. This can reveal more comprehensive patterns within the network. Analyzing network flows provides a more efficient and manageable approach, as it allows for the aggregation of information Besides, network flows provide valuable insights into network traffic patterns even when the traffic is encrypted, as they are derived from packet headers without revealing the actual contents of the encrypted data. A network flow is identified or characterized by a sequence of packets that share the same five-tuple information, which includes the source IP address, destination IP address, source port, destination port, and protocol.

4.3 Optimized Feature Selection

The high-dimensional data contain irrelevant and redundant features that reduce the performance of the models. Hence, Feature selection is required. Feature or variable selection is the process of selecting a subset of significant features from a given set of features. It primarily focuses on removing irrelevant and redundant variables [87]. Feature selection has significant benefits in terms of storage requirements, reducing training time, reducing model complexity and defying overfitting, as irrelevant and redundant features may confuse the learning algorithm. Furthermore, the presence of non-informative variables can add uncertainty to the predictions and reduce the overall effectiveness of the model [87]. Thus, in this work, we select a subset of relevant features based on the model performance (i.e., removing irrelevant features without compromising the performance). There are a variety of methods to reduce the feature set dimensionality. Generally, the selection can be divided into two main categories: supervised methods and unsupervised methods. In supervised methods, the class/target is involved during the selection/elimination of variables in the former, while it is ignored in unsupervised methods [87]. Supervised feature selection can be further classified into three classes: filter, wrapper, and embedded (or intrinsic) methods [88].

Filter Methods

Filter-based feature selection uses a metric to find irrelevant features. Features are selected based on their relevance and relationship evaluation with the target variable (i.e. class in a classification problem) using statistical measures such as correlation and information theory. These methods are fast and simple; however, they are prone to over-selecting features as they evaluate each variable separately, which results in selecting all the highly ranked/scored features.

Wrapper Methods

Wrapper feature selection was introduced in [89]; unlike filter methods which are independent of any machine learning algorithm, wrapper methods rely on the performance of a given predictive model to assess the usefulness of features. This approach involves repeatedly training and testing the model on different subsets of the original feature set and selecting the subset with the least error or best performance. While wrapper feature selection has been shown to be effective in various applications, it can be time-consuming and computationally expensive.

Embedded Methods

Intrinsic or embedded feature selection methods are incorporated into some machine learning algorithms, such as tree-based models. These models perform automatic feature selection during the model fitting/training process. The main advantage of Embedded feature selection methods is that they do not require an external feature selection. In addition, they

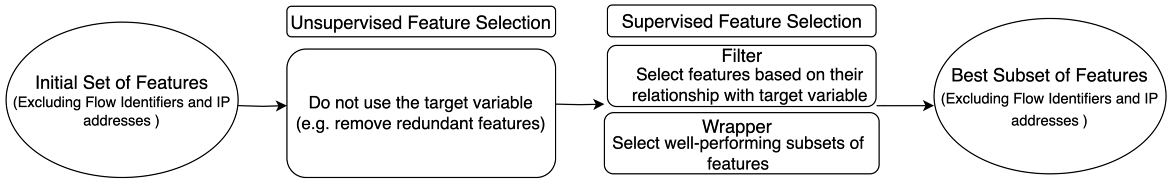

Initial Set of Features
(Excluding Flow Identifiers and IP addresses)
Unsupervised Feature Selection
Do not use the target variable
(e.g. remove redundant features)
Supervised Feature Selection
Filter
Select features based on their relationship with target variable
Wrapper
Select well-performing subsets of features
Best Subset of Features
(Excluding Flow Identifiers and IP addresses)

Figure 10: Hybrid Feature Selection Method Overview.

are faster than wrappers since they are embedded. However, the main downside is that they are usually specific to given learning algorithms (model-dependent) [88]. If data is better fit by another type of model that does not contain built-in feature selection, then predictive performance may be sub-optimal.

Given the advantages and issues related to each method, how should one select a feature selection method? This question has never been addressed because no general or best feature selection method exists. In this study, we will use a hybrid feature selection solution which combines different algorithms to rectify the shortcomings of each incorporated method. Figure 10 illustrates our hybrid feature selection overview. This proceeds in two stages:

1) Unsupervised Feature Selection to identify and remove non-informative (zero-variance and duplicated features) and redundant features. A feature is redundant if it is highly correlated with other feature(s). A correlated feature adds no relevant information, and thus it does not redound to getting a better feature.

2) Hybrid Supervised Features Selection to remove irrelevant features and maintain the top relevant ones. Feature relevance indicates that the feature is necessary for an optimal subset. Consequently, irrelevant features can never contribute to prediction performance. As previously mentioned, wrapper methods can be more accurate but computationally expensive and time-consuming. To tackle this problem, we first apply the filter method to

identify the highly correlated features with respect to the dependent/target variable. Subsequently, we use a wrapper method that consists of two main components:

Search Strategy: A search algorithm is used to add and/or eliminate features. Roughly speaking, this procedure can be forward (Sequential Forward Selection (SFS)) or backward (Sequential Backward Selection (SBS)). In forward selection, the process starts with an empty set and variables are progressively added to the feature subset. In contrast, backward selection starts with the set of all variables and progressively eliminates the least promising ones [88]. However, in machine learning, combining different features may result in different feature significance, where a useful feature may be non-useful combined with other(s), and vice versa. Therefore, SFS and SBS can be limited as we cannot remove/add a feature once it is incorporated/eliminated. To overcome this problem, we will use the floating search method that dynamically add and remove features based on their usefulness when combined with other(s). In this study, we will examine both forward and backward and select the one that results in better performance.

Modeling Method: As mentioned earlier, a wrapper method requires a machine learning model to evaluate its performance over all the possible features sub-sets. Artificial Neural Networks are powerful computational models which can be utilized for solving complex estimations. They are commonly used as classifiers [90] due to their robustness. One of the most popular ANN models is the multi-layer perception (MLP). MLP model is used in this step for feature evaluation. The decision of feature subset selection depends on model performance.

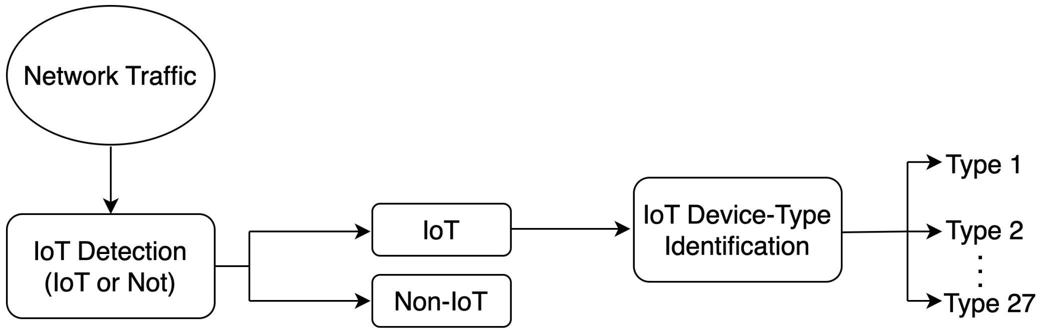


Figure 11: Architecture of IoT Device Inference Engines

4.4 IoT Device Identification

When a device connects to the network, our proposed method, as shown in Figure 11, consists of the IoT detection phase and IoT device-type identification phase. To meet these goals, the first operation is building a binary classifier to distinguish IoT from non-IoT devices. Once the presence of IoT devices is detected (e.g., the device is classified as an IoT), a few-shot learning based on the siamese Neural Network algorithm is used to identify the IoT device-type.

4.4.1 IoT Detection

In order to identify the type of IoT device connecting to a network, it is necessary first to detect the presence of IoT devices. IoT Detection refers to the process of identifying the presence of IoT devices within a network. This involves detecting the existence of IoT devices by analyzing network traffic, characteristics or patterns. IoT Detection focuses on discovering whether IoT devices are present or connected to a network, without necessarily identifying specific details about those devices, by distinguishing them from non-IoT devices. In this context, IoT devices are specific-purpose devices that interconnect with each other without direct human interaction, such as smart kettles and smart plugs, as IoT devices [54]. Conversely, non-IoT devices are general-purpose devices such as laptops, smartphones, computers and tablets that fall under the class of non-IoT. In this phase, we employ a binary classifier and train it on the IoT and non-Iot classes. We apply different binary classifiers, which will be detailed in Chapter 5. Another approach we can use for binary classification tasks is One-Class Classification (OCC) [91]. Unlike traditional binary classification, where the goal is to separate data points into two distinct classes, OCC aims at fitting a model on single-class data and predicting whether new data belongs to the same class or not [92]. In this case, we train a one-class classifier on the IoT devices data where the model captures the density of the IoT class and classifies examples on the extremes of the density function as non-IoT. One-Class Support Vector Machines (OC-SVM) and Deep One-Class Classification (DOCC) are common algorithms used for one-class classification [93]. The model OC-SVM works by mapping the data to a high-dimensional feature space and then identifying a hyperplane that separates the data from the origin of the feature space. The hyperplane is defined by a set of support vectors that are located at the boundary of the data. The distance between the hyperplane and the origin measures the similarity between a new data point and the training data. The model OC-SVM is powerful for detection problems but has some limitations. One of the main limitations is that OC-SVM is based on a linear separation boundary and may not be able to capture non-linear relationships in the data. On the other hand, DOCC is a variant of one-class classification that uses a deep neural network to learn a non-linear boundary that separates the positive examples from the negative examples (i.e., IoT from non-IoT). Deep One Class Classification involves training a DNN on a set of positive examples to learn a discriminative representation for the positive and invariant to the negative examples. During training, the model is only exposed to the positive examples (i.e., IoT data) and is not provided with any labels indicating which examples are negative. The trained model is then used to make predictions on new data, and a score is computed for each observation, which reflects how well the observation fits with the learned representation of the positive examples.

4.4.2 IoT Device-Type Identification

Machine learning has been successfully used to achieve state-of-the-art performance in various applications. However, in many real-world scenarios, it relies on large amounts of annotated examples to achieve high performance. a large storage space for storing data and computing resources, which may be costly or/and time-consuming resulting in significant training overhead. Few-Shot learning is an emerging subfield of machine learning that addresses the challenge of training models with limited labeled data. The term "few-shot" refers to the fact that the model is trained on a few training examples (i.e., a small number of training examples). By leveraging prior knowledge and generalization capabilities, Few-Shot learning algorithms enable models to quickly adapt and make accurate predictions on unseen data. Moreover, Few-Shot learning techniques are specifically designed to handle scenarios where new classes (i.e., device-types) emerge without necessitating a complete retraining process. This feature makes Few-Shot learning particularly valuable in scenarios where collecting large amounts of labeled data is expensive or time-consuming or where new classes are continually emerging. In few-shot learning, deep learning methods have been used to learn effective representations from a few examples and to perform similarity comparisons between examples. Siamese neural network is a deep learning architecture commonly used in few-shot learning.

Siamese Neural Networks (SNNs)

As the name suggests, siamese neural network is a network with two identical sub-networks architecture consisting of multiple layers of artificial neurons. These sub-networks share the same set of weights and parameters. In siamese neural network, instead of a model learning to classify its inputs using classification loss functions, the model takes two inputs through the two identical sub-networks and learns a similarity function that outputs a similarity score between them and checks whether they are the same. The output of the two sub-networks is then concatenated and fed through one or more fully connected layers, which output a similarity score. We employ few-shot learning based on siamese neural networks for IoT device-type identification. In this approach, the network receives data pairs as input to learn the similarity, and the output of the sub-networks are feature vectors (representation) for each example/input data. The distance between these vectors is measured at the similarity layer (using a distance metric such as Euclidian distance) to decide whether they belong to the same class. Figure 12 shows the general architecture of a siamese neural network. As we can see, Input data (i.e., features) are fed through the neural sub-networks, and each network transforms the input into a vector. The vectors are then fed into the distance/similarity function to decide whether they are close enough to be similar. The training process of siamese neural network involves minimizing the difference between the predicted similarity score and the actual similarity score between the two inputs in which examples from the same class are close together while examples from different classes are far apart. This is typically done using a loss function such as contrastive loss.

Model Definition

For this study, it was of interest to not restrict ourselves to a standard artificial neural network in order to investigate and compare the performance of different deep learning models when building an SNN model. We consider four different common deep learning models, including deep feed-forward neural network, convolutional neural network, recurrent neural network and transformer neural network. In the following, we describe the neural networks we consider in our model.

Deep Feed-Forward Neural Network:

MLP is a class of feed-forward neural networks. It consists of an input layer, one or more hidden layers, and an output layer. Each layer contains a number of perceptrons or neurons.

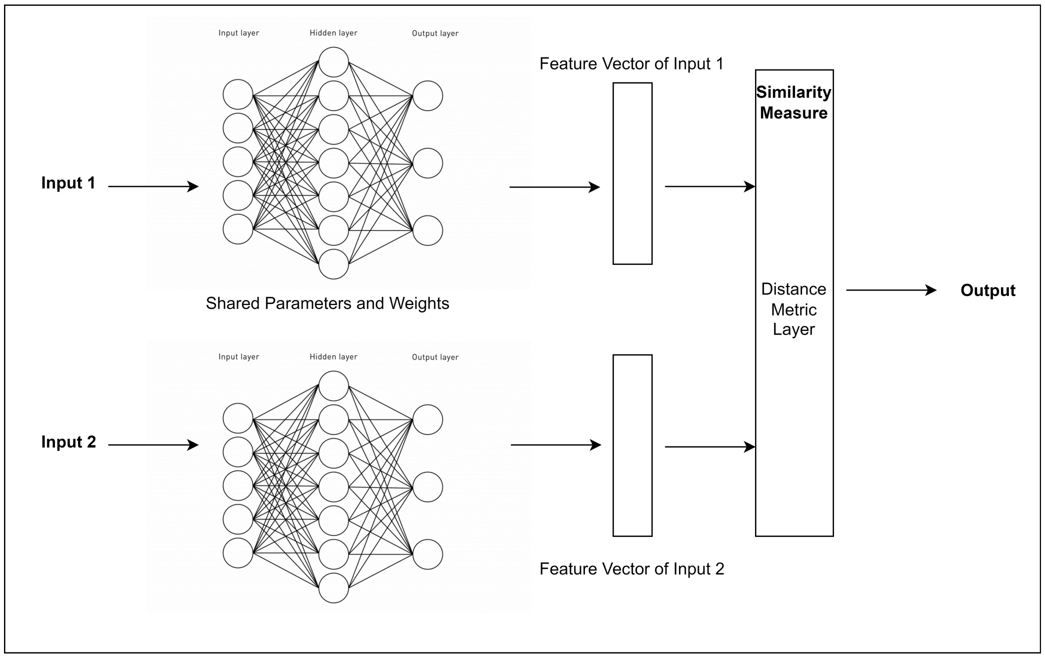


Figure 12: Siamese Neural Network Architecture [6].

The input layer receives the input data, which is passed forward to the hidden layers. Each neuron in the hidden layers receives input from the neurons in the previous layer, computes a weighted sum of the inputs, applies an activation function, and passes the output to the neurons in the next layer.

Convolutional neural network:

The convolutional layer is used to process one-dimensional sequential data, such as signals and sequences, two-dimensional image data and three-dimensional data, such as videos. The input to a 1D-CNN layer is a 1D sequence, represented as a tensor with shape (batch size, sequence length, input dim), where the batch size is the number of samples in a batch, sequence length is the length of the input sequence, and input dim is the number of input features at each time step. The first step of a 1D-CNN layer is to apply convolution to the input sequence. The convolution operation involves sliding a small filter (also called kernel) of fixed width along the sequence, computing the dot product between the filter and a segment of the input at each position, and producing a feature map that captures the presence of local patterns in the input. In this work, we employ a one-dimensional convolutional neural network (1D-CNN) as network traffic data is considered sequential. We convert the input to a 2-dimensional shape to be compatible with the 1D-CNN layer.

Recurrent neural network:

RNNs can capture sequential information from sequential data. Although there are various types of RNNs, the Long Short-Term Memory network is the most commonly used type of RNN. In this work, we use LSTM, which is capable of learning order dependence in sequential data. The input is converted to a 3-dimensional shape to be compatible with the requirements of the LSTM layer.

Transformer Neural Network:

As mentioned in Chapter 2, the transformer consists of an encoder and a decoder component. The encoder has two layers: multi-head attention and a fully connected feed-forward network. In addition to the two layers in the encoder, the decoder has a third layer that performs multi-head attention over the encoder output. In this work, we use the encoder part only and derive the architecture from the transformer encoder introduced in [5]. Specifically, we leverage the attention mechanism to capture information from the flows and 1D-CNN layer. We also add a normalization layer to avoid network degradation and faster training.

4.5 Conclusion

In this chapter, we have presented a description of the research methodology used in this study. We began by outlining the feature selection module, which involved a mixed-methods approach that combined different techniques to automatically identify the best features and eliminate irrelevant and redundant features. We also discussed our two-stage model, including IoT detection and IoT device-type identification. In summary, this chapter focused on the main key steps we followed in this study and presented a high-level idea of the machine learning models we leverage. The experimental setup, details of model architecture and parameters, and the findings of this study will be presented and discussed in the following chapter.

Chapter 5

Experimental Results and Analysis

In this chapter, we provide details of the experimental procedures and evaluate the effectiveness and present the results of our models, namely:

1. Optimized Feature Selection: We evaluate how effective is this process in reducing data dimensionality without affecting model performance.

2. IoT Detection: We evaluate how accurately this module can distinguish between IoT and non-IoT devices.

3. IoT Device-Type Identification: We compare and evaluate the performance of our models using different architectures.

5.1 Experimental Setup

The experiments of this study were conducted on a desktop computer equipped with Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz processor, and 16 GB of RAM. The operating system is Windows 11, and the code is implemented with Python using the scikit-learn [1] library for data preparation and preprocessing and Keras [2] library for deep learning models.

[1]https://scikit-learn.org/stable/

[2]https://keras.io/

5.2 Dataset

For this study, we use the data collected from IoT Sentinel [7] to evaluate the performance of our proposed approach. It is a set of PCAP traffic traces captured from a set of 31 IoT devices targeted for regular consumers [7]. This set represents 27 device types (4 types are represented by 2 devices each), covering the most common device types such as smart lighting, home automation, security cameras, household appliances and health monitoring devices [7]. A list of the devices used in this study is presented in Table 2. Some of these devices connect to the network via ZigBee or Z-wave protocols, but the majority use WiFi or Ethernet [7]. IoT Sentinel data collection is based on passive monitoring of the communication of devices during the setup process. The setup process was repeated 20 times for each device in order to capture sufficient data for machine learning model training. Table 2 lists the IoT devices used in this study along with their connectivity technologies. In this thesis, we extract network-flow-based features using the Canadian Institute for Cybersecurity feature extractor CICFlowMeter [3], which is a network traffic flow generator used to extract bidirectional flows identified by the 5-tuple (source IP address, destination IP address, source port, destination port, and protocol). The tool, CICFlowMeter, reads PCAP files as input and generates more than 80 network traffic flow-based features from the packet header, excluding the packet payload, ensuring that the features can be extracted from encrypted network traffic and outputs related CSV files.

[3]https://www.unb.ca/cic/research/applications.html

Table 2: IoT Devices Used in this Study and Their Connectivity Technologies [7].

Identifier	Device Model	Technology
Aria	Fitbit Aria WiFi-enabled scale	WiFi
HomeMaticPlug	Homematic pluggable switch HMIP-PS	Other
Withings	Withings Wireless Scale WS-30	WiFi
MAXGateway	MAX! Cube LAN Gateway for MAX!	Ethernet
HueBridge	Philips Hue Bridge model 3241312018	ZigBee
HueSwitch	Philips Hue Light Switch PTM 215Z	ZigBee
EdnetGateway	Ednet.living Starter kit power Gateway	WiFi
EdnetCam	Ednet Wireless indoor IP camera Cube	WiFi, Ethernet
EdimaxCam	Edimax IC-3115W Smart HD WiFi Network Camera	WiFi, Ethernet
Lightify	Osram Lightify Gateway	WiFi, ZigBee
WeMoInsightSwitch	WeMo Insight Switch model F7C029de	WiFi
WeMoLink	WeMo Link Lighting Bridge model F7C031vf	WiFi, ZigBee
WeMoSwitch	WeMo Switch model F7C027de	WiFi
D-LinkHomeHub	D-Link Connected Home Hub DCH-G020	WiFi, ZigBee
D-LinkDoorSensor	D-Link Door and Window sensor	Z-wave
D-LinkDayCam	D-Link WiFi Day Camera DCS-930L	WiFi, Ethernet
D-LinkCam	D-Link HD IP Camera DCH-935L	WiFi
D-LinkSwitch	D-Link Smart plug DSP-W215	WiFi
D-LinkWaterSensor	D-Link Water sensor DCH-S160	WiFi
D-LinkSiren	D-Link Siren DCH-S220	WiFI
D-LinkSensor	D-Link WiFi Motion sensor DCH-S150	WiFi
TP-LinkPlugHS110	TP-Link WiFi Smart plug HS110	WiFi
TP-LinkPlugHS110	TP-Link WiFi Smart plug HS110	WiFi
TP-LinkPlugHS100	TP-Link WiFi Smart plug HS100	WiFi
EdimaxPlug1101W	Edimax SP-1101 W Smart Plug Switch	WiFi
EdimaxPlug2101W	Edimax SP-2101 W Smart Plug Switch	WiFi
SmarterCoffee	Smarter SmarterCoffee coffee machine SMC10-EU	WiFi
iKettle2	Smarter Kettle 2.0 water kettle SMK20-EU	WiFi

Table 3 shows the initial list of the extracted features with their descriptions. We obtained a dataset of 540 fingerprints representing 27 device types (20 fingerprints x 27 device-type).

Table 3: The Initial Set of Features Extracted by CICFlowMeter.

Feature	Description
Flow duration	Duration of the flow in Microsecond
Total Fwd Packet	Total packets in the forward direction
Total Bwd packets	Total packets in the backward direction
Total Length of Fwd Packet	Total size of packet in the forward direction
Total Length of Bwd Packet	Total size of packet in the backward direction
Fwd Packet Length Min	Minimum size of packet in forward direction
Fwd Packet Length Max	Maximum size of packet in forward direction
Fwd Packet Length Mean	Mean size of packet in forward direction
Fwd Packet Length Std	Standard deviation size of packet in forward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Bwd Packet Length Max	Maximum size of packet in backward direction
Bwd Packet Length Mean	Mean size of packet in backward direction
Bwd Packet Length Std	Standard deviation size of packet in backward direction
Flow Bytes/s	Number of flow bytes per second
Flow Packets/s	Number of flow packets per second
Flow IAT Mean	Mean time between two packets sent in the flow

Continued on next page

Table 3 – continued from previous page

Feature	Description
Flow IAT Std	Standard deviation time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Total	Total time between two packets sent in the forward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction

Continued on next page

Table 3 – continued from previous page

Feature	Description
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Fwd PSH flags	Number of times the PSH flag was set in packets travelling in the forward direction
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the backward direction
FWD Packets/s	Number of forward packets per second
Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a packet
Packet Length Max	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet

Continued on next page

Table 3 – continued from previous page

Feature	Description
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWR Flag Count	Number of packets with CWR
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Average Packet Size	Average size of packet
Fwd Segment Size Avg	Average size observed in the forward direction
Bwd Segment Size Avg	Average size observed in backward direction
Fwd Bytes/Bulk Avg	Average number of bytes bulk rate in the forward direction
Fwd Packet/Bulk Avg	Average number of packets bulk rate in the forward direction
Fwd Bulk Rate Avg	Average number of bulk rate in forward direction
Bwd Bytes/Bulk Avg	Average number of bytes bulk rate in the backward direction
Bwd Packet/Bulk Avg	Average number of packets bulk rate in the backward direction

Continued on next page

Table 3 – continued from previous page

Feature	Description
Bwd Bulk Rate Avg	Average number of bulk rate in the backward direction
Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
Fwd Init Win bytes	The total number of bytes sent in initial window in the forward direction
Bwd Init Win bytes	The total number of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	Count of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Min	Minimum time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle

Continued on next page

Table 3 – continued from previous page

Feature	Description
Active Max	Maximum time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Min	Minimum time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active

5.3 Data Preparation

Data preparation helps improve the data quality and ensures that the machine learning model produces accurate and reliable results.

5.3.1 Data Labeling, Splitting, and Preprocessing

Data labeling involves assigning labels or categories to the data instances, typically for supervised learning tasks where the goal is to train a model to predict an output variable based on input variables. To this aim, we group all IoT types/classes into one class and label it as "IoT" for the IoT detection module. For IoT device-type identification model, the device types are assigned as labels for each IoT type, such as Aria, HueBridge, etc. After labeling, data splitting is an essential step that involves dividing a dataset into two or more subsets to be used for the training, validation, and testing of a machine learning model. The training set is used to train the model, the validation set is used to evaluate the model's performance during training and to optimize the model's hyper-parameters, and the test set is used to evaluate the final performance of the model on new, unseen data. In this work, we split the data before applying any preprocessing step to avoid data leakage that can occur when preprocessing the entire data before splitting. This ensures that the model is only trained on the training data and does not have access to any information from the test set, thus, avoiding overfitting and unreliable performance. As mentioned, we have 20 PCAP files for each class or device type. To this end, we use 15 PCAP files for training and 5 PCAP files for Evaluation and Testing. On the other hand, data preprocessing prepares raw data for machine learning algorithms by transforming and cleaning it. It is a critical step in building a successful machine learning model, as the quality of the data fed into the model directly affects its performance. Our data preprocessing involves several steps (for both modules, including IoT detection and IoT device-type identification), including:

- Data cleaning: This involves removing or correcting any errors in the data, such as missing values (NaN values) or incorrect data types.

- Data transformation: This involves converting the data into a format that is more suitable for machine learning algorithms, such as scaling or normalizing the data. We employ Min-Max [4] scaling technique to normalize data so that it is scaled to a fixed range. In this technique, each feature (or variable) in the dataset is transformed so that its minimum value is mapped to 0 and its maximum value is mapped to 1. The formula for Min-Max scaling is:

[4]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

$$\text{xscaled} = \frac{(x - \min(x))}{(\max(x) - \min(x))} \quad (1)$$

where x is the original value of a feature, xscaled is the scaled value, min(x) is the minimum value of the feature, and max(x) is the maximum value of the feature.

- Data encoding: This involves the process of converting categorical data into numerical values so that it can be used in machine learning algorithms. However, both sets of features we use in this thesis are numerical. Therefore, we employ this step to encode the target/label variable solely as the target variable is categorical.

5.4 Feature Selection

We now identify traffic features of IoT devices that can be selected from the original feature sets to detect and distinguish them from non-IoT devices and classify their device type. First, since our approach is not intended to rely on features such as MAC/IP addresses, we initially removed the flow identifiers such as Flow ID, Source IP, Destination IP, Source Port and TimeStamp features. Then, to provide insight into feature selection advantages and assess its impact, we trained a full model, which refers to a model that uses all remaining available features to make predictions. In this thesis, we employ the MLP classifier with the default parameters. Then, to determine how beneficial or detrimental the removal of certain features on the model performance would be, the same model was trained and evaluated each time a feature selection method was applied. In this section, we detail the implementation of these methods (Filter, Wrapper, and Embedded as elaborated in Section 4) using Scikit-learn [5], Feature-engine [6] and Mlxtend [7] libraries.

[5]https://scikit-learn.org/stable/
[6]https://pypi.org/project/feature-engine/
[7]http://rasbt.github.io/mlxtend/

5.4.1 Unsupervised Feature Selection

We employed four unsupervised feature selection steps to find and eliminate any zero-variance, quasi-constant, duplicated and correlated features. We use correlation measures to identify and remove correlated features in the context of correlated features. A strong correlation allows us to predict one variable from another. As such, the correlated features will not add additional information. The brute force method is one way to search for correlated features based on a correlation coefficient. However, the problem with such an implementation, aside from its limited performance, is that it may lead to the unintended removal of an important feature.

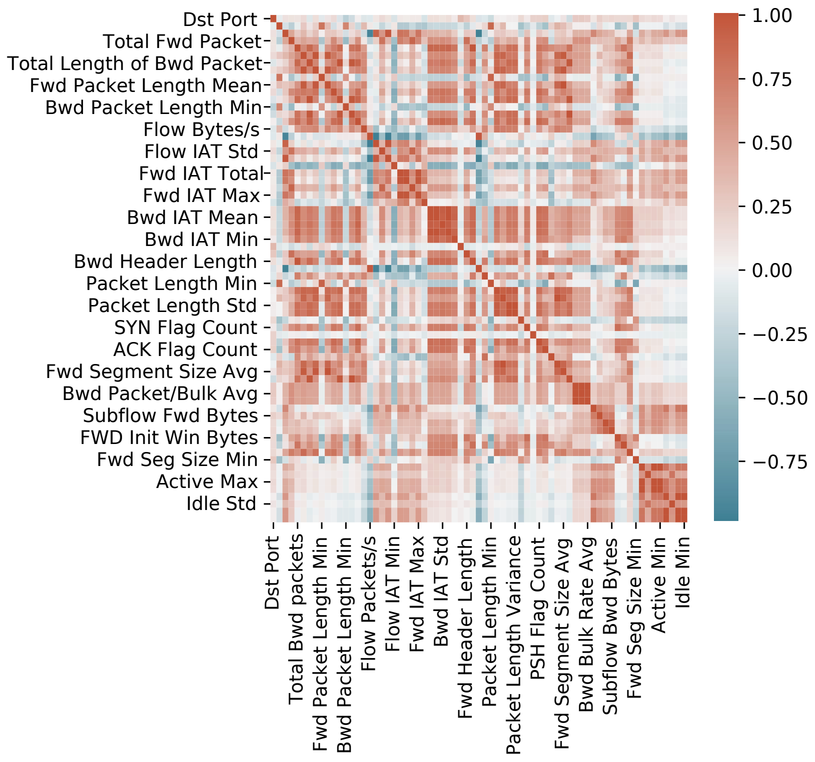


Figure 13: Correlation Matrix Depicting the Correlation among Features for IoT Detection

An alternative approach is to find groups of correlated features and then select, from each group, a feature following certain criteria. This is also termed as smart correlation selection. In this experiment, to compare correlation selection methods, we ran them both.

For the smart correlation method, we selected from each group the feature with the highest variance.

We used the Spearman coefficient, which measures the relationship between two variables monotonically related, even if their relationship is not linear [94], as it provided the best performance in finding correlated features. It selects the minimum set of features with the highest test accuracy compared to the Kendall and Pearson coefficients. Moreover, the Spearman's rank correlation test does not carry any assumptions about the data distribution and is suitable for continuous and discrete ordinal variables. Figures 13 and 14 show the correlated features using the Spearman coefficient. The matrices depict the correlation between each pair of features for the IoT detection model and IoT device-type model, respectively.

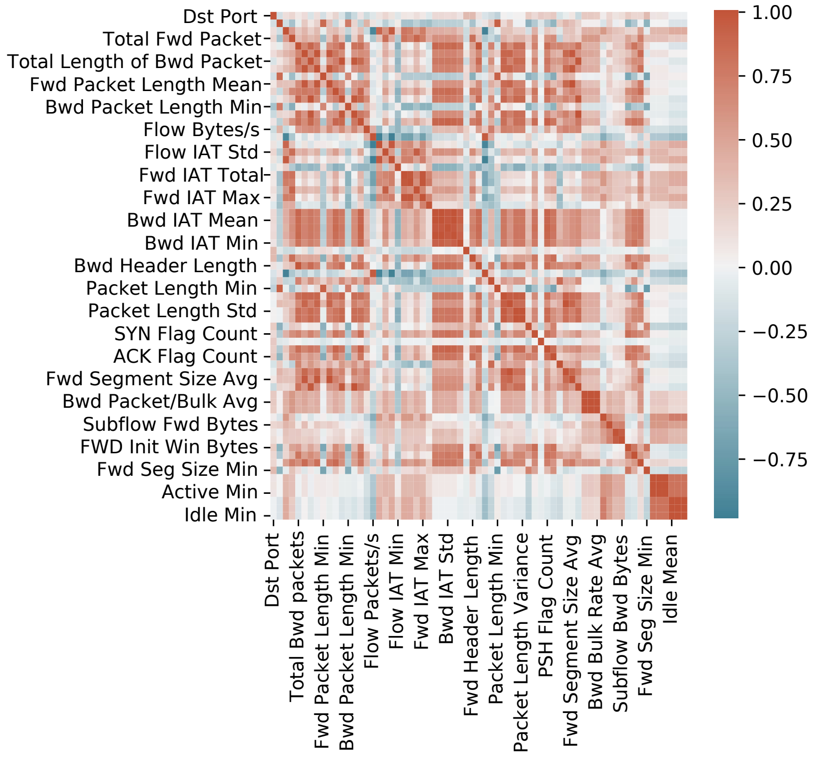


Figure 14: Correlation Matrix Depicting the Correlation among Features for IoT Device-Type

The coefficient ranges from -1 to +1, where -1 represents a negative correlation, 0 represents no correlation, and +1 represents a positive correlation. The correlation coefficient is represented using different colours, with a darker colour indicating a stronger correlation and a lighter colour indicating a weaker correlation. A feature that is highly correlated with other feature(s) (magnitude of 0.90) will be removed in this step. Figure 15 shows an example of a positive correlation between Fwd Packet Length Mean and Fwd Segment Size Avg with +1 correlation coefficient, indicating that as Fwd Packet Length Mean increases, so does Fwd Segment Size Avg.

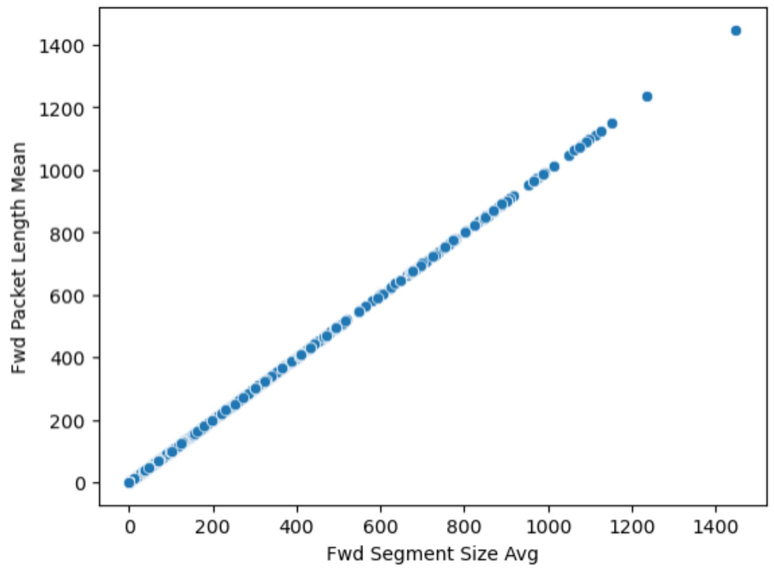


Figure 15: The correlation between Fwd Packet Length Mean and Fwd Segment Size Avg

.

5.4.2 Supervised Feature Selection

We further explored supervised feature selection, which typically consists of two stages: In the first stage, we employ the filter method to reduce the feature space by removing the irrelevant features. We use the wrapper method in the second stage to find the optimal subset from the retrained features.

Supervised Filter Method

It is common to use a statistical measure between independent (input) and dependent (output) variables. As such, the statistical measure/test needs to consider the data types. We chose to use mutual information (information gain), which is powerful and data-type agnostic. Mutual information measures the amount of information we can obtain from one variable given another [95]. It is one of the widely used measures in the context of feature selection based on filter methods, allowing us to capture the relevancy of features in predicting the target variable and the redundancy with other variables.We use SelectKBest [8] method from the scikit-learn library which identifies the features that are most relevant to the target variable and eliminates the ones that are less important or redundant using mutual information. The parameter "K" refers to the number of top features that we want to select. Figures [16,17] represent a visual representation of the scores of features in relation to the target variable, indicating which features are most informative for predicting the target variable for IoT Detection and IoT Device-Type Identification respectively. In order to set the best value of K, we use grid search (i.e. GridSearchCV[9]), a systematic approach that involves defining a range of possible values (numbers of selected features) and testing each combination using cross-validation.

Supervised Wrapper Method

Recall from Section 4.3 that wrapper feature selection is a greedy search algorithm that starts with an empty set of features and relatively adds one feature at a time until a desired number of features is reached.

[8]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
[9]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

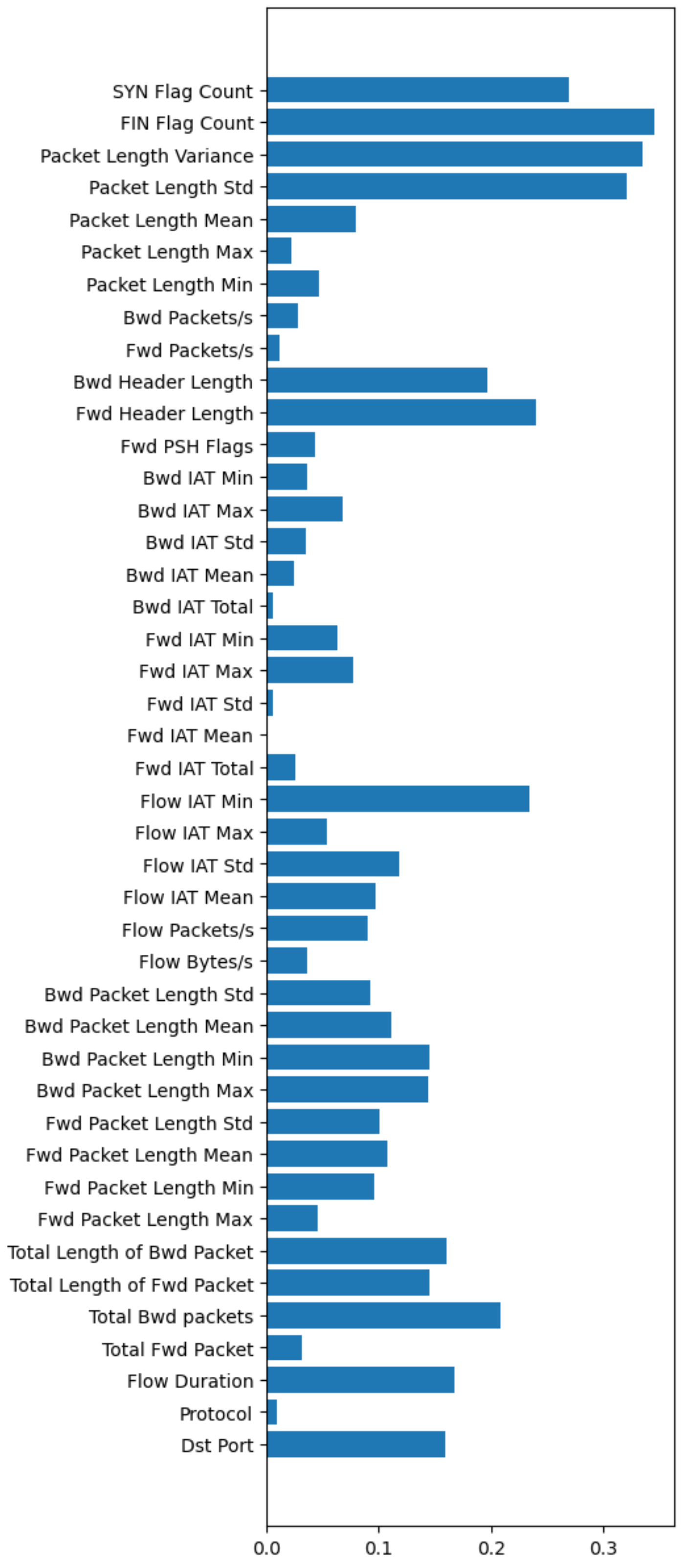


Figure 16: Importance of Each Feature with Respect to the Target Variable (IoT or non-IoT) for IoT Detection.

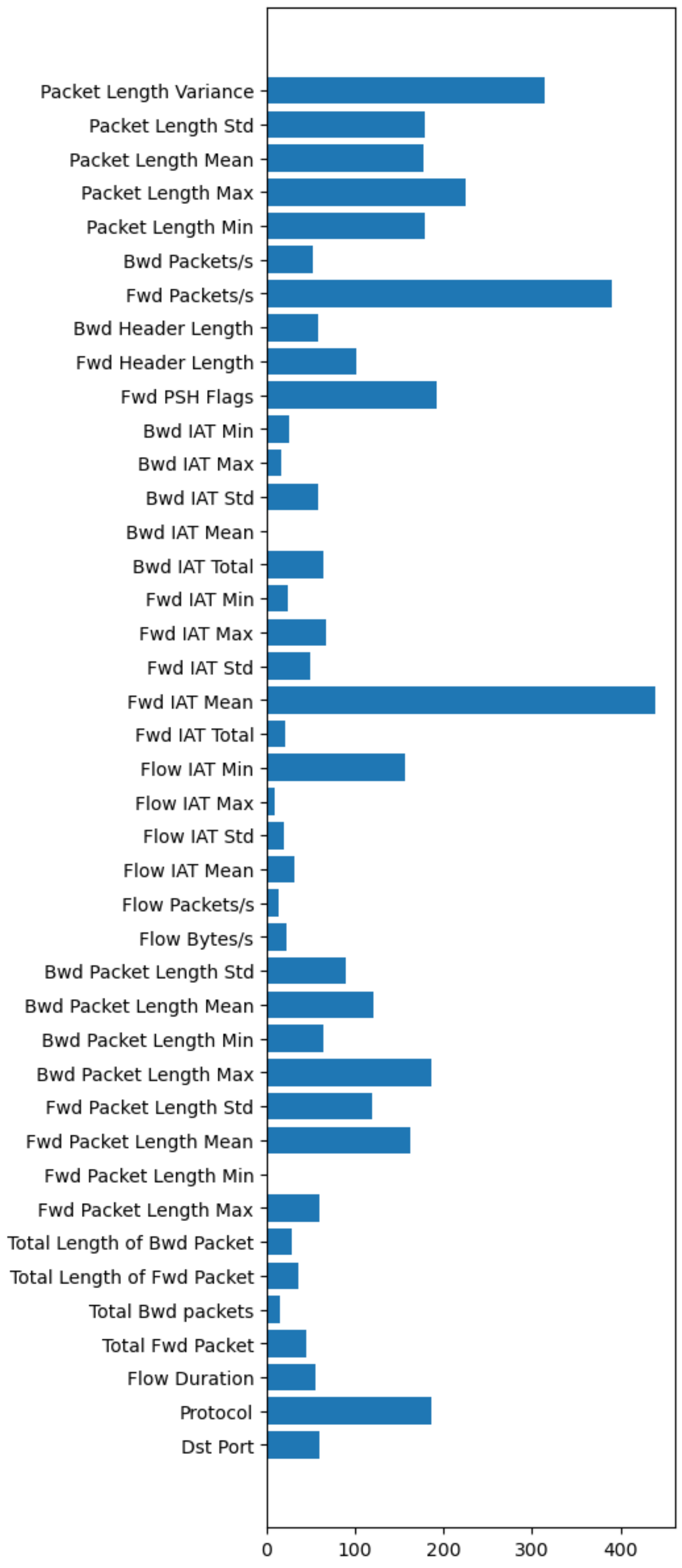


Figure 17: Importance of Each Feature with Respect to the Target Variable (Device-Type) for IoT Device-Type Identification.

Supervised Wrapper Method

Recall from Section 4.3 that wrapper feature selection is a greedy search algorithm that starts with an empty set of features and relatively adds one feature at a time until a desired number of features is reached. In this stage, we use the SequentialFeatureSelector algorithm from the MLxtend library, which selects a subset of features from retained features from the previous stage. The algorithm works by evaluating the performance of the MLP model on different subsets of features using an accuracy metric. The SFFS algorithm is as follows:

1. Initialize an empty set of selected features, $S_k =$ □; $k = 0$, where S is the set of features and k is the number of features.

2. Select the most significant feature x^+ that is not already in the selected features set, train a model using the selected features plus the new feature and measure its performance using the accuracy metric.

3. Update $S_k = S_k + x^+$; $k = k + 1$.

4. Select the least significant feature x^- from previously selected features and evaluate whether the removal of x^- improves the performance of the model.

5. If $M(S_k - x^-) > M(S_k)$ then remove x^- ; $S_k = S_k - x^-$; $k = k - 1$ and repeat step 4; otherwise, repeat step 2.

6. Stop when k = the number of features required.

In order to set k, we define a range of k values to test and use grid search (i.e., GridSearchCV) to search over the parameter grid defined by these k values.

5.4.3 Findings

Here, we report the findings of the optimized feature selection module evaluation for different problems, including IoT detection and IoT device-type identification.

IoT Detection

We begin with traffic features that distinguish IoT devices from non-IoTs. Figure 18 shows the accuracy of a full model that uses all available features (excluding flow identifiers and IP addresses) compared to a model that uses only a subset of selected features. The figure shows the performance after the application of each feature selection method. The results could show that the model using the selected subset of features performs similarly or even better than the full model, which could indicate that some of the features in the full model are redundant and irrelevant, leading to overhead. Alternatively, Tables [4, 5] present the final feature subset and indicate which features are the most relevant and informative for IoT detection and IoT Device-Type identification models, respectively. This information could be useful for feature engineering or data preprocessing in future iterations of the model-building process.

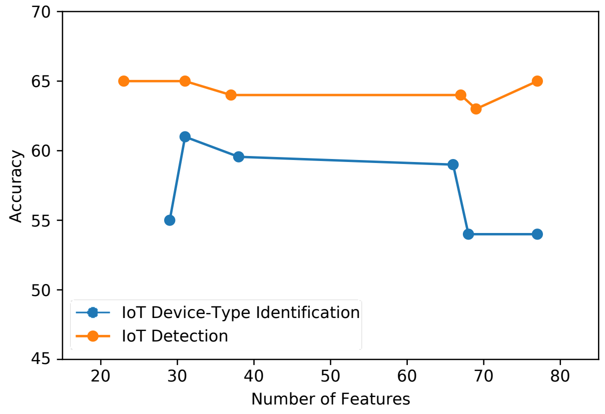


Figure 18: Plot of Model Accuracy versus Number of Features

IoT Device-Type Identification

Focusing on IoT types, we examined our hybrid feature selection using CIC Features. The results can be seen in Figure 18, which shows a moderate degradation in performance when a full model is trained using all the features (the accuracy is around 55%), compared to better performance (accuracy is improved to 62%) when using a selected subset of 31 features. As expected, feature selection allowed us to find a smaller subset of features without negatively affecting the model performance. Table 5 lists the final set of selected features for this model. With the data now preprocessed and ready for use, the next step is to train our machine learning models.

Table 4: Final Subset of Selected Features for IoT Detection.

Module	Subset of Features
IoT Detection	['Dst Port', 'Flow Duration', 'Fwd IAT Min', 'Fwd PSH Flags', 'Fwd Header Length', 'Packet Length Max', 'FIN Flag Count', 'FWD Init Win Bytes', 'Fwd Seg Size Min', 'SYN Flag Count', 'Flow IAT Mean', 'Flow IAT Std', 'Fwd IAT Std', 'Fwd IAT Mean', 'Fwd Packet Length Max', 'Bwd Packets/s', 'Fwd Packets/s', 'Bwd Packet Length Min', 'Fwd Packet Length Std', 'Fwd Packet']

Table 5: Final Subset of Selected Features for IoT Device-Type Identification.

Module	Subset of Features
IoT Device-Type Identification	['Dst Port', 'Protocol', 'Flow Duration', 'Total Fwd Packet', 'Total Bwd packets', 'Fwd Packet Length Min', 'Fwd Packet Length Std', 'Bwd Packet Length Max', 'Bwd Packet Length Min', 'Bwd Packet Length Std', 'Flow IAT Mean', 'Flow IAT Std', 'Flow IAT Min', 'Fwd IAT Total', 'Fwd IAT Min', 'Bwd IAT Total', 'Fwd Header Length', 'Packet Length Max', 'FIN Flag Count', 'SYN Flag Count', 'RST Flag Count', 'Down/Up Ratio', 'Bwd Packet/Bulk Avg', 'Subflow Fwd Packets', 'Subflow Bwd Bytes', 'FWD Init Win Bytes', 'Bwd Init Win Bytes', 'Fwd Act Data Pkts', 'Fwd Seg Size Min', 'Active Max', 'Idle Max']

5.5 Models Training and Performance Evaluation

5.5.1 Evaluation Metrics

We evaluate the performance of the different components of our framework in terms of accuracy, F1 score, recall and precision. We also use AUC (Area under the ROC Curve) metric as it measures the model's ability to distinguish between classes (i.e., determines the ability to avoid wrong classification) and indicates the trade-off between the recall and the number of incorrectly classified samples. These metrics are defined as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

$$F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

$$AUC = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{5}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

Where TP, FP, TN and FN indicate the rate of true positive rate, false positive rate, true negative rate and false negative rate, respectively.

5.5.2 IoT Detection

In IoT Sentinel data [7], there exists only traffic generated from IoT devices. In order to collect data from non-IoT traffic, we use PCAP files of non-IoT devices from UNSW data [68]. The final data encompasses over 39000 IoT (labeled 1) and non-IoT (labeled 0) instances as shown in Table 6.

Table 6: IoT and Non-IoT Dataset Overview.

	# Instances	Percentage
IoT	26964	68.54
Non-IoT	12377	31.46

In this phase, In order to test, evaluate and identify the best-performing machine learning model, we used AutoSklearn [96], an open-source library, built on top of the machine learning library, Scikit-learn, for automated machine learning (AutoML) that automates the process of model selection, hyper-parameter tuning. Random Forest showed the best performance, which achieved an accuracy of 83.23%. In addition, the performance metrics, namely recall, precision, F1 and AUC are presented in Figure 19.

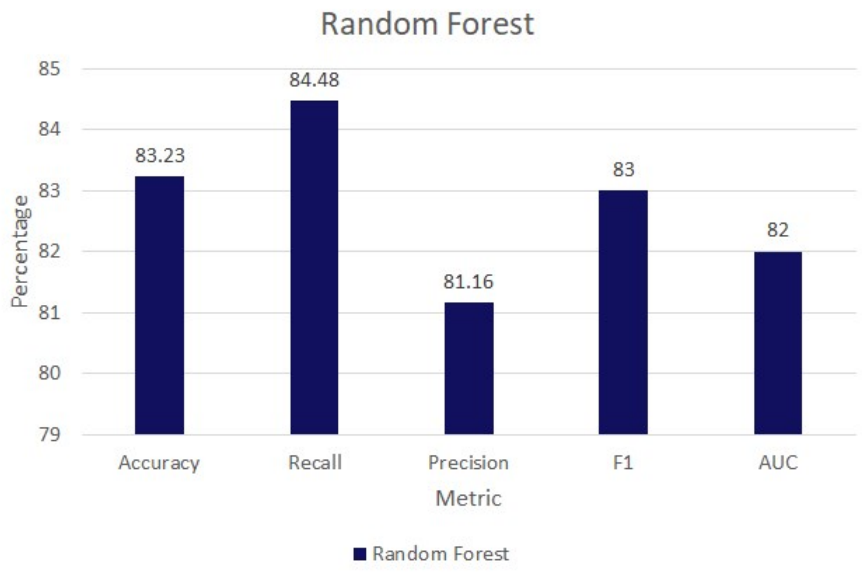


Figure 19: IoT Detector Results.

Figure 20 shows the confusion matrix of the IoT detection model. The columns indicate the true labels, and the rows show predicted labels (i.e., 1 for IoT and 0 for non-IoT).

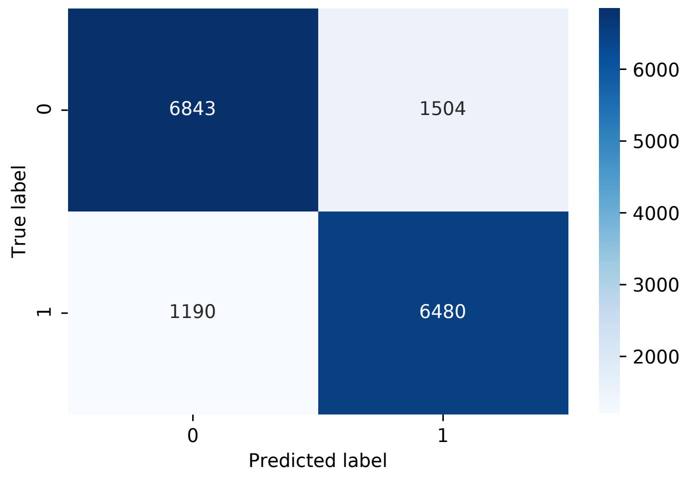


Figure 20: IoT Detector Confusion Matrix.

Handling Imbalanced dataset

Imbalanced data is a common and critical problem in machine learning, where the distribution of the target classes is highly skewed, with one or a few classes having significantly fewer samples than the others. This can result in models that perform poorly, leading to inaccurate predictions and decisions. Effective strategies for addressing imbalanced data include various sampling techniques such as under-sampling, over-sampling and synthetic data generation. Under-sampling involves randomly removing samples from the majority class until the dataset is balanced. This method is simple and computationally efficient, but it can lead to information loss and a reduction in overall model performance. Over-sampling involves randomly duplicating samples from the minority class until the dataset is balanced. This method can be effective in increasing the representation of the minority class, but it can also lead to overfitting and poor generalization performance. Synthetic data generation, on the other hand, is a technique used to balance the class distribution by generating synthetic instances for the minority class. However, the generated data may not accurately represent the true distribution of the original data, which can lead to incorrect results. In this work, we used ClusterCentroid algorithm based on K-means to under-sample the majority class (i.e., IoT Class) in order to balance the dataset. The implementation is conducted using Imblearn [10] library. This approach aims to identify the representative samples in the majority class by clustering the data and selecting the centroids as representatives. By reducing the number of majority class samples while preserving the diversity, Table 7 shows the data instances after conducting re-sampling. The proposed technique can overcome the limitations of previous sampling methods and improve the performance of machine learning models on imbalanced datasets, as shown in Figures22. After balancing the data, gradient boosting was the top model using AutoSklearn and achieved better performance than random forest. As shown in 23, the results showed that gradient boosting slightly outperformed random forest regarding the accuracy, recall, precision and F1 score. The accuracy of gradient boosting was 0.85, while that of the random forest was 0.84. Similarly, the recall of gradient boosting was 95.95%, while that of the random forest was 93.68%. The F1 score of gradient boosting was 95%, while that of the random forest was 93.56%.

On the other hand, the results also showed that one-class SVM performed poorly on this dataset. The accuracy of one-class SVM was 59.78%, while the F1 score was 69.94%. This indicates that one-class SVM was unable to identify the pattern in the dataset and was not an effective method for this problem.

Figure 21 shows the confusion matrix of the IoT detection model after addressing the imbalanced data problem.

[10]https://imbalanced-learn.org/stable/

Table 7: IoT and Non-IoT Dataset Overview after Applying ClusterCentriod Algorithm for Re-sampling.

	# Instances	Percentage
IoT	12000	49.23
Non-IoT	12377	50.77

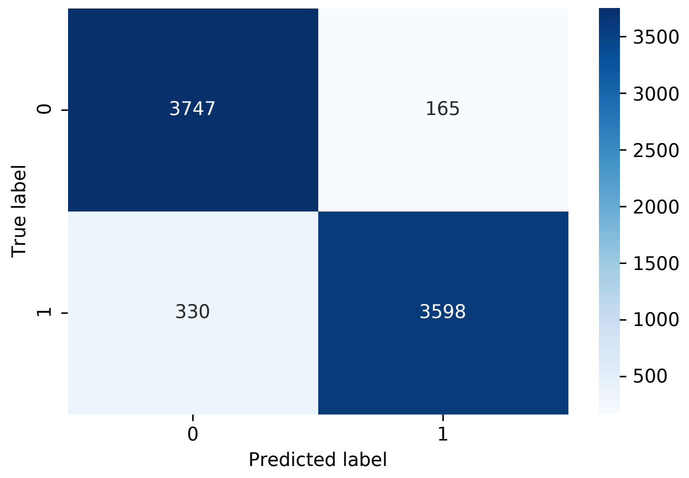


Figure 21: IoT Detector Confusion Matrix After ClusterCentriods Re-Sampling.

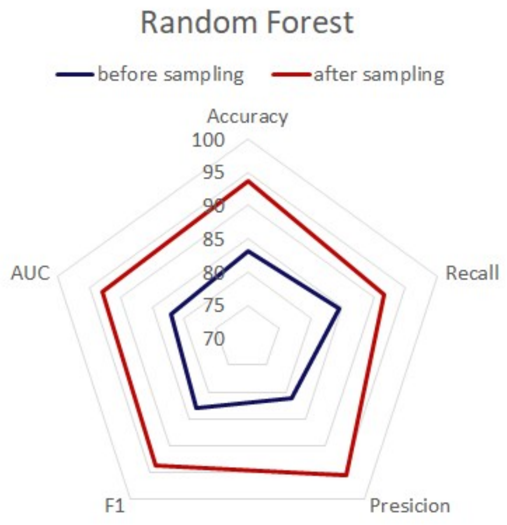


Figure 22: Random Forest Performance comparison after Applying ClusterCentriod Re-Sampling.

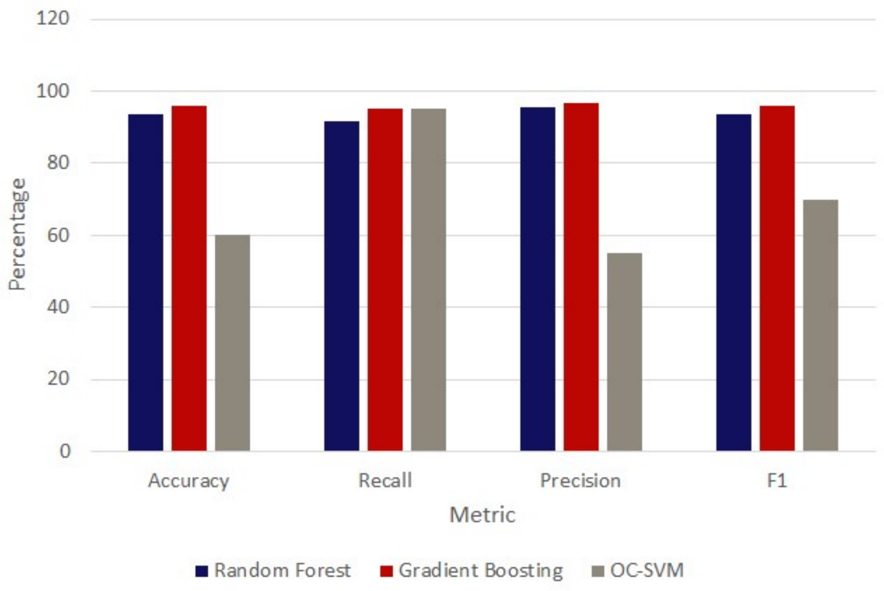


Figure 23: The Performance of Binary Classifiers including RF and GB, Compared with One-Class Classifier SVM.

5.5.3 IoT Device-Type Identification

Recall from the previous Section 4.4.2 that, unlike traditional classification models, the Siamese network receives a pair of flows as an input for training. Therefore, the first step in the process of creating and training this model is to create labeled pairs from labeled data. Before creating data pairs, we split the data into training and testing sets to avoid data contamination and data leakage. The input to this task consists of labeled samples (i.e., fingerprints) belonging to each IoT device-type. Then, given a set of data samples (i.e., fingerprints) for each class/device-type, we create two pairs: a positive pair of two fingerprints that belong to the same class, which is labeled as 1 (the data sample is paired with a random data sample of the same class) and a negative pair of two fingerprints from different classes, which is labeled as 0 (the data sample is paired with a random sample of a different class). Our approach relies on aggregated network flows, as generating device-type fingerprints using a single flow may result in similar behaviour between devices. Thus, network flow aggregation is significant in improving the performance of our system. Our analysis resulted in selecting 12 flows, which is chosen as the best value for all the subsequent experimentation. The output of this process is 858 pairs for each training and testing set. Next, the process involves designing the architecture of the Siamese sub-networks and selecting hyper-parameters for each of the four deep learning models we described in Section 4.4.2, including DFFNN, CNN, RNN and TNN. The architecture of a deep learning model refers to the overall structure of the neural network, such as the number of layers, types of activation functions, and connections between layers. Hyper-parameter selection, on the other hand, refers to the process of choosing the values of the hyper-parameters, which are settings that are not learned during training but are set by the user before training begins. Examples of hyper-parameters include learning rate, batch size, and dropout rates. The selection of hyper-parameters can significantly affect the performance of the model, and choosing appropriate values is critical to achieving high accuracy and preventing overfitting. As mentioned earlier, the sub-networks of the siamese network output two feature vectors for a given data pair, then we use a loss function to compute the similarity between the produced vectors based on a distance measure. We tested Manhattan and Euclidean distances to explore the effect of different distance measures on feature vectors. Furthermore, to determine the optimal values for the hyper-parameters, we explore the impact of various hyper-parameter values on the performance of our models. Specifically, we investigate the effects of varying the activation function, optimizers, learning rate, loss function, dropout, distance metric, batch size, and the number of epochs on model performance. We conducted a comprehensive experimental study, where we trained several deep learning models using different combinations of hyper-parameter values and evaluated their performance. In table 8, we list the hyperparameters that we considered in our experimental study, along with the specific values that we tested for each hyperparameter. For example, we experimented with learning rates ranging from 0.1 to 0.00001, batch sizes ranging from 16 to 128, and dropout rates ranging from 0.1 to 0.6. For each combination of hyper-parameters, we trained the model for 50, 100 and 150 epochs and evaluated its performance on the validation set. To simplify, we report only the best hyper-parameter values found during our experimental study in Table 9, where we summarize the optimal values for each hyper-parameter that resulted in the best performance for each model, including DFFNN, CNN, RNN, and TNN.

Table 8: Considered values for each hyper-parameter in the experimental study.

Parameters	Values
Activation Function	Relu and Tanh
Optimization	Adam, RMSProp, SDG and Nadam
Learning Rate	{0.1, 0.01, 0.001, 0.0001, 0.00001}
Loss Function	Cross-entropy loss and Contrastive loss
Dropout	{0.10, 0.20, 0.30, 0.40, 0.50, 0.60}
Distance Metric	Euclidean distance and Manhattan distance

Table 9: Best Hyper-parameters for each Deep Learning Model.

Parameters	DFFNN	CNN	RNN	TNN
Layers Type	Dense	Conv1D	LSTM	Attention
Activation Function	Relu	Relu and Tanh	Relu	Tanh
Optimization	Nadam	Nadam	Nadam	Nadam
Learning Rate	0.001	0.001	0.0001	0.001
Loss Function	Contrastive	Contrastive	Contrastive	Contrastive
Dropout	0.20	0.20	0.20	0.20
Distance Metric	Euclidean	Euclidean	Manhattan	Manhattan

For example, SNN models based on CNN and DFFNN perform better when using Euclidean distance, while SNN models based on RNN and TNN report better performance when using Manhattan distance. As for the loss function, all models performed better using Contrastive loss, which is defined as follows:

$$L(x_1, x_2, y) = \alpha(1 - y)D^2 + \beta y \max(\text{margin} - D, 0)^2 \qquad (7)$$

where x_1 and x_2 are two feature vectors, y is the paired label denoting whether the two vectors belong to the same class or not, D is the distance, and α and β are constants.

- DFNN: Dense(1024), Dropout(0.20), Dense(512), Dropout(0.20), Dense(128), Flatten().

- CNN: 1D-CNN (4), AveragePooling1D(), Dropout(0.20), 1D-CNN (16), AveragePooling1D(), Flatten(), Dense(10).
- RNN: LSTM(64), Dropout(0.20) LSTM(64), Dropout(0.20), Dense(100).
- TNN: Transformer-Encoder(headers=2), AveragePooling1D(), Dense(64), Dropout(0.20), Dense(27).

All our reported results are obtained from the testing set, containing samples that have not been seen during training time. The early comparison of the models is conducted using AUC as the comparison metric. The performance results of our models are shown in Tables (6-4). Besides, we use PCA embedding plots, to evaluate the performance of the model by visualizing the testing sample embeddings of the data in a lower-dimensional space (i.e., 2D). PCA plots can also provide insight into the distribution of the data and help identify patterns or clusters that can be difficult to see in higher-dimensional spaces. Figures [25,28, 31] show a PCA visualization of the embeddings, with different colours indicating different classes. As shown in Table ??, which shows the performance of the model with different numbers of epochs and batch sizes, SNN based on the transformer model leads to good results, reaching an AUC score of 97.56% when using a batch size of 128 with 150 epochs. The results show that increasing the number of epochs and batch size generally improves the performance of the model.

Figure 24 presents the loss curve obtained after training for 150 epochs, with the y-axis representing the value of the loss function and the x-axis representing the number of training epochs. Based on the graph, we can see that the model error is notably reducing over epochs, which indicates that the model is able to learn effectively from the training and reduces the model error considerably. Figure 25 depicts that generated embeddings, using SNN based on the transformer, of network flows belonging to the same device type but captured on different days are close to each other, while those belonging to different

Table 10: Impact of Batch Sizes on TNN over 50 Epochs.

Score	Epoch = 50				
	16	24	32	64	128
AUC	96.55	95.85	96.40	95.65	90.69
Recall	96.50	90.91	87.41	91.84	74.13
Precision	85.54	87.64	91.46	89.55	84.35
F1	90.88	89.27	89.28	90.74	78.82

Table 11: Impact of Batch Sizes on TSNN over 100 Epochs.

Score	Epoch = 100				
	16	24	32	64	128
AUC	96.50	96.93	97.28	96.43	95.31
Recall	91.14	94.64	92.81	92.07	94.41
Precision	90.93	88.65	91.65	88.57	87.47
F1	91.04	91.60	91.90	90.28	90.82

devices are far from each other. Furthermore, we plot the accuracy curve, with the y-axis representing the accuracy and the x-axis representing the number of training epochs in Figure 26, which shows the accuracy of the model over time, which is a measure of how well the model is able to predict the correct output for a given input. The plot typically shows the accuracy increasing over time as the model learns to make better predictions.

Table 12: Impact of Batch Sizes on TSNN over 150 Epochs.

Score	Epoch = 150				
	16	24	32	64	128
AUC	97.09	96.32	97.13	96.91	97.56
Recall	91.61	87.88	92.77	92.31	99.77
Precision	90.97	91.95	91.92	90.21	75.89
F1	91.32	89.70	92.41	91.28	86.43

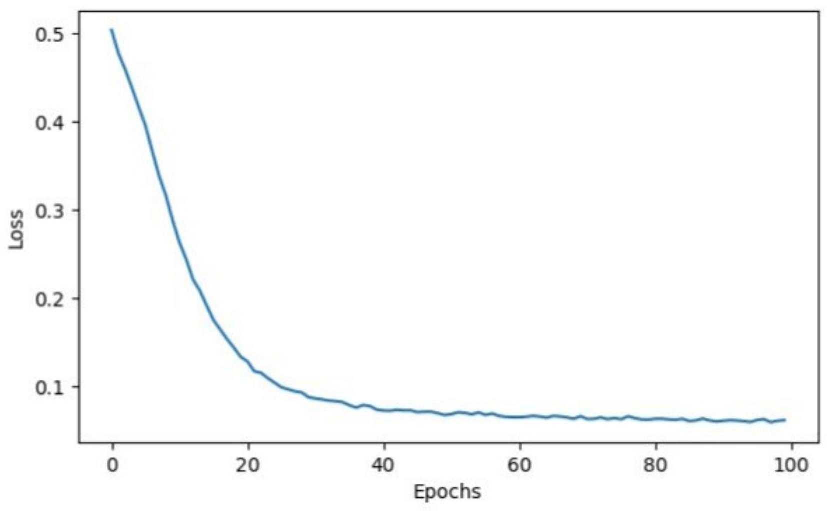


Figure 24: TNN Loss Curve.

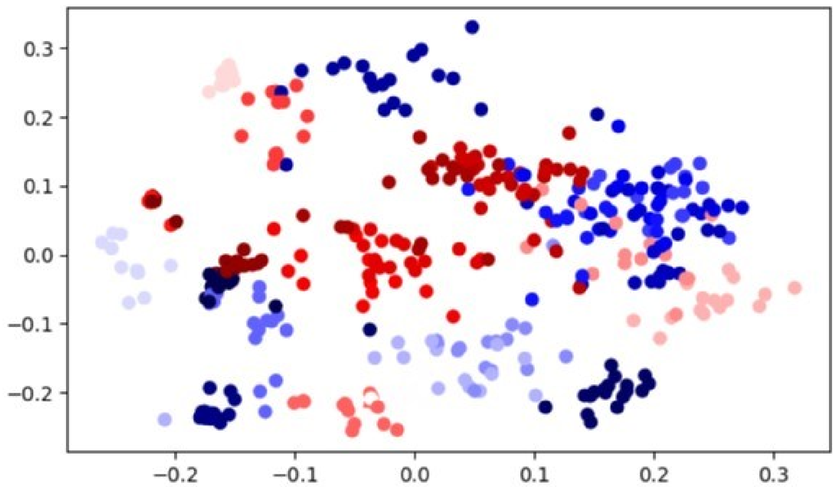


Figure 25: PCA Embedding Visualization of TNN.

Table 13: Impact of Batch Sizes on RNN over 50 Epochs.

Score	Epoch = 50				
	16	24	32	64	128
AUC	86.06	85.42	84.67	80	78.90
Recall	93.24	93.24	91.38	89.51	85.55
Precision	68.14	68.03	68.17	67.25	68.22
F1	78.81	78.74	78.15	76.84	75.91

Table 14: Impact of Batch Sizes on RNN over 100 Epochs.

Score	Epoch = 100				
	16	24	32	64	128
AUC	85.16	85.77	84.94	85.52	72.95
Recall	90.21	89.28	91.61	90.91	87.41
Precision	70.11	69.64	69.68	69.77	59.06
F1	78.94	78.32	79.22	79.01	70.54

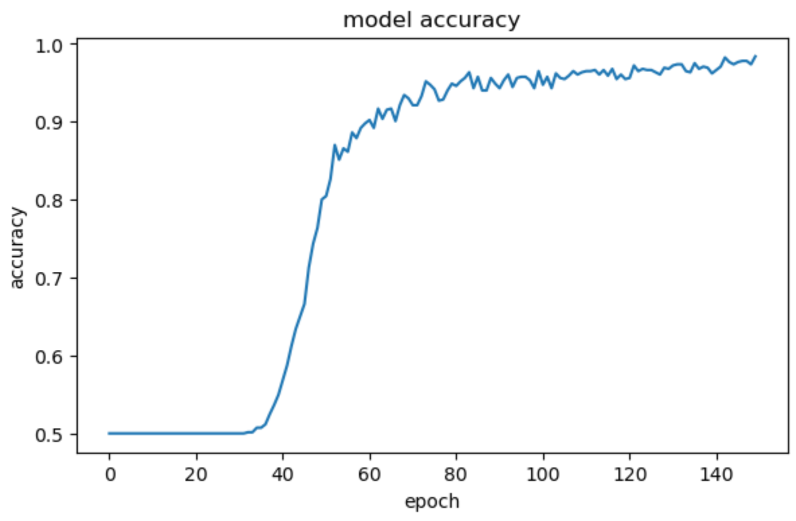


Figure 26: TNN Accuracy Curve.

Conversely, the experimental results presented in Tables [13, 14, 15], obtained using LSTM-based SNN are significantly different.

From the tables, we can see that the Transformer model consistently outperforms the LSTM model across all batch sizes and epochs on all performance metrics. The performance of both models tends to improve with increasing batch size and epochs, but the

Table 15: Impact of Batch Sizes on RNN over 150 Epochs.

Score	Epoch = 150				
	16	24	32	64	128
AUC	96.30	44.73	97.08	47.79	95.41
Recall	97.20	39.86	94.63	42.65	95.33
Precision	84.92	50.29	90.62	52.43	86.46
F1	90.76	44.33	92.65	46	90.82

Table 16: Impact of Batch Sizes on 1D-CNN over 50 Epochs.

Score	Epoch = 50				
	16	24	32	64	128
AUC	85.28	83.08	97.21	87.45	91.13
Recall	84.38	70.86	75.52	73.43	96.50
Precision	82.09	78.35	83.29	83.55	76.24
F1	83.26	74.18	79.08	77.80	85.41

Transformer model consistently achieves higher scores.

Also, Tables [16, 17, 18] show the results of SNN based on the 1D-CNN model where it achieves good results (AUC = 97.79%) when using a batch size of 32 with 100 epochs. Figure 28 depicts that embeddings generated by SNN based on 1D-CNN can capture similarity among similar devices' network flows. Related embeddings are close to each other, while dissimilar ones are far from each other. Moreover, the findings from Tables [19, 20, 21] demonstrate the performance of the DFFNN model-based SNN, which attains an AUC of 98.07% by employing a batch size of 150 and conducting 150 epochs. Figure 31 depicts that embeddings generated by SNN based on DFFNN can capture similarity among similar devices' network flows. Related embeddings are close to each other, while dissimilar ones are far from each other.

Table 17: Impact of Batch Sizes on 1D-CNN over 100 Epochs.

Score	Epoch = 100				
	16	24	32	64	128
AUC	97.21	96.89	97.79	96.28	96.11
Recall	98.14	94.41	95.10	92.31	93.01
Precision	87.71	87.10	91.07	90.21	88.67
F1	92.75	90.76	93.08	91.25	90.79

Table 18: Impact of Batch Sizes on 1D-CNN over 150 Epochs.

Score	Epoch = 150				
	16	24	32	64	128
AUC	96.30	44.74	97.08	47.80	95.40
Recall	97.20	39.86	94.64	42.66	95.33
Precision	84.93	50.29	90.62	52.43	86.46
F1	90.77	44.33	92.65	46	90.82

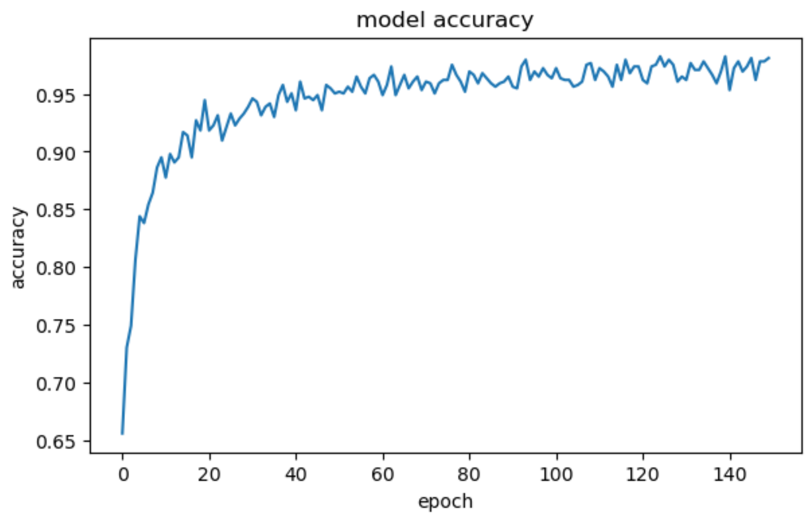


Figure 27: 1D-CNN Accuracy Plot over 150 Epochs.

Table 19: Impact of Batch Sizes on FFDNN over 50 Epochs.

Score	Epoch = 50				
	16	24	32	64	128
AUC	96.69	96.64	96.84	96.64	97.56
Recall	93.94	95.10	95.80	94.17	93.94
Precision	89.76	88.50	87.82	88.21	89.96
F1	91.87	91.80	91.78	91.18	91.98

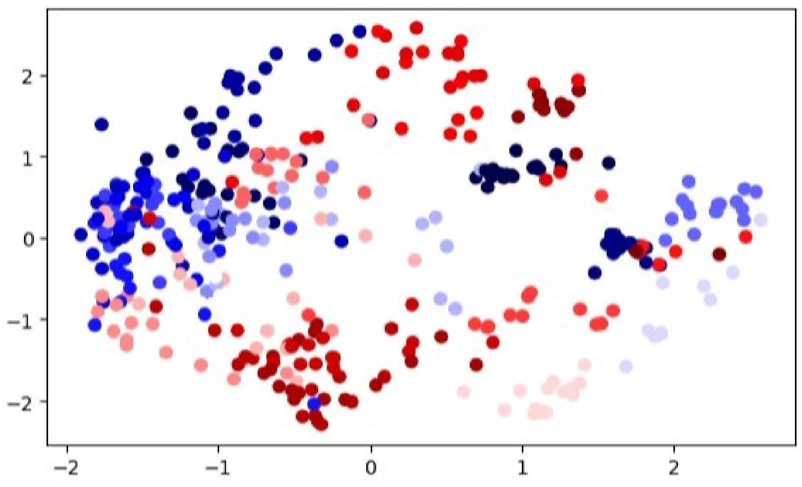


Figure 28: PCA Embedding Visualization of 1D-CNN.

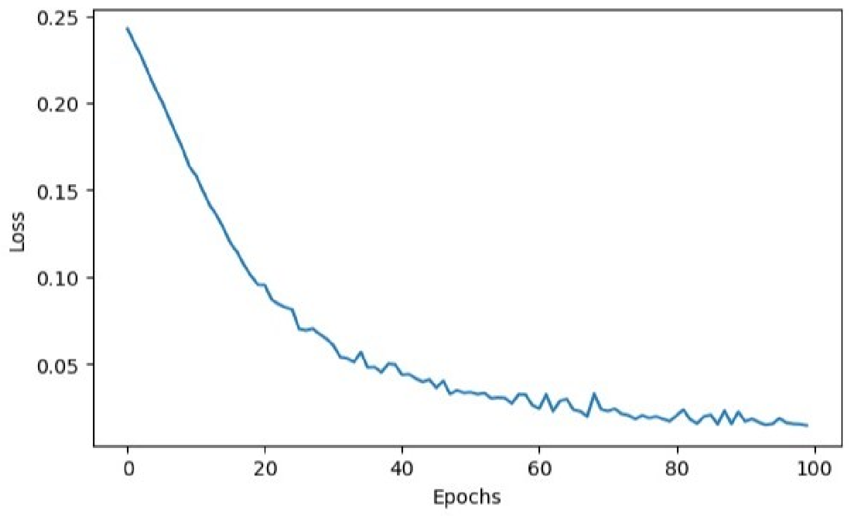


Figure 29: 1D-CNN Loss Curve Plot.

Table 20: Impact of Batch Sizes on FFDNN over 100 Epochs.

Score	Epoch = 100				
	16	24	32	64	128
AUC	95.85	97.27	97.25	97.62	95.78
Recall	94.64	94.87	94.87	95.57	95.10
Precision	91.86	89.65	92.29	89.32	91.28
F1	93.29	92.19	93.64	92.40	93.24

Table 21: Impact of Batch Sizes on FFDNN over 150 Epochs.

Score	Epoch = 150				
	16	24	32	64	128
AUC	97.43	97.29	97.26	97.99	98.07
Recall	95.57	94.17	95.57	94.64	96.04
Precision	90.31	90.79	90.91	91.24	89.6
F1	92.85	92.40	93.15	92.90	92.75

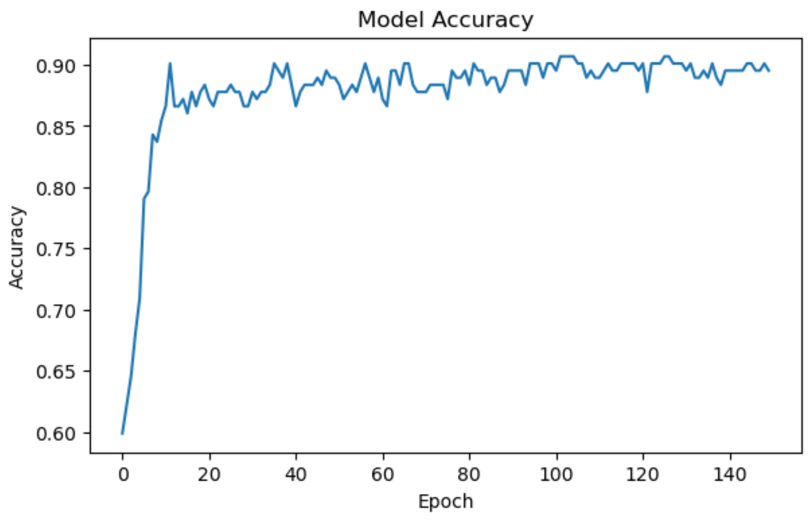


Figure 30: DFFNN Accuracy Plot over 150 Epochs.

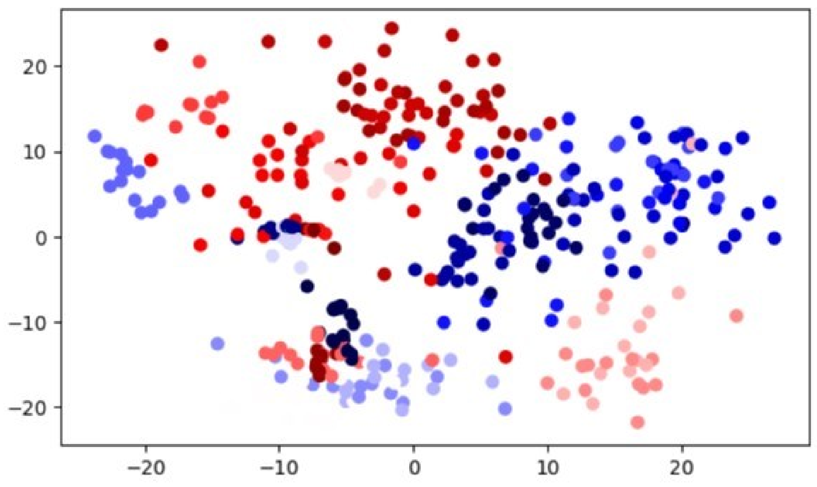


Figure 31: PCA Embedding Visualization of DFFNN.

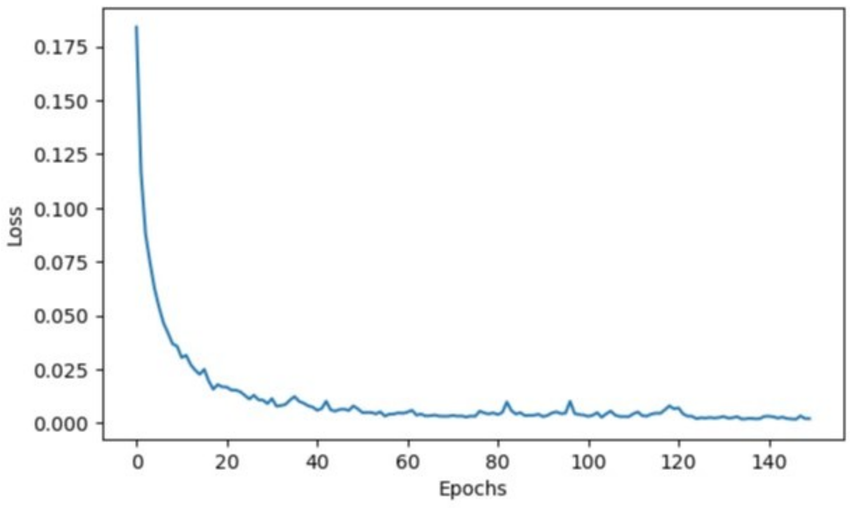


Figure 32: DFFNN Loss Curve over 150 Epochs.

Overall, the best AUC score achieved is 86.06% when using a batch size of 16 with 150 epochs. The highest overall AUC of 98.07% achieved during the experiments was by using Nadam optimizer with a learning rate of 0.001, batch size of 128 and 150 epochs. Consequently, based on our reported results, we can state that deep learning models based on few-shot learning can be effective and yield good results on small data.

5.6 Comparison with Existing Approach

The comparison with existing approaches will provide a better understanding of the advantages and limitations of our approach. In this section, we compare our results with those of the IoT Sentinel approach [7], which used the same dataset for the same problem. Our approach utilizes a different type of features and machine learning models than previous studies, making direct comparison challenging. To overcome this challenge, we extract and use the same set of features used in [7], which is presented in Table 22 in order to 1) evaluate the effectiveness of our optimized feature selection method in improving the results reported in [7], and 2) train our model using Sentinel features.

Table 22: Set of Features used in IoT Sentinel [7].

Type	Features
Link layer protocol (2)	ARP / LLC
Network layer protocol (4)	IP / ICMP / ICMPv6 / EAPoL
Transport layer protocol (2)	TCP / UDP
Application layer protocol (8)	HTTP / HTTPS / DHCP / BOOTP / SSDP / DNS / MDNS / NTP
IP options (2)	Padding / RouterAlert
Packet content (2)	Size (int) / Raw data
IP address (1)	Destination IP counter (int)
Port class (2)	Source (int) / Destination (int)

The features are binary (set to 1 if some selected communication protocols are used and set to 0 otherwise), except the ones marked with (int) in the table, which are integers. Feature selection was performed using our feature selection process presented in Section 4.3.

After only applying Unsupervised Feature Selection, the number of features was reduced from 23 to 18. Recall from Section 5.4.1 that zero-variance features are features that have the same value for all instances. In other words, these features have zero variance and do not provide any useful information for a machine-learning algorithm. Figures [33, 34, 35] show an example of zero-variance features (or constant features), namely IP, ICMP and IP_Padding respectively, where each feature is identified and represented by a vertical line at x=0. This line indicates that all instances (i.e., all classes) in the dataset have a value of 0 for this particular feature.

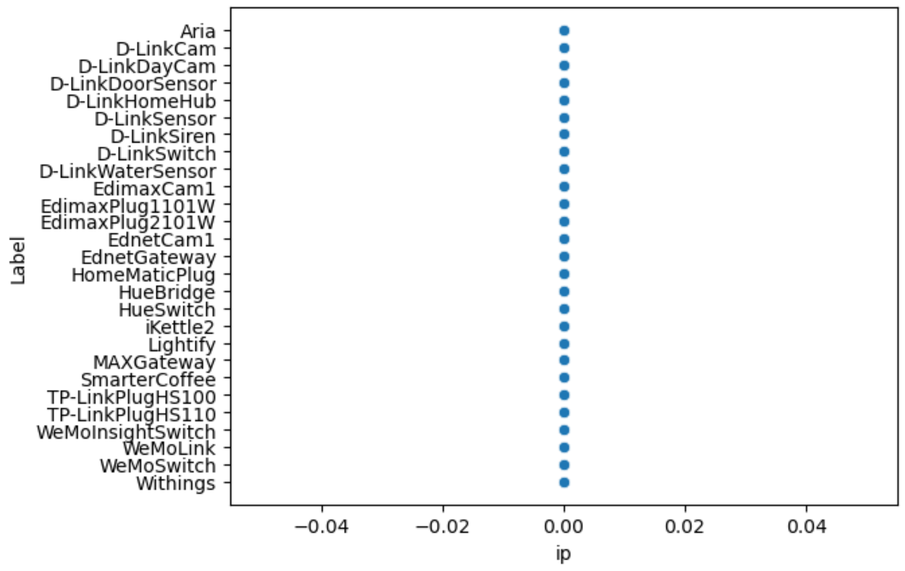


Figure 33: Scatter showing that IP Feature value is 0 for All IoT Types.

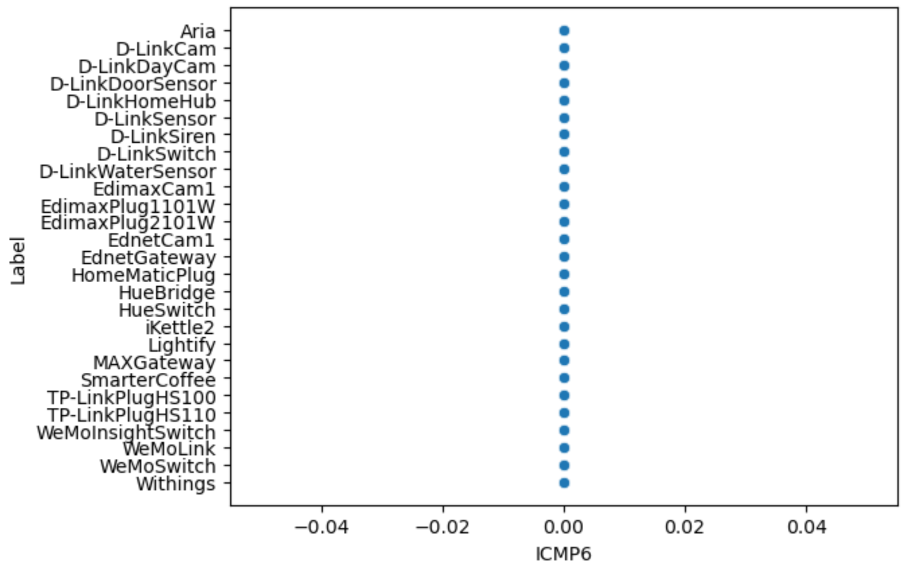


Figure 34: Scatter showing that ICMP Feature value is 0 for All IoT Types.

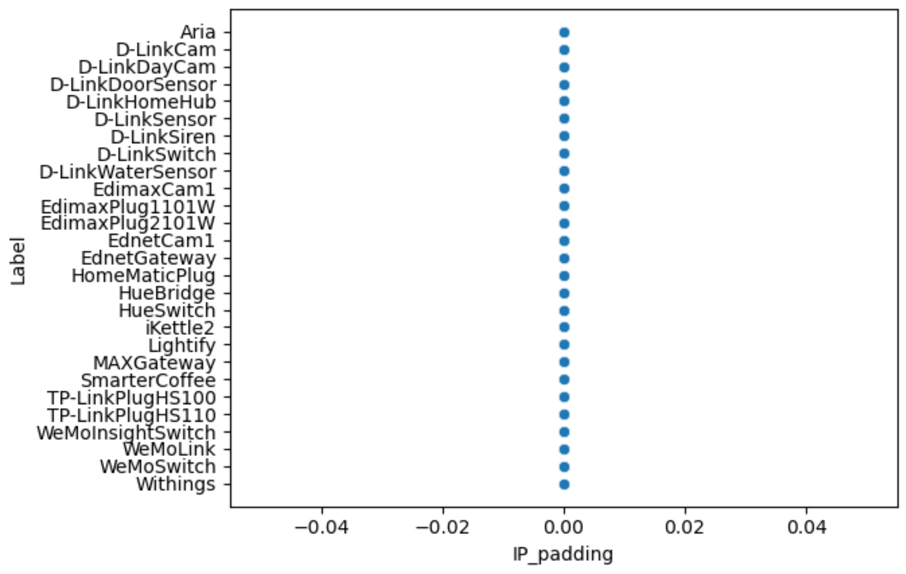


Figure 35: Scatter showing that IP_Padding Feature value is 0 for All IoT Types.

Further application of the supervised feature selection reduced the number of features to 12 without affecting the model performance as shown in Figure 36. The final subset of selected features includes ARP, Packet Size, Destination IP Count, Source Port, Destination Port, TCP, HTTPS, HTTP, SSDP, DNS, MDNS, and NTP.

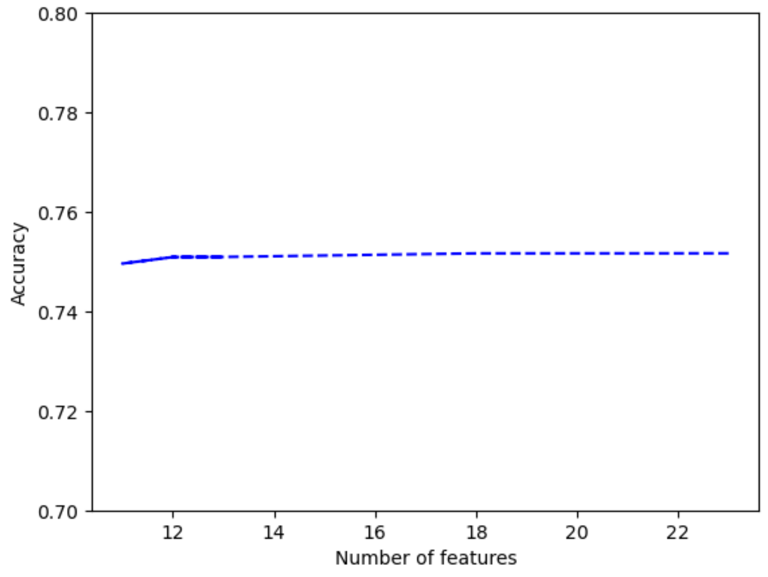
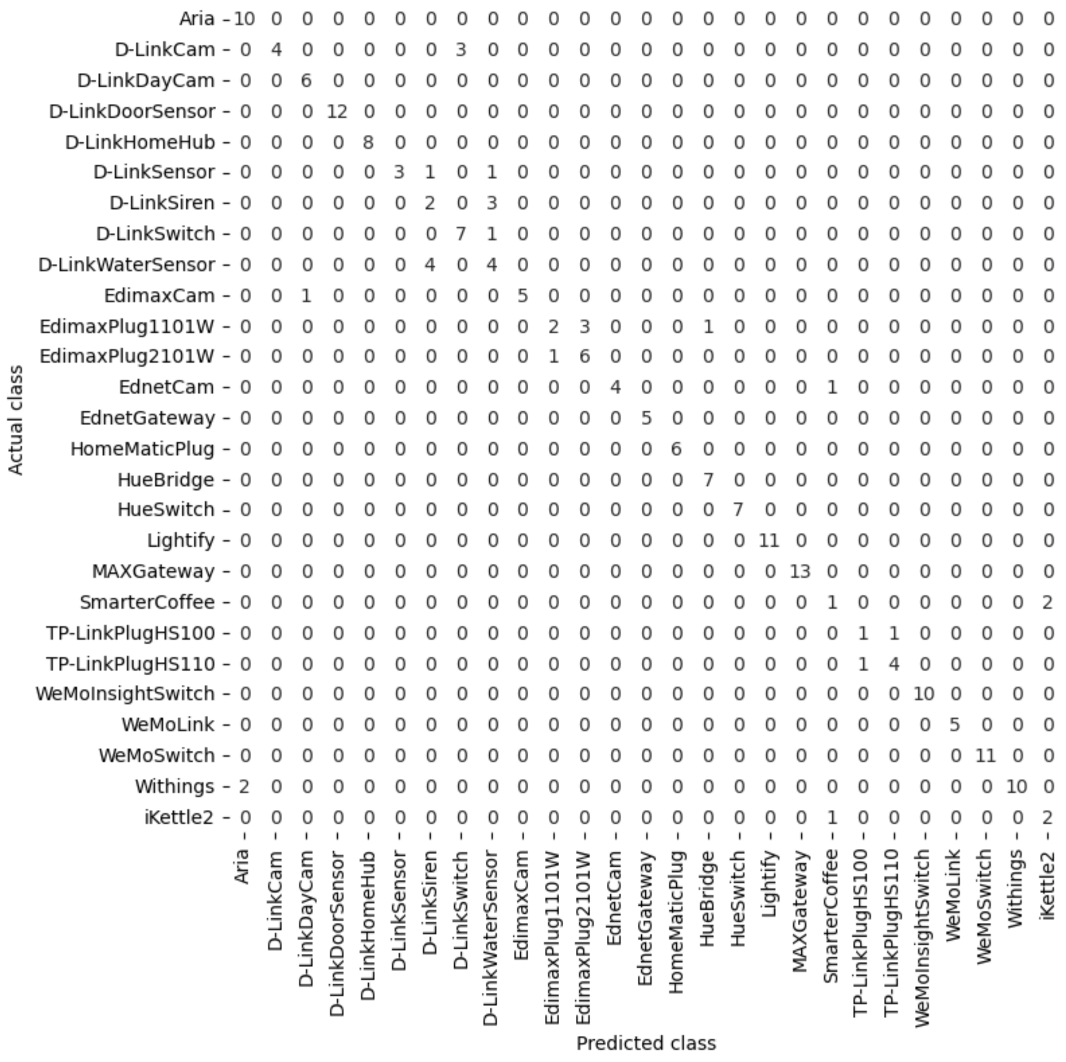


Figure 36: Feature Selection on IoT Sentinel [7] Features.

After selecting the most informative features, we trained a One-vs-All Random Forest classification, which is a "one classifier per device-type" approach that builds a binary classifier for each class. When comparing our results to those of [7], it must be pointed out that we were able to reduce the dimensionality of the problem and improve the model's performance from 81% to 86%. To provide insight into how better the model is performing using the selected features, the confusion matrix is presented in Figure 37.


Actual class
Predicted class

	Aria	D-LinkCam	D-LinkDayCam	D-LinkDoorSensor	D-LinkHomeHub	D-LinkSensor	D-LinkSiren	D-LinkSwitch	D-LinkWaterSensor	EdimaxCam	EdimaxPlug1101W	EdimaxPlug2101W	EdnetCam	EdnetGateway	HomeMaticPlug	HueBridge	HueSwitch	Lightify	MAXGateway	SmarterCoffee	TP-LinkPlugHS100	TP-LinkPlugHS110	WeMoInsightSwitch	WeMoLink	WeMoSwitch	Withings	iKettle2
Aria	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkCam	0	4	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkDayCam	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkDoorSensor	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkHomeHub	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkSensor	0	0	0	0	0	3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkSiren	0	0	0	0	0	0	2	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkSwitch	0	0	0	0	0	0	0	7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D-LinkWaterSensor	0	0	0	0	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EdimaxCam	0	0	1	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EdimaxPlug1101W	0	0	0	0	0	0	0	0	0	0	2	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
EdimaxPlug2101W	0	0	0	0	0	0	0	0	0	0	1	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EdnetCam	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0
EdnetGateway	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
HomeMaticPlug	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0
HueBridge	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0
HueSwitch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0
Lightify	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0
MAXGateway	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0
SmarterCoffee	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2
TP-LinkPlugHS100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
TP-LinkPlugHS110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0
WeMoInsightSwitch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0
WeMoLink	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
WeMoSwitch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0
Withings	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0
iKettle2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2

Figure 37: Confusion Matrix for a One-vs-All Random Forest Classification Model.The x-axis and y-axis Represent the Predicted and Actual or True Class Labels, Respectively.

Additionally, in order to conduct a meaningful comparison of the two approaches, the same evaluation metric is used, namely accuracy, to report the results that were achieved. In this work, an automatic feature selection is employed, which results in the selection of 31 features from the network flow data to train a Siamese neural network composed of four different neural networks, as previously detailed. In contrast, the authors of IoT Sentinel extracted from packet headers a number of 23 features, which they use to train a random forest model for each device type. Their reported accuracy is 81% on average. Compared to IoT Sentinel, depending on the model configuration, our proposed approach achieves the following results: 92.31% for SNN based on a transformer; 91.16% for SNN based on DFFNN; 90.21% for SNN based on CNN; and 80.07% for SNN based on RNN. Thus, except for the last configuration, our proposed models outperform the IoT Sentinal approach, achieving around 10% higher accuracy. In addition, it is worth mentioning that employing few-shot learning allow us to handle even situations where the availability of labeled IoT traffic is limited. The reason is that our approach uses SNN to compare the similarity between traffic features. Also, our approach does not require retraining a model when a new device connects to the network. Likewise, it does not require training a model for each device. In contrast, IoT Sentinel relies on training a classifier for each device and requires training a new model when a new device connects to the network. The results of this study demonstrate the effectiveness of feature selection in improving model performance and reducing computational load. The selection of relevant features not only improves the accuracy of the model but also makes it more interpretable by eliminating irrelevant or redundant features. In conclusion, feature selection is an important process in machine learning that can significantly improve model performance and reduce computational complexity. A further finding is that few-shot learning models outperform the model in [7].

Chapter 6

Conclusion

This thesis investigated the use of deep learning techniques for identifying the types of Internet of Things devices based on their network traffic patterns. We began by discussing the importance of device identification for securing IoT networks and reviewed the relevant literature on deep learning for device identification. A hybrid feature selection approach is presented to automatically select the best features without relying on feature engineering and domain knowledge. The feature selection method combines and utilizes multiple methods to achieve a more generalized solution. The comparison between the two machine learning models (with and without feature selection) revealed some interesting findings. After applying feature selection, the model achieved a similar or even better performance than the original model, using fewer features and requiring less computational resources. This suggests that feature selection is an important step in machine learning that can help improve the performance of models, particularly when dealing with high-dimensional data. We then proposed and presented a novel deep learning-based solution for IoT device type identification, which we evaluated using a real-world network traffic dataset. Our approach employs the few-shot learning techniques based on the Siamese neural network. The proposed solution is based on passive fingerprinting using bidirectional flows, which are extracted from the encrypted network traffic. Four deep learning models, including DFFNN, CNN, RNN and TNN model, have been applied and evaluated on a real IoT dataset. The experimental results show that the proposed models are effective and efficient. Our results showed that our proposed solution effectively and accurately identified the types of IoT devices with high precision and recall rates. We also compared the performance of our proposed solution with [7] and showed that our deep learning model outperformed these algorithms in terms of accuracy and computational efficiency.

Our study also addressed the challenges of using deep learning for IoT device-type identification. These challenges included the need for large datasets,

Overall, this study contributes to the growing body of research on using deep learning for IoT security and provides insights into the potential and challenges of using deep learning techniques for identifying IoT device types. We hope that our findings will inspire further research and innovation in this important area and help improve IoT ecosystems' security and privacy.

In future work, we aim to explore and investigate how our approach behaves on industrial IoT datasets. We also aim to extend the fingerprinting process to identify an anomalous device within the network (i.e., unexpected device behavior compared to its normal or intended functioning). Furthermore, it may be useful to investigate Neural Network Search (NAS), such as Google Search [97], to automate the design of neural network architectures for SNNs and the hyper-parameter tuning.

Bibliography

[1] Maria Stoyanova, Yannis Nikoloudakis, Spyridon Panagiotakis, Evangelos Pallis, and Evangelos K Markakis. A survey on the internet of things (iot) forensics: challenges, approaches, and open issues. IEEE Communications Surveys & Tutorials, 22(2):1191–1221, 2020.

[2] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. IEEE Communications Surveys Tutorials, 21(3):2702–2733, 2019.

[3] Alex Shenfield and Martin Howarth. A novel deep learning model for the detection and identification of rolling element-bearing faults. Sensors, 20(18):5112, 2020.

[4] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long short term memory recurrent neural network classifier for intrusion detection. In 2016 International Conference on Platform Technology and Service (PlatCon), pages 1–5, 2016.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[6] Kyle Martin, Nirmalie Wiratunga, Sadiq Sani, and Stewart Massie. A convolutional siamese network for developing similarity knowledge in the selfback dataset. CEUR Workshop Proceedings, 2017.

[7] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pages 2177–2184. IEEE, 2017.

[8] Sam Lucero et al. Iot platforms: enabling the internet of things. White paper, 2016.

[9] Vijayanand Thangavelu, Dinil Mon Divakaran, Rishi Sairam, Suman Sankar Bhunia, and Mohan Gurusamy. Deft: A distributed iot fingerprinting technique. IEEE Internet of Things Journal, 6(1):940–952, 2018.

[10] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In 26th {USENIX} Security Symposium ({USENIX} Security 17), pages 1093–1110, 2017.

[11] Donald R Reising, Michael A Temple, and Julie A Jackson. Authorized and rogue device discrimination using dimensionally reduced rf-dna fingerprints. IEEE Transactions on Information Forensics and Security, 10(6):1180–1192, 2015.

[12] Nicolas Sklavos and Odysseas G Koufopavlou. Mobile communications world: Security implementations aspects-a state of the art. Comput. Sci. J. Moldova, 11(2):168–187, 2003.

[13] Wen-Chuan Hsieh, Yi-Hsien Chiu, and Chi-Chun Lo. An interference-based prevention mechanism against wep attack for 802.11 b network. In International Conference on Network Control and Engineering for QoS, Security and Mobility, pages 127–138. Springer, 2004.

[14] Oracle. What is iot?

[15] Mark Patel, Jason Shangkuan, and Christopher Thomas. What's new with the internet of things?, Jan 2018.

[16] Fredrik Dahlqvist, Mark Patel, Alexander Rajko, and Jonathan Shulman. Growing opportunities in the internet of things, Sep 2020.

[17] Tim Hahn. Internet of threats securing the internet of things for industrial and utility companies. [Online] Available at https://www.ibm.com/downloads/cas/ZJRRVRKW, Mar 2018.

[18] Alex Schiffer. How a fish tank helped hack a casino.

[19] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. A survey on security and privacy issues in internet-of-things. IEEE Internet of Things Journal, 4(5):1250–1258, 2017.

[20] ISTR Symantec. Executive summary 2018 internet security threat report. Symantec Corporation, USA, 123:04, 2018.

[21] Selena Larson. Fda confirms that st. jude's cardiac devices can be hacked, Jan 2017.

[22] HP News. Hp study reveals 70 percent of internet of things devices vulnerable to attack, Jul 2014.

[23] Zscaler. Iot devices in the enterprise 2020: Shadow iot threat emerges.

[24] Yingying Chen, Wade Trappe, and Richard P Martin. Detecting and localizing wireless spoofing attacks. In 2007 4th Annual IEEE Communications Society Conference on sensor, mesh and ad hoc communications and networks, pages 193–202. IEEE, 2007.

[25] Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. Defending against vm rollback attack. In IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012), pages 1–5. IEEE, 2012.

[26] Chris Karlof and David Wagner. Hidden markov model cryptanalysis. In International Workshop on Cryptographic Hardware and Embedded Systems, pages 17–34. Springer, 2003.

[27] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. ACM SIGCOMM Computer Communication Review, 34(2):39–53, 2004.

[28] Sana Benzarti, Bayrem Triki, and Ouajdi Korbaa. A survey on attacks in internet of things based networks. In 2017 International conference on engineering & MIS (ICEMIS), pages 1–7. IEEE, 2017.

[29] Y-C Hu, Adrian Perrig, and David B Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428), volume 3, pages 1976–1986. IEEE, 2003.

[30] Mohan V Pawar and J Anuradha. Network security and types of attacks in network. Procedia Computer Science, 48:503–506, 2015.

[31] Robert Sloan and Richard Warner. Unauthorized access: The crisis in online privacy and security. Taylor & Francis, 2017.

[32] J Tidy. Hacker tries to poison water supply of florida city. BBC News, 2021.

[33] Miraqa Safi, Sajjad Dadkhah, Farzaneh Shoeleh, Hassan Mahdikhani, Heather Molyneaux, and Ali A Ghorbani. A survey on iot profiling, fingerprinting, and identification. ACM Transactions on Internet of Things, 2022.

[34] Qiang Xu, Rong Zheng, Walid Saad, and Zhu Han. Device fingerprinting in wireless networks: Challenges and opportunities. IEEE Communications Surveys & Tutorials, 18(1):94–104, 2015.

[35] Shancang Li, Kim-Kwang Raymond Choo, Qindong Sun, William J Buchanan, and Jiuxin Cao. Iot forensics: Amazon echo as a use case. IEEE Internet of Things Journal, 6(4):6487–6497, 2019.

[36] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile web users through motion sensors: Attacks and defenses. In NDSS, 2016.

[37] Sebastian Zimmeck, Jie S Li, Hyungtae Kim, Steven M Bellovin, and Tony Jebara. A privacy analysis of cross-device tracking. In 26th USENIX Security Symposium (USENIX Security 17), pages 1391–1408, 2017.

[38] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In 2013 IEEE Symposium on Security and Privacy, pages 541–555, 2013.

[39] Kai Yang, Qiang Li, Xiaodong Lin, Xin Chen, and Limin Sun. ifinger: Intrusion detection in industrial control systems via register-based fingerprinting. IEEE Journal on Selected Areas in Communications, 38(5):955–967, 2020.

[40] Chao Shen, Chang Liu, Haoliang Tan, Zhao Wang, Dezhi Xu, and Xiaojie Su. Hybrid-augmented device fingerprinting for intrusion detection in industrial control system networks. IEEE Wireless Communications, 25(6):26–31, 2018.

[41] David Formby, Preethi Srinivasan, Andrew Leonard, Jonathan Rogers, and Raheem A Beyah. Who's in control of your control system? device fingerprinting for cyber-physical systems. In NDSS, 2016.

[42] Michael D Twa, Srinivasan Parthasarathy, Cynthia Roberts, Ashraf M Mahmoud, Thomas W Raasch, and Mark A Bullimore. Automated decision tree classification of corneal shape. Optometry and vision science: official publication of the American Academy of Optometry, 82(12):1038, 2005.

[43] Leo Breiman. Random forests. Machine learning, 45:5–32, 2001.

[44] Vladimir Vapnik. The nature of statistical learning theory. Springer science & business media, 1999.

[45] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1):21–27, 1967.

[46] Allan Pinkus. Approximation theory of the mlp model in neural networks. Acta numerica, 8:143–195, 1999.

[47] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458, 2015.

[48] L Busk Linnebjerg and R Wetke. Long short term memory. Hear. Balanc. Commun, 12:36–40, 1997.

[49] Kai Yang, Qiang Li, and Limin Sun. Towards automatic fingerprinting of iot devices in the cyberspace. Computer Networks, 148:318–327, 2019.

[50] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. Characterizing industrial control system devices on the internet. In 2016 IEEE 24th International Conference on Network Protocols (ICNP), pages 1–10. IEEE, 2016.

[51] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and N Asokan. Audi: Toward autonomous iot device-type identification using periodic communication. IEEE Journal on Selected Areas in Communications, 37(6):1402–1412, 2019.

[52] Mustafizur R Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Iot devices recognition through network traffic analysis. In 2018 IEEE international conference on big data (big data), pages 5187–5192. IEEE, 2018.

[53] Samaresh Bera, Sudip Misra, and Athanasios V. Vasilakos. Software-defined networking for internet of things: A survey. IEEE Internet of Things Journal, 4(6):1994–2008, 2017.

[54] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys Tutorials, 17(4):2347–2376, 2015.

[55] Anat Bremler-Barr, Haim Levy, and Zohar Yakhini. Iot or not: Identifying iot devices in a short time scale. In NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, pages 1–9, 2020.

[56] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. Profiliot: a machine learning approach for iot device identification based on network traffic analysis. In Proceedings of the symposium on applied computing, pages 506–509, 2017.

[57] Antônio J Pinheiro, Jeandro de M Bezerra, Caio AP Burgardt, and Divanilson R Campelo. Identifying iot devices and events based on packet length from encrypted traffic. Computer Communications, 144:8–17, 2019.

[58] Linna Fan, Lin He, Enhuan Dong, Jiahai Yang, Chenglong Li, Jinlei Lin, and Zhiliang Wang. Evoiot: An evolutionary iot and non-iot classification model in open environments. Computer Networks, 219:109450, 2022.

[59] Jorge Ortiz, Catherine Crawford, and Franck Le. Devicemien: network device behavior modeling for identifying unknown iot devices. In Proceedings of the International Conference on Internet of Things Design and Implementation, pages 106–117, 2019.

[60] Linna Fan, Shize Zhang, Yichao Wu, Zhiliang Wang, Chenxin Duan, Jia Li, and Jiahai Yang. An iot device identification method based on semi-supervised learning. In 2020 16th International Conference on Network and Service Management (CNSM), pages 1–7, 2020.

[61] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 559–564. IEEE, 2017.

[62] Nesrine Ammar, Ludovic Noirie, and Sebastien Tixeuil. Autonomous identification of iot device types based on a supervised classification. In ICC 2020 - 2020 IEEE International Conference on Communications (ICC), pages 1–6, 2020.

[63] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of iot devices. In Proceedings of the 2018 workshop on attacks and solutions in hardware security, pages 41–50, 2018.

[64] Yantian Luo, Xu Chen, Ning Ge, Wei Feng, and Jianhua Lu. Transformer-based device type identification in heterogeneous iot traffic. IEEE Internet of Things Journal, pages 1–1, 2022.

[65] Jiaqi Bao, Bechir Hamdaoui, and Weng-Keen Wong. Iot device type identification using hybrid deep learning approach for increased iot security. In 2020 International Wireless Communications and Mobile Computing (IWCMC), pages 565–570, 2020.

[66] Franck Le, Jorge Ortiz, Dinesh Verma, and Dilip Kandlur. Policy-based identification of iot devices' vendor and type by dns traffic analysis. Policy-Based Autonomic Data Governance, pages 180–201, 2019.

[67] Leonardo Babun, Hidayet Aksu, Lucas Ryan, Kemal Akkaya, Elizabeth S. Bentley, and A. Selcuk Uluagac. Z-iot: Passive device-class fingerprinting of zigbee and z-wave iot devices. In ICC 2020 - 2020 IEEE International Conference on Communications (ICC), pages 1–7, 2020.

[68] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. IEEE Transactions on Mobile Computing, 18(8):1745–1759, 2018.

[69] Hossein Jafari, Oluwaseyi Omotere, Damilola Adesina, Hsiang-Huang Wu, and Lijun Qian. Iot devices fingerprinting using deep learning. In MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM), pages 1–9, 2018.

[70] Girish Vaidya, Akshay Nambi, T.V. Prabhakar, Vasanth Kumar T, and Suhas Sudhakara. Iot-id: A novel device-specific identifier based on unique hardware fingerprints. In 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), pages 189–202, 2020.

[71] Florent Galtier, Romain Cayre, Guillaume Auriol, Mohamed Kâaniche, and Vincent Nicomette. A psd-based fingerprinting approach to detect iot device spoofing. In 2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC), pages 40–49, 2020.

[72] Linning Peng, Aiqun Hu, Junqing Zhang, Yu Jiang, Jiabao Yu, and Yan Yan. Design of a hybrid rf fingerprint extraction and device classification scheme. IEEE Internet of Things Journal, 6(1):349–360, 2019.

[73] Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. Machine learning in iot security: Current solutions and future challenges. IEEE Communications Surveys & Tutorials, 22(3):1686–1721, 2020.

[74] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. IEEE Communications Surveys & Tutorials, 22(3):1646–1685, 2020.

[75] Ruizhong Du, Jingze Wang, and Shuang Li. A lightweight flow feature-based iot device identification scheme. Security and Communication Networks, 2022, 2022.

[76] Kahraman Kostas, Mike Just, and Michael A Lones. Iotdevid: A behavior-based device identification method for the iot. IEEE Internet of Things Journal, 2022.

[77] Rajarshi Roy Chowdhury, Sandhya Aneja, Nagender Aneja, and Emeroylariffion Abas. Network traffic analysis based iot device identification. In Proceedings of the 2020 the 4th International Conference on Big Data and Internet of Things, pages 79–89, 2020.

[78] Phornsawan Roemsri and Rattikorn Hewett. Device identification for iot security. In 2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP), pages 866–870, 2021.

[79] Chenxin Duan, Hao Gao, Guanglei Song, Jiahai Yang, and Zhiliang Wang. Byteiot: A practical iot device identification system based on packet length distribution. IEEE Transactions on Network and Service Management, 19(2):1717–1728, 2022.

[80] Salma Abdalla Hamad, Wei Emma Zhang, Quan Z. Sheng, and Surya Nepal. Iot device identification via network-flow based fingerprinting and learning. In 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pages 103–111, 2019.

[81] Ibbad Hafeez, Markku Antikainen, Aaron Yi Ding, and Sasu Tarkoma. Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge. IEEE Transactions on Network and Service Management, 17(1):45–59, 2020.

[82] Jay Thom, Nathan Thom, Shamik Sengupta, and Emily Hand. Smart recon: Network traffic fingerprinting for iot device identification. In 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), pages 0072–0079, 2022.

[83] Kevin Merchant, Shauna Revay, George Stantchev, and Bryan Nousain. Deep learning for rf device fingerprinting in cognitive communication networks. IEEE Journal of Selected Topics in Signal Processing, 12(1):160–167, 2018.

[84] Sandhya Aneja, Nagender Aneja, and Md Shohidul Islam. Iot device fingerprint using deep learning. In 2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS), pages 174–179, 2018.

[85] Lei Bai, Lina Yao, Salil S. Kanhere, Xianzhi Wang, and Zheng Yang. Automatic device classification from network traffic streams of internet of things. In 2018 IEEE 43rd Conference on Local Computer Networks (LCN), pages 1–9, 2018.

[86] Feihong Yin, Li Yang, Jianfeng Ma, Yasheng Zhou, Yuchen Wang, and Jiahao Dai. Identifying iot devices based on spatial and temporal features from network traffic. Security and Communication Networks, 2021, 2021.

[87] Max Kuhn, Kjell Johnson, et al. Applied predictive modeling, volume 26. Springer, 2013.

[88] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. Journal of machine learning research, 3(Mar):1157–1182, 2003.

[89] Ron Kohavi and George H John. Wrappers for feature subset selection. Artificial intelligence, 97(1-2):273–324, 1997.

[90] Soledad Galli. Python feature engineering cookbook: over 70 recipes for creating, engineering, and transforming features to build machine learning models. Packt Publishing Ltd, 2020.

[91] Bernhard Scholkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, John Platt, et al. Support vector method for novelty detection. Advances in neural information processing systems, 12(3):582–588, 2000.

[92] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C Prati, Bartosz Krawczyk, and Francisco Herrera. Learning from imbalanced data sets, volume 10. Springer, 2018.

[93] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In International conference on machine learning, pages 4393–4402. PMLR, 2018.

[94] Gregory W Corder and Dale I Foreman. Nonparametric statistics for non-statisticians, 2011.

[95] Mario Beraha, Alberto Maria Metelli, Matteo Papini, Andrea Tirinzoni, and Marcello Restelli. Feature selection via mutual information: New theoretical insights. In 2019 international joint conference on neural networks (IJCNN), pages 1–9. IEEE, 2019.

[96] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. Advances in neural information processing systems, 28, 2015.

[97] Hanna Mazzawi, Xavi Gonzalvo, Aleks Kracun, Prashant Sridhar, Niranjan Subrahmanya, Ignacio Lopez-Moreno, Hyun-Jin Park, and Patrick Violette. Improving keyword spotting and language identification via neural architecture search at scale. In Interspeech, pages 1278–1282, 2019.