Lightweight intrusion detection of attacks on the Internet of Things (IoT) in critical infrastructures.

OTOKWALA, U.J.

2024

Lightweight Intrusion Detection of Attacks On the Internet of Things (IoT) in Critical infrastructures

Uneneibotejit Job Otokwala

A thesis submitted in partial fulfilment
of the requirements of
Robert Gordon University
for the degree of Doctor of Philosophy

May 2024

Abstract

Critical Infrastructures (CI) are essential for various aspects of human activities, spanning across different sectors. However, the integration of Internet of Things (IoT) devices into CI has introduced a new dimension to security challenges due to IoT vulnerabilities. Traditional Machine Learning (ML) and Deep Learning (DL) approaches have proven to offer promising results in detecting intrusions; however, their computational demands make them impractical for resource-constrained IoT devices. This study proposes an Optimized Common Features Selection and Deep Autoencoder (OCFSDA) technique for lightweight intrusion detection, which is computationally efficient and cost-effective for the IoT. The OCFSDA was achieved by leveraging Shallow Deep Learning to develop a lightweight intrusion detection model suitable for IoT devices. The key contributions of this research, as contained in this thesis, are highlighted as follows:

Feature Selection and Augmentation: Initial experiments with various machine learning algorithms on smart grid datasets revealed challenges with class imbalance, requiring data augmentation. Moreover, feature selection was essential to reduce dimensionality, but single techniques produced suboptimal results. To address this, an ensemble of feature selection techniques was employed to generate a common feature subset compatible with multiple learning algorithms.

Deep Autoencoder-based Feature Extraction: The Common Feature Technique (CFT) subset underwent feature extraction using LSTM Autoencoder, resulting in a bottleneck layer of 5 nodes. These extracted features were then subjected to Shallow Deep Learning, evaluated, pruned, and deparameterized.

Resilience Against Adversarial Attacks: Adversarial training with semi-supervised learning enhanced the OCFSDA model's resilience against adversarial attacks. The model was further optimized using quantization techniques.

Performance Evaluation: Experimental results using benchmark IoT datasets (MQTT-IoT-IDS2020, CICIDS2017) on both Windows and Raspberry Pi 4 platforms demonstrated impressive performance. The OCFSDA model achieved high overall accuracy (99% and 97% on respective datasets) with significantly reduced execution times (0.30s and 0.12s) and memory usage (2KB). Moreover, the model exhibited robustness against adversarial attacks, outperforming benchmark models in terms of accuracy and recall.

Overall, the proposed OCFSDA model offers a promising solution for lightweight intrusion detection in IoT devices, addressing computational constraints while ensuring high performance and resilience against cyber threats.

Keywords: Internet of Things, Machine Learning, Deep Learning, Feature Selection, Intrusion Detection, Data Augmentation, Data Compression.

Declaration of Authorship

I hereby declare that I am the exclusive author of this thesis. All excerpts within this document have been explicitly identified, and all sources of information have been appropriately referenced and acknowledged in the bibliography. It is important to note that some portions of the work presented in this thesis have been previously published in the following publications.

1. U. Otokwala, A. Petrovski and H. Kalutarage, "Improving Intrusion Detection Through Training Data Augmentation," 2021 14th International Conference on Security of Information and Networks (SIN), Edinburgh, United Kingdom, 2021, pp. 1-8, doi: 10.1109/SIN54109.2021.9699293.

2. Otokwala, U., Petrovski, A., Kalutarage, H. (2021). Effective Detection of Cyber Attack in a Cyber-Physical Power Grid System. In: Arai, K. (eds) Advances in Information and Communication. FICC 2021. Advances in Intelligent Systems and Computing, vol 1363. Springer, Cham. https://doi.org/10.1007/978-3-030-73100-7_57

3. Otokwala, U., Arifeen, M., & Petrovski, A. (2022, September). A Comparative Study of Novelty Detection Models for Zero Day Intrusion Detection in Industrial Internet of Things. In UK Workshop on Computational Intelligence (pp. 238-249). Cham: Springer Nature Switzerland.

4. U. J. Otokwala and A. Petrovski, "Ensemble Common Features Technique for Lightweight Intrusion Detection in Industrial Control System," 2023 IEEE 6th International Conference on Industrial Cyber-Physical Systems (ICPS), Wuhan, China, 2023, pp. 1-6, doi: 10.1109/ICPS58381.2023.10128040.

5. Otokwala, U., Petrovski, A., & Kalutarage, H. (2024). Optimized common features selection and deep-autoencoder (OCFSDA) for lightweight intrusion detection in Internet of Things. International Journal of Information Security, 1-23.

6. U. J. Otokwala, A. Petrovski, I. Kotenko. "Enhancing Intrusion Detection Through Data Perturbation Augmentation Strategy". 2024 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT) - **accepted - awaiting publication**

Acknowledgements

I extend my heartfelt gratitude to my supervisory team, ably led by Dr. Andrei Petrovski and Dr. Harsha Kalutarage, for their unwavering support and guidance throughout my academic journey. Your encouragement, motivational advice, and inspirational contributions have been instrumental in shaping my growth as a researcher and writer. Indeed, I am truly thankful for the opportunities to learn and develop under your mentorship. In addition, I express my profound gratitude to the Niger Delta Development Commission (NDDC) for awarding me a scholarship for my Ph.D. research. The financial support and commitment provided by the commission has played a pivotal role in creating a conducive environment for my research, minimizing distractions, and enabling me to focus on my academic pursuits. I am also indebted to my elder brother, Dr. Job Otokwala, for his consistent financial and moral support throughout my academic journey, spanning from undergraduate to Ph.D. studies. Furthermore, my sincere appreciation goes to my late father, Mma, Nathan, Lydia, Bernice (Miss), Ikan, James, Nloguga, Manas, Gomiluk, Ddad, Petra, Evang. Aniekan Johnny and the Church of Christ, Aberdeen. Your support has been invaluable, and your encouragement and prayers have been a source of strength and inspiration to me. To my beloved family — my wife, Ofonime, my son, Unique, and my daughters, Sharon and Valerie — I owe you all a debt of gratitude for your unwavering support and care throughout my Ph.D. journey. Thank you for standing by me, enduring my shortcomings, and contributing to my success. Your legacy will always be cherished. Last but not least, I am profoundly and eternally grateful to God Almighty for His grace in my life, guiding me through the intensive years of my research. Indeed, His divine intervention has been my anchor, and I am truly thankful for His blessings.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis aims to develop a lightweight intrusion detection model for detecting attacks on the Internet of Things (IoT) in Critical Infrastructures (CI). CI refers to infrastructures that, if compromised, could severely impact military and economic security Burns (2019). Recent developments in CI architecture have integrated IoT devices into the network. Given the constant threats and attacks on IT equipment and devices Hossain et al. (2019), a successful breach or attack on one device within the critical infrastructure can have a cascading effect on others, posing significant economic and national security implications. Therefore, this thesis proposes effective approaches for intrusion detection of attacks in such a way that reduces the computational cost on the IoT devices.

This first chapter provides a background for this research work. It also outlines the purpose of the research, problem statement, significance of the study and research questions addressed by this study. Furthermore, it explains how the central question was broken down into sub-questions, each of which is addressed separately. Additionally, the chapter discusses the motivation behind investigating these research questions and highlights the contributions this thesis makes to the field of study. Finally, an outline of the remaining chapters constituting the thesis structure is presented, along with details of the published literature resulting from this research.

1.1 Background

Critical infrastructures are essential to the corporate existence of nation-states, organizations, homes, and humanity. Because of their critical role, prior to recent times, they

were isolated from the internet and corporate networks, and security concerns were mostly related to flaws and errors that occurred during deployment Pliatsios et al. (2020). Interestingly, as technology advances, the architecture of CI has evolved, particularly with the convergence of CI and the corporate network. This advancement has also led to the integration of the IoT into the architecture of CI. The integration has essentially been for the purposes of value addition via data acquisition, surveillance, data monitoring, preventive maintenance, measurement, and a whole lot of other services. More importantly, the main role of the IoT in the CI architecture is the automatic exchange of the information acquired between it and the control systems or other devices without any manual intervention. In fact, Maglaras et al. (2018) opined that the connection of the IoT system to the Supervisory Control and Data Acquisition (SCADA) system is such that the operators using the Human-to-Machine Interface (HMI) can remotely control the devices and issue commands such as opening a valve, setting/adjusting a temperature/ pressure point, or starting/stopping a pump. Therefore, the integration of IoT into CI has been a revolution that has increased system output and efficiency. Nonetheless, it is also important to state that notwithstanding the goodwill that the integration of IoT has brought into the CI systems, it has also evolved into a significant source of cybersecurity threat, highlighting the system's vulnerability to security and privacy concerns. This is largely due to the enabling gulping hole for breaches by both state and non-state actors to launch a slew of attacks on the CI, Simon (2017). In fact, according to Djenna et al. (2021), IoTs are vulnerable to various security issues, resulting in major privacy concerns for end users such as Critical Infrastructures, smart homes, and so on. OConnor et al. (2019) attributes the successful breaches on IoTs to design flaws. Notwithstanding the enhanced productivity, Vargas and Tien (2023) also postulated the enormous security and privacy concerns arising from using the devices in CI and their implications. For instance, Neisse et al. (2017) opined that manufacturers of these devices exclude security and privacy as a priority but are more concerned with satisfying the market and, hence, the attendant vulnerabilities in the devices. According to Choi et al. (2018), security by design was lacking in most cases as manufacturers of the devices are always in competition with one another to flood the markets with ready-made smart devices irrespective of their non-conformity to security standards. Stellios et al. (2018) in their work went a step further by modeling IoT-enabled attack vectors against critical infrastructures on three entities: the attacker, the IoT device, and the target critical infrastructure. In their submission, the authors describe (a) the attacker as one who has the capabilities, technical skills, and required motivation to access the IoT device. (b) They also describe IoT

vulnerabilities as embedded vulnerabilities and network vulnerabilities. (c) The connectivity between the IoT device and the actual target is categorized by direct and indirect connectivity with a critical system. As enunciated by the authors, the three categories are further enhanced by lack of patches that would have cured the design flaws. Patches help to cure vulnerabilities and prevent attacks because they are regularly updated based on new threats. With these and other reasons, it is apparent that a compromised device could further be exploited for secondary attacks, except when the devices are hardened to enable it detect and prevent exploits. Examples abound of attacks due to exploits of IoT devices on CI.

1.2 Definition of problem

The Internet of Things has become an integral component in CI, especially because of the crucial role it plays. However, because IoTs are vulnerable and susceptible to breaches and attacks due to the flaws in their security landscape, they have sometimes been used as a source of attacks on CI. Several successful breaches and attacks have been recorded and these are often aimed at rendering the infrastructures non-responsive to legitimate demands Riggs et al. (2023); Villegas-Ch et al. (2023). The concomitant effect of prolonged disruption or shutdown of CIs has security implications. Against this backdrop, it is imperative to effectively detect cyber intrusions against the IoT because such threats, breaches, and successful intrusions tend to obliterate the IoT and the data it transmits.

1.2.1 Motivation

The Internet of Things has pervaded every strata of human endeavor, and its application and integration into critical infrastructures has gone a long way in enhancing productivity and improving data collection. This research is motivated by the need to address the growing threats and cyber attacks facing IoT devices and critical infrastructures. The threats, breaches, and cyber attacks are occasioned by vulnerabilities inherent in IoT devices such that traditional intrusion detection models are either unsuitable or expensive for resource-constrained IoT devices. Although several studies (Chapter 2) have been conducted with promising results on lightweight intrusion detection systems for IoT devices, some of the studies are either computationally expensive, inapplicable, or unsuitable (see Table 2.1 and Appendix 1). Therefore, buoyed by this apparent need for a more effective and efficient model, I was motivated to undertake several studies leading

to this thesis on how to optimize feature selection approaches to reduce dimensionality and hence achieve an optimized lightweight intrusion detection model for the detection of cyber attacks on the IoT in critical infrastructures.

1.2.2 Purpose of Research

The purpose of this research is to develop an effective and efficient lightweight intrusion detection model using feature selection approaches as a tool to achieve dimensionality reduction. To underscore the importance of this research, consideration is given to current studies and the limits to their effectiveness in detecting intrusions in the IoT based on both traditional machine learning and deep learning approaches. This is with a view to splitting the research project into smaller studies, with each chapter used to answer the fundamental questions outlined in this thesis. To achieve the aim of this research, the objectives below are the targets to ensure effective implementation of the proposed model.

- Obtain relevant IoT-related cyber attack datasets from benchmark sources. The class types of the dataset will be ensured that they correlate with the vulnerabilities and attack types indicated in section 2.5.
- Analyze the datasets using relevant tools to identify the patterns and signatures of attacks.
- Apply traditional Machine Learning and Deep Learning algorithms, including other relevant methods, to effectively identify and classify the relevant data.
- Deploy the model in an IoT domain with a view to validate the model through inferencing.

1.2.3 Research Questions

In many cases, IoT devices are small and lightweight, and the limited dimensions, therefore, place some hardware restrictions on them in terms of memory, energy consumption, communication abilities, and computational abilities. These restrictions, coupled with the inherent vulnerabilities in the IoT devices and the ease with which they could be attacked or co-opted into attacking the CI, make the central research question *How can an effective, efficient and optimized lightweight intrusion detection model for the IoT be achieved?* inevitable. This central research question is further broken down into the following research questions:

1. **RQ1:** - How can effective generalization be achieved in order to improve intrusion detection of attacks on IoT devices?

2. **RQ2:** - In what ways can dimension reduction techniques be employed to obtain optimal features for constructing a lightweight intrusion detection model for resource-constrained IoT devices?

3. **RQ3:** - In the realm of IoT security, how can the strategies outlined in RQ2 be optimized to bolster resilience, efficiency, and overall performance of the intrusion detection model?

1.3 Contribution to the Field

In this thesis, an effective and computationally efficient lightweight intrusion detection model against cyberattacks on the IoT is proposed. The main contributions of this thesis are as follows:

1. Development of a data augmentation strategy based on Sort, Augment, and Combine (SAC) approach. This approach was used both for the oversampling of minority classes and for creating synthetic data to augment the entire dataset. The data augmentation approaches are described in Section 3.4 and Chapter 5 of this thesis.

2. Development of a Common Features Technique (CFT) for the selection of optimal / non-redundant features that are adaptable to several learning algorithms. The technique helps in dimensionality reduction and a description of its application is provided in Section 3.3 and Chapter 6 of this thesis.

3. Development of a deep learning optimized lightweight intrusion detection model for detecting cyberattacks against the IoT. The proposed model is computationally efficient and inexpensive in terms of computation time and memory usage. In addition, the model was made resilient against adversarial attacks. (Chapter 7)

1.3.1 Significance of Study

This study's significance lies in its potential applications and implementations in the realm of IoT for efficient intrusion detection of attacks. By doing so, the study aims to contribute to the reduction of cyber-attacks via IoTs on critical infrastructures enabled by IoT technologies.

1.4 Thesis Structure

This thesis consists of eight chapters, with the first being this introduction. The remaining chapters are organised as follows:

1.4.1 Chapter 2 Background and Related Work

Provides an overview of what constitutes a critical infrastructure, the role of IoT in critical infrastructures, IoT vulnerabilities, and cyberattacks. The chapter also discusses issues relating to intrusion detection and expounds on literature and related studies on lightweight intrusion detection-related work based on feature selection approaches, machine learning approaches, deep learning approaches, fog/cloud-based approaches, and other approaches. Finally, gaps in the related studies stemming from the comparison of the outputs of the studies are also discussed in this chapter.

1.4.2 Chapter 3 Methodology

This chapter introduces the research methods used throughout this thesis to answer the research questions. It starts with the identification of the right data through feature selection, data compression, data augmentation, pruning & deparameterization, shallow deep learning, bayesian optimization, adversarial attack, quantization, deployment, and inferencing. In addition, it provides detailed ethical practices relating to the methodology used in the course of performing the experiments.

1.4.3 Chapter 4 Effective detection of cyberattacks in a cyber-physical power grid system

This chapter explores IoT integration into the power grid and its attendant security concerns. This chapter addresses attacks on intelligent electronic devices (IEDs) in smart grid networks. Building on the concerns raised in Chapter 2. It also provides an explanation of the use of machine learning algorithms to accurately detect and classify attacks on IoT devices. Furthermore, with the application of several machine learning algorithms, the chapter provides an explanation of how the data was analyzed and evaluated and then goes on to explain how the best model was eventually selected.

1.4.4 Chapter 5 Improving Intrusion Detection Through Training Data Augmentation

This study examines the inherent challenge of imbalanced class distribution within the dataset, which explains ineffective generalization and, thus, ineffective classification. This chapter refocuses on achieving computational efficiency by employing robust data augmentation and feature selection methodologies. These strategies bolster data generalization and classification, particularly in scenarios involving imbalanced datasets.

1.4.5 Chapter 6 Ensemble Common Features Technique for Lightweight Intrusion Detection in Industrial Control System

Explores the use of three feature selection techniques to identify common features that can effectively reduce the dimensionality of data while also enhancing classification accuracy at minimal computational costs. In addition, the chapter introduces multiple learning algorithms, recognizing that relying solely on a single feature selection technique and learning algorithm may not yield the most effective classification outcomes. The ensemble feature selection techniques, adoption of various learning algorithms, and use of several datasets, including the results of the study, are recorded and discussed in this thesis.

1.4.6 Chapter 7 Optimized Common Features Selection & Deep-Autoencoder (OCFSDA) For Lightweight Intrusion Detection in the Internet of Things

This chapter presents the final study in this thesis. With the previous chapter exploring the use of common feature selection technique (CFT) for the reduction of computation cost, further dimensionality reduction of the CFT data was delved into through feature extraction and optimization processes in this chapter. The process involves pruning, deparameterization, resilience, quantization, and inferencing, all of which were aimed at achieving a computationally efficient and lightweight intrusion detection model tailored for the Internet of Things (IoT). The results of the study are also presented and discussed.

1.4.7 Chapter 8 Conclusion

Reviews the materials presented in the previous chapters and provides a summary of the results and discussions. This chapter also discusses the limitations of the research and makes useful suggestions on how the study could be further extended.

Chapter 2

Background and Related Work

In this chapter, relevant studies are meticulously examined, particularly in terms of their alignment with the research objectives of this thesis. The exploration begins with an analysis of critical infrastructure, underscoring the pivotal role of the IoT within it. Subsequently, the prevailing security challenges and the diverse spectrum of attacks encountered are discussed. Following this, various intrusion detection methodologies are scrutinized, with a specific emphasis on lightweight intrusion detection techniques, thereby highlighting the gaps in the existing literature.

2.1 Critical Infrastructures

Several definitions have been ascribed to critical infrastructure, largely due to the importance that the country attaches to it or the associated threats. The United States, for example, has 16 critical sectors, which, according to Home Land Security, refer to as infrastructures that, if compromised, could severely impact military and economic security Act (2001); Andrew et al. (2020); Lewis (2019). Similarly, the UK has 13 critical infrastructure sectors, which are defined as facilities, systems, sites, information, people, networks, and processes necessary for the country to function and upon which daily life depends Cox et al. (2022); UK (2017). Accordingly, the EU Commission defines critical infrastructure as physical resources, services, information technology facilities, networks, and infrastructure assets that, if disrupted or destroyed, would have a serious implication on the health, safety, security, or economic well-being of citizens or the effective functioning of governments Markopoulou and Papakonstantinou (2021). From these three definitions, it can be inferred that a successful breach or attack on one device within

the critical infrastructure is likely to have a cascading effect on others, thereby posing significant economic and national security implications. Interestingly, these critical infrastructures were isolated from other networks but with advancement in technology and developments, critical infrastructure architecture have improved resulting in its integration with information technology and IoT into its network. The integration has therefore exposed the critical infrastructures to a myriad of cyber-attacks Markopoulou and Papakonstantinou (2021).

2.2 Internet of Things in Critical Infrastructures

IoT devices have been fully integrated into the CI framework Viganò et al. (2020), and it is instructive to say that since its inception and integration, it has transformed several boundaries of internet technology. This is because of the value additions that it brings; be it through the energy sector, "smart homes", home automation, network-enabled medical devices, intelligent vehicles, smart roads and sensor bridges, industrial and energy production systems, smart grid, revolutionized health care system, transportation, supply chain, and others. Simon (2017). IoT devices offer several services, including data collection, preventive maintenance, pressure and temperature monitoring. Some examples of IoT used in critical infrastructures are NORBIT Aptomar Torsæter (2019), which is used for oil spill detection, vessel collision avoidance and environmental observation of birds and mammals, CargoSense solution, which includes sensors that can be used in shipments to track temperature, humidity, and pressure every five minutes Cil et al. (2022). Other notable solutions in this realm include Tachyus Sensors and Wzzard Wireless Sensors Dall'Ora et al. (2019); JPT staff (2019). While Tachyus Sensors are cutting-edge technologies tailored for the oil and gas sector, facilitating the monitoring and analysis of extraction processes, the Wzzard sensors, on the other hand, monitor a range of parameters such as temperature, flow, vibration, and levels in equipment and tanks. These solutions empower companies to optimize their operations, enhancing output and efficiency.

These devices have indeed revolutionized the acquisition, processing, and use of data. Interestingly, with the evolution and incorporation of wireless communications, these devices are connected through the Internet to other CI equipment networks to ensure ease of interconnection and data transfer. However, while the adoption and integration of the IoT into the CI for efficiency and a myriad of other benefits accrue from it, there are also some concerns about security and privacy issues. According to Stellios et al. (2018),

with the rapid expansion of the Internet of Things (IoT), it is not unexpected that numerous recent cyberattacks leverage IoT devices. Attackers often exploit vulnerable IoT technology as an initial step to compromise connected critical systems. Therefore, what seems like an improvement in efficiency through the connection of the IoT has also accelerated the susceptibility of the CI to security violations through IoT-related vulnerabilities Simon (2017).

2.3 Internet of Things Vulnerabilities

The widespread adoption of IoT, coupled with its processing of vast data volumes, renders it an attractive target for non-state actors, raising significant privacy concerns. The inherent security vulnerabilities are exacerbated by design failures from the outset, as security considerations were often neglected during manufacturing due to market pressures and competition Choi et al. (2018); Neisse et al. (2017). This design oversight leaves IoT devices susceptible to a variety of cyber-attacks, with intrusion detection seldom prioritized by manufacturers. Compounding these challenges is the resource-constrained nature of IoT devices, which limits the deployment of traditional intrusion detection systems. The devices exhibit constraints in processing power, storage capacity, and power consumption. Additionally, a prevalent vulnerability lies in the lack of user control over IoT devices, a phenomenon known as mini-control. Despite numerous inherent security issues, users typically have limited control over these devices, exposing connected systems to various malicious attacks Alladi et al. (2020); Hassija et al. (2019). Legacy issues further compound IoT vulnerabilities stemming from the absence of firmware updates and patches. Many off-the-shelf devices suffer from easily guessable login credentials, posing significant security risks. Privacy concerns among end-users in critical sectors, such as infrastructures and smart homes, are escalating Tyagi et al. (2020). The adaptability of IoT systems to seamlessly integrate with other systems exacerbates these concerns. The Message Queue Telemetry Transport (MQTT) protocol, notable for its lightweight publish-and-subscribe connectivity, operates on resource-constrained devices like low-power embedded sensors. However, MQTT lacks an inherent security layer, necessitating users to address security issues themselves. Moreover, these devices face constraints in size and weight, limiting hardware capabilities, including storage, energy reserves, computational power, and communication technologies.

For instance, IoTs are often designed with low power consumption in mind to prolong

their battery life. The design often involves optimizing hardware components and software algorithms to minimize energy usage. However, an IoT under attack will likely have its battery drained in no distant time.

These constraints hinder the integration of highly secure code, relying on power sources vulnerable to attacks. This susceptibility can lead to energy depletion and potential service interruptions for these IoT devices Friedman (2018).

Exploiting the vulnerabilities present, malicious actors have launched a multitude of cyber-attacks on critical infrastructures, marking a substantial increase in cyber threats with growing complexity, as noted by Markopoulou and Papakonstantinou (2021). Protecting these devices becomes paramount, given their integral role in safeguarding critical infrastructure (CI) networks. Several factors contribute to these vulnerabilities, with a prominent issue being the lack of a comprehensive compliance and regulatory framework. In the intricate landscape of the IoT ecosystem, device manufacturers often neglect compliance due to the heterogeneous nature of the IoT architecture. This complexity makes achieving standard software compliance and seamless integration with interconnected applications challenging Bicaku et al. (2020); Zheng et al. (2022).

To address these challenges, the U.S. introduced a regulatory framework aimed at mitigating IoT vulnerabilities. The IoT Cybersecurity Enhancement Act of 2017 mandates that any IoT product procured by the government must adhere to minimum security standards. The legislation further stipulates that vendors ensure that IoT devices are not only patchable but can be authenticated or certified promptly, lack default passwords, and are free of vulnerabilities Kumar et al. (2019); O'Hara (2019); Schneier (2017); Sha et al. (2018); Tewari and Gupta (2020). Despite these efforts, IoTs continue to be exploited, posing a persistent threat to the uninterrupted functioning of critical infrastructures. Several reasons could be attributed to this, and one of them is a lack of regulatory authority to provide guidelines and standards and enforce compliance. The absence of this has led to different manufacturers producing sub-optimal products and non-conforming IoT devices.

2.4 Internet of Things Challenges

The vulnerabilities inherent in the Internet of Things (IoT) have compounded existing challenges, rendering the IoT network itself a formidable obstacle. These challenges can be succinctly outlined as follows:

2.4.1 Energy consumption

Because of the size of the IoT, energy consumption has become one of the main challenging constraints such that a successful cyber-attack on an IoT device have the potential to make the IoT device consume more energy thus draining its battery (if battery powered) in a faster manner.

2.4.2 Heterogeneity

The ubiquity of the IoT brings with it a variety of devices belonging to various families. Some are sensors, actuators, or smart appliances which run on different circuitry. They also use diverse protocols for communications and so the ubiquity makes it harder to easily patch going by its proprietary software Butun et al. (2019b).

2.4.3 Communications

IoT devices use different technologies just as other devices, and their communication could be wired or wireless communications, such as Bluetooth, ZigBee, and LPWAN. Interestingly, each of these technologies comes with its own challenges.

2.4.4 Data Privacy

The confidentiality of the data captured or monitored by IoT may raise concerns in certain scenarios. For example, during standard operation, a device's location and health could be determined by the network administrators or neighboring devices when requested. However, when the devices are in private mode, they should be able to withhold their location information in order to safeguard their (IoT) privacy Butun and Gidlund (2019); Butun et al. (2019a). Therefore, the inability of IoT devices to provide some security with regard to privacy highlights the challenges posed by IoT vulnerabilities.

2.4.5 Self-awareness

IoT devices ought to autonomously self-organize, executing predefined tasks in response to real-world environmental conditions with minimal human intervention; however, because they are not scalable, they become less amenable to changes, including attacks Butun et al. (2019b).

2.5 Attacks on IoT and IoT-enabled Critical Infrastructure

Numerous instances of IoT breaches and successful cyber-attacks on IoT-enabled Critical Infrastructures have been documented. Typically, attackers target the vulnerabilities present in IoT devices as the initial step towards compromising either the IoT devices themselves or the interconnected critical systems. These attacks are often focused on undermining the system by overwhelming it until it becomes unresponsive to legitimate requests or disrupting the operations of the IoT devices or the interconnected CI network Protogerou et al. (2021). Sometimes, multiple IoT devices are hijacked and utilized as bots to execute DDoS attacks on other networks. Consider the threat model in Figure 2.1, where the IoT monitors and transmits data to the SCADA system. However, attackers could also attack the IoT devices and manipulate the data. The attacker could also co-opt the devices as bots to attack the CI SCADA system and render it non-responsive. The following is a list of successful cyber-attacks targeting critical infrastructures and IoT devices:

2.5.1 Denial of Service (DoS) attacks:

A DoS attack targeting an IoT system seeks to inundate the device or network infrastructure with an excessive volume of traffic, rendering it unresponsive to legitimate services. This occurs when an attacker inundates the system's communication infrastructure, either at the network or application layer, with an overwhelming number of superfluous connection requests. Due to resource limitations, IoT devices are more vulnerable to such attacks and are less equipped to handle the excessive traffic NCSC (2020).

2.5.2 Distributed Denial of Service (DDoS) attacks

DDoS attacks are similar to the DoS attack. However, in this case, the attacker enlists the help of several systems or devices (bots) to create a large botnet army, with each bot generating a number of requests which add together to overwhelm the target critical infrastructures and make it non-responsive Bertino and Islam (2017); Cvitić et al. (2021). Examples of IoT-enabled DDoS attacks abound; for example, in October 2016, a significant attack occurred, orchestrated by approximately 150,000 compromised IoT devices under the control of the Mirai botnet. This attack unleashed an unprecedented DDoS assault with a traffic strength of 1.2 Tbps, a magnitude 40 to 50 times larger than typical attacks. As a result, access to services such as Amazon, Netflix, PayPal, SoundCloud, Twitter, and others was temporarily disrupted for several hours due to the overwhelming

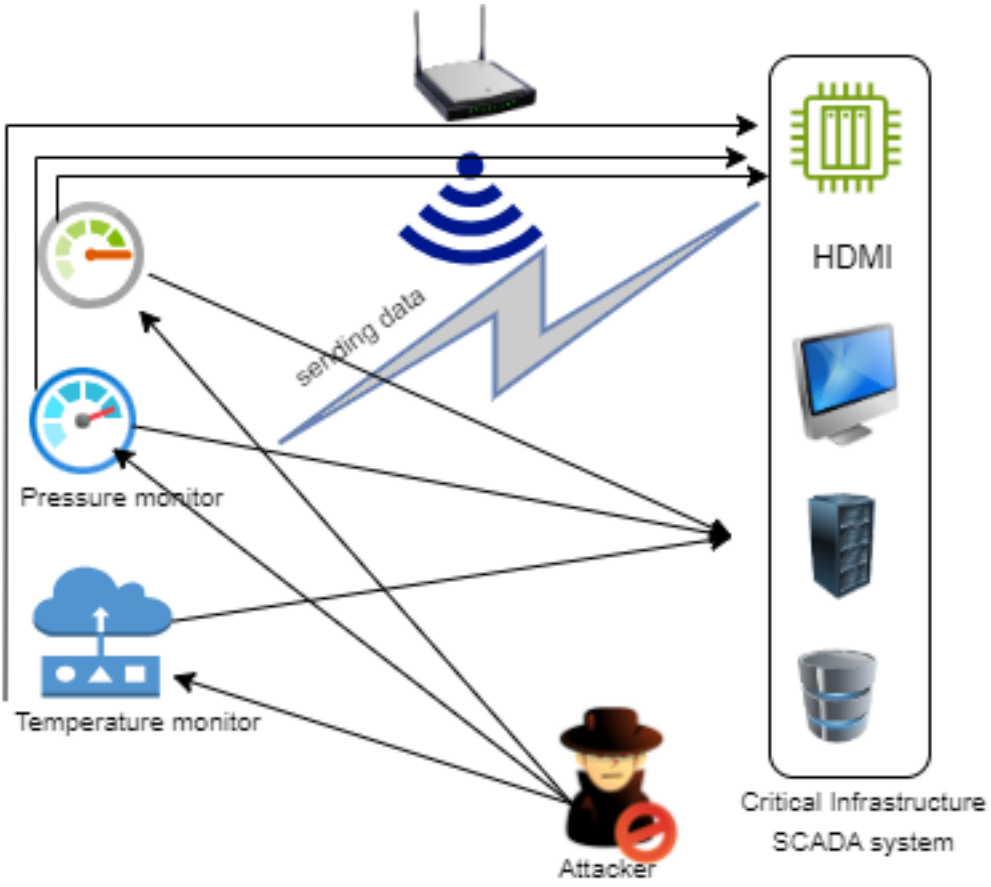


Figure 2.1: Threat model of attacks from

influx of traffic. Pishva (2017)

2.5.3 Malware Injection

Malware injection is an attack where malicious software is installed on devices, allowing it to hijack and spread to other devices. These attacks typically exploit unpatched vulnerabilities in smart IoT devices, enabling them to propagate and cause harm to other enterprise applications. In some cases, these compromised devices are then utilized as bots for DDoS attacks. A well-known example of malware is WannaCry ransomware, which encrypts files and essential services, thereby preventing legitimate users until a ransom is paid Ngo et al. (2020); Resul and Gündüz (2020). This form of attack poses significant risks to the security and functionality of IoT systems.

2.5.4 SQL injection attacks

An IoT in critical infrastructure is usually data-driven based, and so, a SQL injection attack is aimed at stealing, altering or deleting the data content, thus leading to data integrity concerns Damghani et al. (2019); Gowtham and Pramod (2021). Attackers may execute SQL queries to access data contained in the edge devices or database. Interestingly, almost all of the IoT-enabled critical infrastructures have databases and during transmission of data from sensors, the data could be captured, modified, and forwarded to the intended database.

2.5.5 Reconnaissance

This is a form of attack in which the attacker gathers information about the target either covertly or otherwise Rodofile et al. (2017). With effective reconnaissance, attackers can obtain critical information about the target devices or network, including vulnerabilities (both hardware and software) that could be exploited. Interestingly, the peculiarity of IoT and their vulnerabilities makes them a soft target for exploitation to target other devices in the network.

2.5.6 Privilege Escalation

Privilege Escalation involves the attacker obtaining an admin's administrative credentials, allowing the attacker to access more resources in the IoT or IoT-based system. To achieve this, the attacker tries to obtain administrative access by using password cracking and harvesting techniques, which are then used to carry out this attack Yu et al.

(2021). Another classical example is the attack on the German Steel maker, Bundesamt für Sicherheit in der Informationstechnik (BSI) Lee et al. (2014a) where attackers gained administrative privileges and cause severe damage to furnace equipment. It is thought that the attack impacted the PLCs, HMIs, SIS controllers, and the alarm systems.

2.5.7 Power System and IoT-Enabled Smart Grid Attacks

The power system has undergone a transformation, becoming smarter through the integration of information technology and advanced communications. This development has been facilitated by the introduction of robots and Electronic Intelligent Devices (EID), which aim to enhance system efficiency, energy management, reliability, resilience, and the adoption of innovative technologies Çolak and Irmak (2023). However, the implementation of an IoT-based smart grid, which encompasses numerous interconnected nodes, presents an expanded attack surface for potential cyber-attacks on critical infrastructure. Similar to other vital systems, the smart grid has been the target of various attacks. One notable incident occurred in March 2018 when a cyber-attack targeted a nuclear power plant in the United States Khoei et al. (2022); Smith (2018). Interestingly, for purposes of reputation, the particular type of attack is hardly made public by the affected country or organization other than saying that the system has been attacked. A classic example of one such attack was the cyber-attack against the Ukrainian smart grid in December 2015, where attackers successfully shut down approximately 30 substations, resulting in a widespread blackout that left around 230,000 people without electricity Khoei et al. (2022). The sheer scale of a smart grid network means that the potential points of attack are vast. Compromising a single device could potentially render the entire grid susceptible to cyber-attacks. Even when the infrastructure is considered relatively secure, the smart devices utilized within the system remain at risk. This risk presents the possibility of a cascade effect, leading to a complete shutdown of the electricity grid if appropriate protection measures are not in place. Another unexplained attack, as captured by Gorman (2009), was the attack on U.S. power grid utilities in 2009, where it is believed that state-sponsored actors allegedly performed a reconnaissance attack by gathering information about the infrastructure of the power system. Unfortunately, the extent of the attack was never realised in the public domain. Furthermore, Gunduz and Das (2020) in their survey highlighted some of the smart grid threats and how the threats transform into attacks. The authors then went further to categorise the different types of attack types and the various countermeasures.

2.5.8 Oil & Gas and Petrochemical Industry attacks

An attempted cyber-attack against a petrochemical plant aimed to disrupt operations and trigger a potentially fatal explosion. Thankfully, a flaw in the attackers' source code prevented the explosion from occurring. In other words, a mistake within the attackers' source code was the sole reason the catastrophe was averted Iaiani et al. (2021); Stoddart (2022). Furthermore, this particular source code had not been identified in any previous cyber-attacks. All the hacking tools employed in this IoT-based assault were custom-built. Similarly, an oil and gas firm was attacked and in the attack, a smart camera was reportedly exploited as a means to trigger a catastrophic explosion of the Baku-Tbilisi-Ceyhan (BTC) Turkey pipeline in 2008. The attacker allegedly gained unauthorized access through the compromised camera to the industrial control system responsible for managing the pipeline network, specifically the Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs). They manipulated the valve stations with alarming precision, resetting the pressure levels. However, their malicious actions did not stop there. They also deactivated the alarm system and maliciously increased the pressure of crude oil flowing through the pipeline beyond safe limits, ultimately resulting in the horrific explosion and the subsequent spilling of over 30,000 barrels of oil. Di Pinto et al. (2018); Miller et al. (2021).

In a related development, in 2021, a Colonial pipeline supplying oil on the US East Coast was attacked via remote access, leading to the shutting down of the pipeline. The impact was not only restricted to the company, but it also affected the citizens as they suffered a temporary shortage of petrol while the attackers got their wish through the ransom payment Beerman et al. (2023). The real implications have always been the cybersecurity vulnerabilities that might affect other parts of the critical infrastructure. Furthermore, In 2012, a group of attackers which called itself "Cutting Sword of Justice" carried out an attack on the Saudi State Oil company, Saudi Aramco and in a matter of hours, about 35,000 computers were partially wiped out or totally made unusable Simon (2017). Another case of a cyber-attack against the Oil industry was the malware attack on Gazprom in 1994, in which the attackers took control of the gas flow control system for a number of hours, rendering it unusable Lobo (2018). Similarly, the Venezuelan oil company PDVSA reported about the attack on their production system, leading to the reduction of production by up to 87.6% Stergiopoulos et al. (2020). In another development, a leak-detection device was rendered inoperable after it was impaired for several hours, thereby impeding the monitoring of three oil derricks pipelines in Southern California in 2019 Lobo (2018). In 2012, VIA A PLC-actuator, a ballast control of an

oil rig, was manipulated, therefore causing the rig to tilt by 17 degrees Lobo (2018); Stergiopoulos et al. (2020).

2.5.9 Nuclear infrastructure attacks

A classical example of such an attack is the Stuxnet cyber-attack, which specifically aimed to sabotage the Iranian nuclear program by compromising and disabling uranium enrichment centrifuges at the nuclear facility. Stuxnet, a malicious worm, exploited default passwords provided by Siemens to gain access to the Windows operating systems responsible for running the WinCC and programmable logic controller (PLC) programs that managed the industrial plants. What made Stuxnet particularly remarkable was its ability to target and reprogram the intended systems Mohee (2022). It is worth noting that the impact of Stuxnet extended beyond the Iranian nuclear plant, affecting several countries in Europe and Asia as well.

2.5.10 Water Distribution System Hacking

The water distribution system, like other critical infrastructures, operates as a Cyber-Physical System. It incorporates a range of sensors that measure essential process variables such as temperature, pressure, flow rate, and water level. These sensors collect data from the physical system and transmit it to the controllers for analysis and control Adepu et al. (2020). Unfortunately, the water distribution system has experienced various cyber-attacks specifically designed to disrupt its normal functioning. One notable example is the Kemuri Water Company attack. In this incident, the attacker successfully accessed the system and potentially had the ability to manipulate crucial elements. They could have, for instance, raised the water alarm level or unauthorizedly adjusted the valves. Fortunately, these malicious activities were eventually discovered, albeit with some delay Nagpal et al. (2023).

2.5.11 Dam Cyber-attacks

Dams play a vital role in various aspects, including irrigation, electricity generation, and clean water supply. However, they are vulnerable to cyber-attacks. An example is the Bowman Avenue Dam Adamo et al. (2021), where an attacker managed to breach the SCADA system. This intrusion allowed the attacker to conduct reconnaissance activities, gathering sensitive information about the dam's operations, such as water levels, temperatures, and device status. The consequences of such an attack could be severe, including manipulation of water flow, altering chemical dosages in water treatment, or

even opening the floodgates. Another significant event illustrating cyber-attacks' impact on IoT-enabled critical infrastructure occurred in Venezuela. Following a cyber attack on the Guri Dam, the country's primary electricity generator, a prolonged blackout affected the nation for several days. The Guri Dam was the location of the Simon Bolivar Hydroelectric Plant Vaz (2019).

2.5.12 Healthcare System Cyber-attacks

IoT-enabled healthcare infrastructure plays a crucial role in our society, but any disruption to its functionality can have catastrophic consequences. For example, IoT devices used in the healthcare sector are responsible for monitoring patients' vital signs in real-time. However, if these devices were to fail, it would severely impact the collection of vital data such as blood pressure, blood sugar levels, heart rate, and more Djenna and Saïdouni (2018). Another alarming incident involved a cyber-attack on a healthcare facility, where attackers gained unauthorized access through compromised vendor login credentials. Subsequently, they implemented the SamSam ransomware technique to specifically target the facility's servers and encrypt critical files, causing significant disruptions and potential harm Coventry and Branley (2018).

2.5.13 Train System Hacking

The process of gaining unauthorized access to a system, commonly known as hacking, revolves around obtaining the system's password. Hackers employ various methods such as brute force, man-in-the-middle (MITM) attacks, and social engineering to accomplish this. The vulnerability of IoT passwords lies in the fact that they are often default or simplistic text-based passwords, making them easy to crack. These passwords are usually weak and susceptible to being seen or guessed with minimal effort Nam et al. (2020). An instance where IoT vulnerabilities were exploited is the adoption of Industry 4.0 in the train industry, where IoT technologies were implemented to address issues like train failures, train station security, and communication in remote locations Laiton-Bonadiez et al. (2022). However, this advancement also brought risks, as demonstrated by a teenager who hacked the tram system in Lodz, Poland, using a homemade transmitter. This cyber-kinetic attack resulted in trains being redirected, causing injuries to 12 individuals Resul and Gündüz (2020). Similarly, a light rail system in San Francisco fell victim to a ransomware attack. The incident was triggered when an employee unknowingly clicked on a malicious link in a phishing email Genç et al. (2017).

2.6 Intrusion Detection

In 1980, the concept of intrusion was first introduced by Anderson (1980), defining intrusion as any deliberate unauthorized attempt to access, manipulate information, or compromise a system's integrity and reliability. Similarly, Heady et al. (1990) characterized intrusion as actions that compromise computing resources' confidentiality, integrity, or availability. Despite the persistent reality of system vulnerabilities, swift detection and response to breaches are paramount. This is where an Intrusion Detection System (IDS) plays a crucial role, acting as a second line of defence by alerting to unusual behaviors Pradhan et al. (2020).

Intrusion detection systems are generally classified into anomaly detection and misuse (or signature) detection. Anomaly detection identifies intrusions by detecting deviations from normal behavior, creating a profile of a network or host in a non-attacked state. Any deviation from this established norm is considered an attack. On the other hand, misuse or signature detection matches traffic patterns against known attack patterns. Although both methods have their strengths and weaknesses, anomaly detection is often deemed more effective in detecting novel attacks. Notably, anomaly detection relies on data-driven approaches, and given the nature of IoT device data, it is crucial to employ effective means for detecting anomalies in IoT network traffic Elmubark et al. (2019); Manokaran and Vairavel (2022); Sen and Mehtab (2020a,b). Interestingly, data-driven approaches often involve intricate patterns that require advanced scientific approaches to uncover hidden patterns and build models that gain insights from extensive datasets. Discovering these patterns involves using Machine Learning (ML), Neural Networks (NN), and even data augmentation for effective generalization, which is an approach where the learning algorithm is able to effectively learn the patterns in the dataset and then use it to classify new and unseen data more effectively. However, because of the resource-constrained nature of the IoT, these classical approaches are expensive and impractical for deployment on IoT devices for intrusion detection. This underscores the importance of lightweight intrusion detection Manokaran and Vairavel (2022); Sharma et al. (2019).

2.7 Lightweight Intrusion Detection of attacks on Internet of Things

In recent years, there has been a growing research focus on utilizing Artificial Intelligence (AI) methodologies, such as Machine Learning (ML) and Deep Learning (DL), to detect

attacks on various devices. These AI-based approaches have shown great effectiveness in identifying intrusions, leading to significant advancements in Intrusion Detection Systems (IDS) Foley et al. (2020). AI-based IDS provides robust detection and prevention of malicious activities, offering an additional layer of defence against unauthorized intrusions. It is most suitable as a Host-based IDS because it can analyze data based on system baselines and identify anomalies. However, these IDS are not well-suited for the unique characteristics of IoT devices, as highlighted by Roy et al. (2022). IoT and embedded systems face limitations regarding hardware resources such as Random Access Memory (RAM) and flash memory Zakariyya et al. (2023). These constraints hinder the effective implementation of robust security measures. For example, the computational costs associated with data analysis can overwhelm IoT devices. Additionally, the presence of redundant features in datasets can impact the overall classification outcomes. Unfortunately, these approaches require substantial computational resources, making their implementation on resource-constrained devices like IoT challenging. To address the challenges posed by the ineffective implementation of both traditional and AI-based intrusion detection systems, several authors have proposed a number of intrusion detection approaches for detecting attacks on IoT and similar resource-constrained devices. This has become necessary on account of the computational limitations of the IoT hardware.

2.7.1 Lightweight IDS based on feature selection approaches

Feature selection, as earlier stated, is an approach to reduce the dimensionality of data. There are so many approaches to it as a way of achieving a lightweight intrusion detection model. Some of the proposals include:

A lightweight Intrusion Detection System based on filter selection was proposed by Fatima et al. (2023). The paper introduces a lightweight Intrusion Detection System (Li-IDS) designed for IoT environments, addressing security concerns arising from the widespread use of smart devices. The model employs a filter selection approach, utilizing SelectKBest with chi-square feature ranking to identify the most relevant features. It then evaluates these features using six Machine Learning (ML) models on the TON-IoT dataset. The evaluation metrics, including accuracy, False Negative Rate (FNR), False Positive Rate (FPR), training and testing time, as well as CPU and memory usage, demonstrate that the proposed Li-IDS is lightweight, adaptive, and efficient, making it suitable for deployment in resource-limited IoT systems. From the output of the proposals, it is obvious that some of the models did not yield promising results, which could be attributed to the problem of sub-optimal features for some models. Similarly,

using three feature selection approaches, Quincozes et al. (2023) in their study extensively assessed the efficacy of both supervised and unsupervised Machine Learning (ML) techniques in detecting various DoS attacks, such as Sync flooding, Grayhole, and other forms of attacks. Using real-world data from a Wireless Sensor Network (WSN)-based dataset, the evaluation of the approach was carried out, and performance metrics such as accuracy, recall, precision, F1-Score, and processing time were employed for a comprehensive comparison. The findings revealed that supervised ML algorithms, particularly the REPTree algorithm, achieved an F1-Score of 95.69% averaging a processing time of 0.931 seconds per sample classification. In a related vein, Wang et al. (2019) proposed a Sort Aggregation Ensemble Feature Selection (SA-EFS) method designed for classification tasks in high-dimensional datasets. The study tackles the challenge of redundant and irrelevant features through the combination of outcomes from three feature selection methods—chi-square test, maximum information coefficient, and XGBoost—utilizing a sorting aggregation strategy. Subsequently, it assesses the influence of aggregation strategies, specifically arithmetic mean and geometric mean, on the model's overall integration performance. Three classifiers are employed to assess classification and prediction: KNN, Random Forest, and XGBoost. The experimental results demonstrate that the arithmetic mean aggregation in SA-EFS significantly enhances classification accuracy compared to individual feature selection methods, with a recommended threshold interval setting of 0.1. Interestingly, while the SA-EFS approach offers advantages, such as enhanced overall accuracy, there are potential disadvantages that need to be considered, as using sort aggregation introduces additional complexity, which could lead to a higher computational cost of employing multiple feature selection techniques and aggregating their results. Second, the effectiveness of SA-EFS relies on the performance of the three chosen feature selection methods, and it is such that if any of the methods underperform, it could potentially impact the overall effectiveness of SA-EFS.

Additionally, Zhang et al. (2022) proposes an ensemble-based automatic feature selection method called EAFS to address challenges in intrusion detection due to redundant and irrelevant data in network traffic. EAFS dynamically selects features based on their importance, evaluated through an NSOM metric. The method aims to enhance classification accuracy while reducing computational complexity. Experimental results on three large-scale datasets demonstrate the effectiveness of EAFS compared to other recent methods, highlighting its superiority in performance. Furthermore, Alghanam et al. (2023) proposes an enhanced version of pigeon-inspired optimization (PIO) feature selection, incorporating a local search algorithm (LS-PIO), and employs an ensemble learning approach

based on multiple one-class classifiers to bolster the performance of a network intrusion detection system (NIDS). Evaluation on four benchmark datasets—BoT-IoT, UNSW-NB15, NLS-KDD—demonstrates the superiority of the proposed LS-PIO and ensemble-based NIDS over existing techniques documented in the literature, as evidenced by metrics such as F-score, accuracy, AUC, FPR, and TPR. In another study, Das et al. (2021) proposes a machine learning-based approach for network intrusion detection, employing ensemble supervised learning and feature selection techniques. Comparative analysis of various machine learning models and feature selection methods aims to develop a versatile detection mechanism with high accuracy and minimal false positive rates. Experiments conducted on NSL-KDD, UNSW-NB15, and CICIDS2017 datasets demonstrate the effectiveness of the proposed model, achieving a detection rate of 99.3% with a 0.5% false alarm rate. Additionally, Alazzam et al. (2020) proposed a wrapper feature selection algorithm for Intrusion Detection Systems (IDS) that uses a pigeon-inspired optimizer, outperforming several state-of-the-art feature selection methods in terms of accuracy. Furthermore, Zhou et al. (2020) addressed challenges in current intrusion detection algorithms, including redundant data, performance limitations for different attack types, and adaptability to novel attacks. To overcome these issues, the study proposed a framework that combines feature selection and ensemble learning techniques. The framework incorporates a heuristic algorithm called CFS-BA for dimensionality reduction, which selects an optimal feature subset based on feature correlation. It also integrates an ensemble approach by combining C4.5, RF, and Forest by Penalizing Attributes (Forest PA) algorithms. A voting technique is employed to merge probability distributions from these base learners for attack recognition. Experimental results on the NSL-KDD, AWID, and CIC-IDS2017 datasets show improved performance of the proposed CFS-BA-Ensemble method.

In 2018, Acharya and Singh (2018) proposed an Intrusion Detection System (IDS) model that utilized the intelligent water drop (IWD) algorithm for feature selection. The model was evaluated using a Support Vector Machine (SVM) classifier and demonstrated lower false alarm rates and improved accuracy compared to other approaches. However, the study did not specifically test the model's effectiveness in an Internet of Things (IoT) environment or explore its applicability to other models. Another study by Halim et al. (2021) proposed a Generic Algorithm (GA-based) Feature Selection (GbFS) method to address feature selection challenges in network security and intrusion detection. The objective was to enhance classifier accuracy by retaining essential information with a minimal number of features. The research focused on strengthening network security against

cyber-attacks through machine learning, particularly in the development of firewalls and Intrusion Detection Systems (IDSs). The GbFS method underwent parameter tuning and incorporated a fitness function, demonstrating improved accuracy when evaluated on three benchmark network traffic datasets (CIRA-CIC-DOHBrw-2020, UNSW-NB15, and Bot-IoT) compared to standard feature selection methods. Similarly, Zhang et al. (2023) proposed a data-driven Network Intrusion Detection System (NIDS) that integrated Feature Selection and Deep Learning (FS-DL). FS-DL aimed to improve data quality by using methods like standard deviation and association rule mining to eliminate redundant features. It employed a simple neural network structure with three layers and few neurons to balance accuracy and time cost. Experimental results indicated that FS-DL achieved improved detection performance with a few features, reaching an overall accuracy of 91% and 84% for UNSW-NB15 and NSL-KDD datasets. However, the authors mentioned a trade-off between detection accuracy and time cost without providing specific details such as the acceptable accuracy threshold or the impact on real-time detection.

In a related context, Mafarja et al. (2020) addressed the security of IoT environments against intrusions and proposed a novel wrapper feature selection method based on the augmented Whale Optimization Algorithm (WOA) to handle the challenge of high dimensionality. The approach incorporated V-shaped and S-shaped transfer functions into WOA, outperforming other well-known evolutionary optimizers when evaluated using the N-BaIoT dataset, representing real IoT traffic. The study demonstrated that WOA with a V-shaped transfer function and an elitist tournament binarization method excelled in accuracy, fitness, number of features, running time, and convergence curves. These results suggest the potential deployment of the proposed approach in IoT intrusion detection systems. Additionally, researchers Jaw and Wang (2021) presented an approach that addressed irrelevant features and novel attack detection. The method involved a Hybrid Feature Selection (HFS) approach coupled with an ensemble classifier, resulting in reduced model complexity and improved algorithm generalization.

2.7.2 Lightweight IDS based on Machine Learning approaches

Numerous studies advocate for machine learning as a solution for lightweight intrusion detection in IoT devices. For instance, Selim et al. (2021) proposed an anomaly detection method for identifying cyber-attacks in industrial IoT infrastructure, utilizing various algorithms such as Logistic Regression, Linear Discriminant Analysis, and Naïve Bayes. Real-world datasets covering 15 anomaly scenarios were employed, with Classification

and Regression Tree (CART) and Naïve Bayes (NB) yielding an improved accuracy. Similarly, Garmaroodi et al. (2020) introduced an SVM-based approach for intrusion detection in an IoT-enabled water purification plant, integrating supervised learning and artificial neural networks to model system components. Hasan et al. (2019) addressed IoT infrastructure security concerns, evaluating machine learning models like Decision Trees and Random Forests to predict attacks and anomalies, with Decision Trees and Random Forests outperforming other techniques. Manhas and Kotwal (2021) assessed various machine learning techniques for IDS implementation, identifying decision trees as the most effective classifier for distinguishing normal and malicious network connections.

In addition, Kasongo and Sun (2020) introduced a filter-based feature reduction technique using the XGBoost algorithm. Various machine learning models, including SVM, kNN, LR, ANN, and DT, were deployed on the reduced feature space for both binary and multiclass classifications. The XGBoost-based feature selection method notably enhanced the test accuracy of the Decision Tree, elevating it from 88.13% to 90.85% for binary classification. For larger networks such as ISPs and enterprise networks, Kumar and Lim (2019) proposed EDIMA, a distributed modular solution tailored for IoT malware-induced attacks. EDIMA uses machine learning algorithms to classify traffic on edge devices, integrating components such as a packet traffic feature vector database, a policy module, and an optional packet sub-sampling module. The classification efficacy of EDIMA was assessed using testbed experiments. Fenanir et al. (2019) addressed IoT attack challenges in lightweight intrusion detection using feature selection and classification algorithms. Logistic regression, naive Bayes, decision tree, random forest, k-nearest neighbor, support vector machine, and multilayer perceptron were evaluated, with the Decision Tree algorithm demonstrating superior performance across multiple datasets. Davahli et al. (2020) proposed the GABGWO model, combining Genetic Algorithm (GA) and Grey Wolf Optimizer (GWO), to develop a lightweight intrusion detection system (LIDS) based on support vector machines (SVM). The GABGWO algorithm identified relevant traffic features and enhanced LIDS performance regarding the accuracy and computational costs. Abdulla et al. (2023) concentrated on mitigating DoS attacks in IoT networks, acknowledging the potential for IoT devices to serve as launchpads for larger-scale attacks. Through packet analysis and applying four machine learning algorithms, the study achieved a 98% accuracy in detecting IoT DoS attacks. However, these studies do not address class imbalances arising from DoS and DDoS attacks, and the adaptability of models to new and evolving threats remains crucial. Therefore, if the EDIMA proposed by Kumar and Lim (2019) lacks adaptability and requires frequent

updates to address new threats, it may not offer the required long-term security for IoT systems.

Chawla and Thamilarasu (2018) introduced a machine learning-based intrusion detection system that provides security as a service for IoT networks. The study delineated the framework and intricacies of intrusion detection, underscoring the importance of compatibility among diverse network communication protocols in the IoT. In countering DDoS threats in IoT networks, Jan et al. (2019) proposed a lightweight detection approach employing an SVM based on supervised machine learning. Through simulations, the proposed method demonstrated improved performance using a concise feature set, ensuring effective classification and swift detection. Additionally, Priya et al. (2022) proposed a machine learning-based Intrusion Detection System (ML-IDS) tailored for IoT network threats. The study proposes a supervised ML-IDS with a centralized and lightweight architecture. Comparative analysis of various categorization techniques on three datasets revealed that the decision tree algorithm outperforms the others in intrusion detection results. In a related context, Nõmm and Bahşi (2018) investigated using unsupervised learning models with reduced feature sets to minimize computational demands. In addition, they advocated training a single model for all IoT devices instead of individual models for each device, aiming to optimize resource utilization. Leveraging the MQTT protocol, Jaafar et al. (2022a) proposed a lightweight Intrusion Detection System (IDS) tailored specifically for IoT environments. Their method seeks to enhance the efficiency and efficacy of a machine learning-based IDS using a support vector machine, particularly targeting attacks within MQTT-utilizing IoT systems. They applied feature selection techniques to streamline the model's complexity, evaluating the results using multiple metrics. Similarly, Roy et al. (2022) addressed IoT security concerns by proposing an intrusion detection model that utilizes machine learning to identify cyber-attacks and anomalies in resource-constrained IoT networks effectively. Through optimization techniques such as multicollinearity removal, sampling, and dimensionality reduction, the model identifies crucial features for intrusion detection using minimal training data and time. Experimental evaluations of the CICIDS2017 and NSL-KDD datasets demonstrated an improved detection rate with a low false alarm rate. Siddharthan et al. (2022) introduced an intelligent intrusion detection system employing Elite Machine Learning algorithms (EML) for cyber-attack recognition while employing a lightweight protocol to manage time constraints between devices. Their experimental setup involved a testbed with hardware and sensors connected using the MQTT protocol, achieving an average

accuracy of over 99%. While these proposed models show promise, it's crucial to acknowledge that their resilience to emerging cyber-attack types may not be explicitly addressed. For example, measures to enhance the models' robustness against adversarial attacks were overlooked by the authors. Additionally, reducing data features or dimensions doesn't necessarily result in a lightweight model unless tailored for the IoT environment. Interestingly, some of the proposed approaches overlooked this aspect, and the authors omitted information regarding model sizes and computation time, focusing solely on overall accuracy and recall.

2.7.3 Lightweight IDS based on Deep Learning approaches

Reflecting on the criticality of IoT security and the imperative for intrusion detection in IoT networks, Sharma et al. (2024) proposed a Deep Learning (DL) model for intrusion detection utilizing a filter-based method to select pertinent features, thereby reducing feature volume. Two distinct deep learning architectures, Deep Neural Network (DNN) and Convolutional Neural Network (CNN), underwent training and testing using publicly accessible datasets, NSL-KDD and UNSW-NB 15. The DL model exhibited superior accuracy rates for both datasets. Similarly, Utomo and Hsiung (2019) investigated the ramifications of intrusions leading to system failure and advocated for the utilization of Long-Short Term Memory (LSTM) recurrent neural network techniques for data-driven attack detection. Experimental trials were conducted using data from two smart meters over a month, resulting in a training set of 120,000 samples and a testing set of 40,000 samples. Results showcased an accuracy of 92%, a True-Positive Rate of 81%, and a False Positive Rate of 50%. However, the 50% FPR poses a high risk of false alarms, and the authors did not provide information regarding the model size. Also, Mushtaq et al. (2022) addressed the challenges posed by the "curse of dimensionality" and the trade-off between false alarm rate and detection rate in designing an intrusion detection system. They proposed a hybrid framework integrating a Deep Autoencoder (AE) with Long Short Term Memory (LSTM) and Bidirectional LSTM (Bi-LSTM) for intrusion detection. The approach involved feature optimization using AE and classification using LSTMs to discern between normal and anomalous samples. Evaluation of the NSL-KDD dataset demonstrated that the AE-LSTM framework surpassed other deep and shallow machine learning techniques. Specifically, on the NSL-KDD dataset, AE-LSTM achieved a classification accuracy of 89%, a detection rate of 89.84%, and a false alarm rate of 11%, showcasing superior performance compared to recent state-of-the-art techniques.

Similarly, Xu et al. (2020) introduced an IoT intrusion detection system based on an

LSTM autoencoder. This method utilized the LSTM autoencoder to capture time series features and exploit its feature learning capabilities for classification. Experimental assessments validated the model's high accuracy and low false alarm rate. Saba et al. (2022) proposed a model that integrates edge computing and deep learning techniques for IoT intrusion detection. The model employs gated convolution to enhance CNN's performance in detecting anomalies and mitigating DDoS attacks, thereby achieving improved accuracy rates across various DDoS attack types. In a related development, recognizing the constraints of traditional machine learning techniques among escalating cybersecurity threats in IoT networks, Ullah and Mahmoud (2021) proposed a novel anomaly-based IDS using CNN. The CNN model, designed for multiclass classification, is implemented in various formats and validated using diverse intrusion detection datasets. Transfer learning is applied for binary and multiclass classification using a pre-trained CNN model, showcasing improved performance in accuracy, precision, recall, and F1 score.

Additionally, Nguyen et al. (2022) presents Realguard, a Deep Neural Network (DNN)-based network intrusion detection system (NIDS) tailored for direct deployment on local gateways, bolstering security for IoT devices within the IoT ecosystem. Rearguard excels in accurately identifying multiple cyber attacks in real-time while maintaining a minimal computational footprint. The proposed methodology incorporates lightweight feature extraction and an efficient attack detection model driven by deep neural networks. Practical evaluations validate Realguard's prowess in real-time attack detection, achieving an average accuracy of 99.57%, outperforming competitors at 98.85%. Moreover, the solution operates seamlessly on resource-constrained gateways such as Raspberry Pi, processing approximately 10,600 packets per second. Li et al. (2020) proposed a novel deep learning approach for intrusion detection using a multi-convolutional neural network (multi-CNN) fusion technique, showcasing superior performance in identifying unknown intrusions compared to existing methods. The study by Li (2022) focuses on enhancing the effectiveness and precision of intrusion detection systems tailored for the Internet of Things (IoT) using deep learning techniques. They introduced an embedded model (EM) for streamlined feature selection and integrated it with a lightweight intrusion detection model named XCNN to minimize device strain and enhance computational efficiency. In addition, incorporating the Attention mechanism (Self-Attention) mitigates training time and long-term dependency issues. Evaluation of simulation platforms and datasets, including NSL-KDD, CIC-IDS2017, and CSE-CIC-IDS2018, demonstrated improved training efficiency in IoT environments. Furthermore, NG and Selvakumar (2020) addresses IoT vulnerabilities and associated cyberattacks by proposing a solution that

leverages vector convolutional deep learning (VCDL) within a fog-based framework. This distributed approach distributes IoT traffic training and computation across fog nodes, thereby enhancing scalability. According to the authors, experimental results on the UNSW Bot-IoT dataset demonstrate that the distributed deep learning approach can achieve better performance than centralized methods in terms of accuracy, precision, and recall.

Concerned with IoT vulnerabilities, breaches, and the surge of zero-day attacks linked to IoT protocols, Diro and Chilamkurti (2018) advocate for the application of Deep Learning (DL) in cybersecurity. They leverage advancements in CPU technology and neural network algorithms to harness DL's high-level feature extraction capability, which is considered to be resilient against small mutations or novel attacks. The study focuses on employing DL for attack detection in the social Internet of Things, demonstrating its ability over traditional machine learning approaches. Specifically, the deep learning model outperforms its shallow counterparts, and a distributed attack detection system proves more effective than centralized detection systems. Similarly, to address the challenge of identifying hostile behaviors and attacks in IoT networks, Hanafi et al. (2023) introduces a model that combines an Improved Binary Golden Jackal Optimization (IBGJO) algorithm with an LSTM network. The IBGJO algorithm is enhanced through opposition-based learning for feature selection from IDS data, optimizing subset selection to mitigate local optima issues. The IBGJO-LSTM model utilizes LSTM for sample classification, achieving an accuracy of 98.21% on the NSL-KDD and CICIDS2017 datasets. Comparative analysis showcases its superior accuracy compared to other models and traditional methods such as SVM, KNN, and Naive Bayes (NB). In a related development, Rizvi et al. (2022) introduces a 1D-Dilated Causal Neural Network (1D-DCNN) tailored for identifying security breaches in extensive IoT networks. Departing from conventional deep learning methods that require substantial computational resources, the proposed 1D-DCNN leverages dilated convolution with a dilation rate of 2 to counteract information loss stemming from pooling and down-sampling. This enables the model to widen its receptive field, facilitating more comprehensive contextual data gathering. The efficacy of their approach was assessed on the CIC-IDS2017 and CSE-CIC-IDS2018 datasets, which showed better accuracy when compared to existing deep learning methods, according to the authors. Specifically, the proposed model achieves a malicious attack detection rate precision of 99.7% for CIC-IDS2017 and 99.98% for CSE-CIC-IDS2018 in simulation experiments. Abdel-Basset et al. (2021b) presents a semi-supervised deep learning

strategy dubbed SS-Deep-ID, which aims to enhance efficiency, bolster performance robustness, and sustain computational efficiency for real-time intrusion detection. Given the scarcity of labeled records despite the exponential IoT growth, their Semi-Supervised Deep Intrusion Detection (SS-Deep-ID) approach employs a multiscale residual temporal convolutional (MS-Res) module to augment the network's ability to grasp spatiotemporal representations. They introduced an improved traffic attention (TA) mechanism to estimate importance scores, assisting the model in focusing on critical information during the learning process. In addition, a hierarchical semi-supervised training method is implemented, considering the sequential traits of IoT traffic data. SS-Deep-ID is crafted for integration into fog-enabled IoT networks to deliver efficient real-time intrusion detection. Evaluations conducted on the CIC-IDS2017 and CIC-IDS2018 datasets demonstrate that SS-Deep-ID enhances intrusion detection efficiency and robustness while maintaining computational efficiency. Okey et al. (2023) propose a transfer learning Intrusion Detection System (IDS) founded on Convolutional Neural Network (CNN) architecture, leveraging pre-trained CNN models (VGG16, VGG19, Inception, MobileNet, and EfficientNets) trained on the CIC-IDS2017 and CSE-CICIDS2018 datasets. Their model, named efficient-lightweight ensemble transfer learning (ELETL-IDS), amalgamates the three best-performing models (InceptionV3, MobileNetV3Small, and EfficientNetV2B0) using a model averaging approach. Through comprehensive evaluation, ELETL-IDS surpasses existing state-of-the-art IDS proposals, achieving 100% accuracy, precision, recall, and F-score. The study employs Matthew's Correlation Coefficient (MCC) and AUC-ROC metrics, validating the model's reliability. The lightweight and efficient design of ELETL-IDS renders it suitable for deployment in cloud IoT systems for intrusion detection. Wang et al. (2022b) introduce a knowledge distillation model based on a Triplet Convolution Neural Network (TCNN) to tackle anomaly detection challenges in industrial cyber-physical systems (CPS) using resource-constrained IoT devices. Their model employs a robust loss function during training to enhance stability and introduces a novel K-fold cross-training method for improved accuracy. Experimental results on the NSL-KDD and CIC IDS2017 datasets demonstrate significant performance advantages over traditional deep learning approaches and recent state-of-the-art models. Moreover, TCNN achieves an 86% reduction in computational cost and model size with only a 0.4% accuracy loss compared to the original model.

Given the widespread adoption of IoT devices and the frequent security breaches they face, Idrissi et al. (2021) delve into deploying a Deep Learning-Based Host-Intrusion

Detection System (DL-HIDS) on specific commercial IoT devices. Their aim is to optimize DL-HIDS for hardware limitations, including memory consumption and inference timing. The study evaluates various DL-HIDS models to identify the most suitable for each device based on its characteristics. The results show high accuracy (up to 99.74%) and fast inference times across different devices. While the study underscores the potential of customized IDS for each device, it also underscores the necessity of central IDS support in fog or cloud layers for comprehensive IoT security. Notably, the proposed model achieved a compact size of 2.704KB, although details on recall, precision, and F-score were omitted. In addition, apart from utilizing the MQTT IoT-IDS2020 dataset, the authors did not validate their approach using other datasets. In a parallel effort addressing IoT intrusion detection, Boppana and Bagade (2023a) express concern that manufacturers, in striving to meet demand, often overlook producing cost-effective, user-friendly devices, neglecting security considerations. Consequently, they proposed an unsupervised model named GAN-AE, which combines a GAN and an autoencoder to detect unknown intrusions in MQTT IoT applications. Outperforming other unsupervised models such as autoencoder, One-Class SVM (OCSVM), and Isolation Forest (IF), the GAN-AE model achieves an accuracy and F1-Score of 97% on both a custom-built and public MQTT dataset. Furthermore, Ahmad et al. (2023) proposes a novel approach for identifying malicious network traffic. The framework employs a Support Vector Machine (SVM) and Convolutional Neural Network (CNN) with a Gated Recurrent Unit (GRU), fine-tuned using a Slime Mould Algorithm (SMA), achieving accuracies of 98.45% and 94.84%, respectively. Evaluation of the KDD dataset demonstrates the model's efficacy in detecting DDoS attacks with improved efficiency compared to other solutions.

In summary, although the proposed deep learning models have shown optimal performance, it is crucial to acknowledge potential limitations for effective implementation. They can be computationally intensive, unsuitable for resource-constrained IoT devices, and have considerable training times, making them less suitable for real-time intrusion detection. Dependence on large labeled datasets, limited interpretability, and complex parameter tuning pose challenges. Generalization to diverse datasets presents IoT environments in difficulty, and adversarial attacks can undermine effectiveness, leading to increased energy consumption. Robustness concerns are often not addressed. In distributed systems, challenges may impact performance and effectiveness, introducing potential issues. The key challenges include the following:

- **Communication Overhead:** In a distributed system, various components or

nodes must establish communication in order to exchange information. This communication incurs additional costs, such as the time and resources needed to transmit data between nodes. The presence of excessive communication overhead can have a detrimental impact on the overall performance of the system.

- **Synchronization Issues:** It is essential to guarantee the synchronisation and cohesive operation of distributed components. Synchronisation issues can occur when multiple nodes require coordination of their actions or sharing of state information. Insufficient synchronisation can result in discrepancies and impact the precision of the anomaly detection system.
- **Increased Complexity:** Decentralising computation across multiple nodes frequently results in heightened intricacy compared to a centralised system. Dealing with this intricacy, which involves organising, exchanging information, and handling errors, can be difficult and might affect the overall dependability of the system.

2.7.4 Lightweight IDS based on feature extraction approaches

Another avenue for achieving lightweight intrusion detection involves feature extraction through dimensionality reduction. Recognizing the potential costs associated with deploying deep learning models on IoT devices for intrusion detection, Zhao et al. (2021) proposed a Principal Component Analysis (PCA) technique tailored for efficient feature extraction with minimal computational overhead. Their method incorporates structural enhancements such as expansion and compression, inverse residual, and channel shuffle to optimize feature extraction. To address the multiclassification challenges stemming from sample distribution imbalances, the authors introduce a specialized NID loss. Experimental results using real-world NID datasets showcase the method's enhanced classification performance, low model complexity, and reduced size, rendering it well-suited for classifying IoT traffic in both normal and attack scenarios. Similarly driven by the imperative to bolster intrusion detection systems, Aburomman and Reaz (2016) advocate for an ensemble approach combining Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) feature extraction algorithms, termed PCA-LDA. This ensemble method surpasses individual feature extraction techniques, yielding higher precision rates. The study underscores the ensemble's efficacy in maximizing feature informativeness for intrusion detection. In a different vein, Sakurada and Yairi (2014) promote the use of autoencoders for anomaly detection by leveraging nonlinear dimensionality reduction. This approach proves adept at detecting subtle anomalies and offers advantages over linear

PCA, presenting a computationally lighter alternative to kernel PCA. Additionally, the study highlights autoencoders' capacity to learn normal states and exhibit differential responses to anomalous input in the hidden layer.

In a recent advancement, Saheed et al. (2022) introduced a Machine Learning-based Intrusion Detection System (ML-IDS) aimed at fortifying security and privacy within IoT networks. Their methodology begins with feature scaling using min-max normalization on the UNSW-NB15 dataset, encompassing diverse attack types and normal network activities. Subsequently, Principal Component Analysis (PCA) was employed to reduce dimensionality, followed by analysis using six machine learning models. The experimental findings showcase competitive performance, boasting an accuracy of 99.9% and a Mathew correlation coefficient (MCC) of 99.97%, underscoring the efficacy of ML-IDS in addressing security and privacy concerns within IoT networks. Furthermore, Li et al. (2022) delve into the challenges surrounding effective feature extraction for network intrusion detection using prevailing deep learning methods, particularly in capturing the hierarchical structure of network flows. To counter this, they propose a Hierarchical and Dynamic Feature Extraction Framework (HDFEF) technique. This framework treats complete network activities as sequences of packets with multiple network flows and employs a hierarchical network model with an attention mechanism to dynamically adjust feature representations. The discriminant vectors obtained through multispace mapping are then utilized for classification. Experimental results across the CSE-CIC-IDS2018, CIC-IDS2017, and UNSW-NB15 datasets underscore the superiority of HDFEF over other state-of-the-art methods in network intrusion detection. Aligned with feature extraction strategies, Pajouh et al. (2016) present a model employing a two-layer dimension reduction approach and a two-tier classification module for intrusion detection in IoT networks. Focusing on the crucial task of identifying intrusions, particularly against User to Root (U2R) and Remote to Local (R2L) attacks, the dimension reduction module uses component analysis and linear discriminant analysis to diminish dataset dimensionality. Meanwhile, the two-tier classification module uses Naïve Bayes and the Certainty Factor version of K-Nearest Neighbor to pinpoint suspicious behaviors. Experimental results using the NSL-KDD dataset demonstrate the outperformance of the proposed model over previous intrusion detection models tailored for U2R and R2L attacks.

Although the proposed intrusion detection models show promise, challenges may impact real-time applications. Also, ensembled models introduce computational complexity, potentially hindering efficiency in resource-constrained settings. Similarly, longer training times, decreased interpretability, and higher dimensionality are some of the drawbacks

associated with ensemble methods. Furthermore, the quality and representativeness of training data influence machine learning model performance, thereby affecting generalization to real-world IoT network attack scenarios. Adaptability to adversarial attacks requires continuous updates and retraining that most of the models did not include in their approach. In addition, some of the datasets used in the approaches lack viability in IoT environments, necessitating demonstration for deployment assurance.

2.7.5 Lightweight IDS based on fog and cloud-based approaches

Due to the inherent resource constraints of the IoT, researchers have proposed fog and cloud-based approaches to mitigate the impact of IoT limitations. For instance, Sudqi Khater et al. (2019) advocate for a lightweight Intrusion Detection System (IDS) technique tailored for fog and IoT networks. Their approach harnesses fog computing to address cloud-related challenges. The proposed IDS adopts a vector space representation and implements a Multilayer Perceptron (MLP) model, which is well suited for resource-constrained fog and IoT devices. Evaluation against the ADFA-LD and ADFA-WD datasets, which contain various exploits and attacks, validates the effectiveness of the IDS. Achieving a 94% Accuracy, 95% Recall, and 92% F1-Measure in ADFA-LD, and 74% Accuracy, 74% Recall, and 74% F1-Measure in ADFA-WD, the IDS uses a single hidden layer and a minimal number of nodes. Notably, the performance evaluation was conducted on a Raspberry Pi. With increasing concerns over new malware strains such as Satori, Reaper, Amnesia, and Masuta, which exploit software vulnerabilities using Mirai's leaked source code, Kumar and Lim (2019) highlight the urgency of the issue. The authors further contend that the malware strains pose challenges for conventional solutions, such as firewalls, because they infect IoT devices during the scanning/infecting phase rather than during an attack. To address this, they proposed EDIMA, a distributed modular solution crafted for detecting IoT malware network activity in large-scale networks such as ISPs and enterprise networks. EDIMA leverages machine learning algorithms for edge device traffic classification, backed by a packet traffic feature vector database, a policy module, and an optional packet sub-sampling module. Evaluation of EDIMA's classification performance through testbed experiments yields promising results, contributing to the ongoing efforts in IoT security enhancement.

Another cloud-based strategy was presented by Parra et al. (2020), advocating for a cloud-based intrusion detection system employing distributed deep learning to combat a spectrum of attacks, including DDoS, phishing, and Botnet attacks. Their model integrates a Distributed Convolutional Neural Network (DCNN) for on-device phishing and

application layer DDoS attack detection, supplemented by a cloud-based temporal LSTM network for botnet attack detection. To address the need for reliable and efficient IoT connections, Almiani et al. (2020) proposed a fully automated intrusion detection system tailored for Fog security against cyber-attacks. Their model harnesses multi-layered recurrent neural networks customized for implementation in Fog computing security, strategically positioned near end-users and IoT devices. Evaluation conducted on the NSL-KDD dataset underscores the model's stability and robustness, validated through diverse performance metrics such as the Mathew correlation and Cohen's Kappa coefficients, underscoring its efficacy in thwarting security threats in Fog computing and IoT landscapes. Similarly, Alrawais et al. (2017) introduced a mechanism that leverages fog to enhance the dissemination of certificate revocation information among IoT devices, thereby fortifying security. The study also delineates potential research avenues, focusing on exploiting fog computing to address security and privacy concerns in IoT environments.

2.7.6 Lightweight IDS based on other approaches

Sanchez et al. (2021) present a lightweight solution for detecting Distributed Denial of Service (DDoS) attacks on IoT devices and critical infrastructure (CI). Their proposed method employs Analysis of Variance (ANOVA) for feature reduction, addressing the costliness associated with traditional machine learning and deep learning approaches. The study demonstrated a notable reduction (up to 84.21%) in the required data input for detection, with only a marginal (0.1%) decrease in accuracy. Through a comprehensive analysis of DDoS attack characteristics using ANOVA, the authors compare their approach with recent DL-based DDoS detection systems, showcasing comparable results. Similarly, Anthi et al. (2018) express concern over privacy challenges stemming from the proliferation of interconnected Internet of Things (IoT) devices, which often harbor sensitive personal information, rendering them susceptible to cyber-attacks and potential weak points in secure infrastructures. In response to notable incidents such as the Mirai botnet attacks, the study advocates for a dedicated Intrusion Detection System (IDS) tailored to monitor IoT ecosystems. Introducing Pulse, a novel IDS for IoT, the study leverages Machine Learning (ML) methodologies, particularly focusing on the system's capacity to effectively identify network scanning probing and basic Denial of Service (DoS) attacks in heterogeneous IoT environments during the initial development stages. Oh et al. (2014) discuss the challenges traditional security approaches encounter in adapting to the limited computing power and memory size of IoT devices. Proposing a

lightweight security system, the authors introduce a malicious pattern-matching engine. To address resource constraints, the system optimizes memory usage and implements techniques such as auxiliary shifting and early decision to efficiently reduce matching operations on resource-constrained devices. Experimental findings showcase significant speedups, with a maximum speedup of 2.14 observed with an IoT object, underscoring the scalable performance and effectiveness of the proposed system in addressing security concerns in the IoT environment. Furthermore, Lee et al. (2014b) present an energy-based approach to identify attacks in 6LoWPAN networks. By analyzing nodes' energy consumption patterns, particularly across different routing schemes, the system flags irregular energy usage as a potential malicious activity. Simulations validate the system's efficacy in accurately detecting and mitigating such attacks.

In addressing the multifaceted challenges of IoT security and data confidentiality stemming from the widespread deployment of IoT-enabled devices, particularly their susceptibility to unauthorized access and various cyber threats in wireless communication, Gupta et al. (2013) propose leveraging computational intelligence techniques to craft adaptive and cognitive intrusion detection systems capable of efficiently identifying malicious network activities. Their proposal introduces a three-tier architecture tailored specifically for intelligent intrusion detection systems in wireless networks, aiming to overcome the limitations of conventional intrusion detection systems in handling the intricacies and scale of wireless networks among evolving user behaviors and activity patterns. Likewise, Jun and Chi (2014) focused on security issues arising from the deployment of services and applications in IoT environments, where a myriad of situation-aware sensors continuously generate vast amounts of data. The authors highlight the challenge faced by real-time intrusion detection systems (IDS) in swiftly processing diverse data patterns to respond to hacking attacks. To tackle this challenge, they advocate for the integration of Complex Event Processing (CEP) technology into traditional IDS to enhance their performance in IoT settings. This integration enables the IDS to effectively discern complex patterns among events and process large message volumes with minimal latency. In addition, the study introduces an event-processing IDS architecture based on a thorough security requirements analysis and outlines the implementation details for real-time event processing, leveraging the Esper CEP engine for complex event processing and event series analysis. Similarly, Le et al. (2016) propose an Intrusion Detection System (IDS) tailored to detect Routing Protocol for Low power and Lossy network (RPL) topology attacks. Their IDS utilizes a RPL specification derived through a semi-automated profiling technique, which abstracts high-level operations from network simulation traces. This

specification, which delineates legitimate protocol states and transitions along with corresponding statistics, is then instantiated as rules in intrusion detection agents, specifically cluster heads, disseminated across the network for monitoring purposes. To optimize resource utilization, cluster members relay pertinent information to the cluster head rather than broadcasting to all nodes, enabling cross-verification by the cluster head. Through simulations, the proposed system demonstrated improved accuracy in detecting RPL topology attacks with minimal overhead, ensuring scalability in large-scale networks.

In a related context, prompted by the IoT's susceptibility due to its open deployment setting and resource limitations, as well as the necessity to overcome the constraints of conventional intrusion detection systems due to the IoT's heterogeneous and distributed nature, Deng et al. (2019) proposed a comprehensive system framework along with key security technologies. These include key management, authentication, access control, routing security, privacy protection, intrusion detection, fault tolerance, and intrusion prevention. The study underscores the pivotal role of intrusion detection in fortifying IoT security and explores various intrusion detection technologies, comparing their applicability within IoT architecture. In addition, it delves into the burgeoning importance of data mining and machine learning methodologies in scrutinizing network intrusion technology, culminating in the validation of the proposed model's efficacy using public databases. Furthermore, Summerville et al. (2015) introduced an ultra-lightweight deep packet anomaly detection approach specifically tailored for resource-constrained IoT devices. The method leverages efficient bit-pattern matching for feature selection, thus minimizing computational overhead. Implementing the discrimination function as a lookup-table enables rapid evaluation and adaptable feature space representation. Through experimentation with off-the-shelf IoT devices, the authors showcase the effectiveness of this approach in achieving near-optimal payload discrimination. To address the intricacies of intrusion detection datasets, Li and Majd (2023) harnessed machine learning and deep learning algorithms alongside feature selection techniques, focusing on the UNSW-NB15 dataset and employing multi-access edge computing. Their LightGBM approach yields remarkable results, overcoming challenges related to data imbalance and missing data. Similarly, Huang et al. (2023) discuss the challenge of identifying unknown cyberattacks in an evolving cyber threat landscape and the prevalence of emerging technologies such as 5G and digital twins. Recognizing the effectiveness of existing intrusion detection systems (IDSs) in detecting known cyberattacks but their limitations in handling unknown threats, the authors introduce HiDE-IDS, inspired by artificial immunity (AIm). This approach entails mapping normal and abnormal network samples to self

and nonself antigens in a multidimensional space. A hierarchical differential evolution algorithm then evolves antigens to create cyberattack detectors. A filtering mechanism eliminates invalid antigens, leaving behind those used to generate detectors for identifying both known and unknown cyberattacks. Experimental results demonstrate improved training efficiency compared to recent IDSs, with HiDE-IDS achieving favorable false positive rates for normal data.

Derhab et al. (2019) proposed a blockchain-driven strategy, merging blockchain and Software-defined Network (SDN) technologies to protect commands within Industrial IoT setups. The framework incorporates RSL-KNN, which combines Random Subspace Learning (RSL) and K-Nearest Neighbor (KNN) techniques, effectively countering fraudulent commands. Murali and Jamalipour (2019) introduced an Artificial Bee Colony (ABC)-inspired mobile Sybil attack modeling, alongside a lightweight intrusion detection algorithm crafted for mobile RPL. The study delved into Sybil attack behaviors and RPL's performance under such conditions, analyzing factors such as packet delivery ratio, control traffic overhead, and energy consumption. The proposed algorithm's efficacy was evaluated on the basis of accuracy, sensitivity, and specificity. Moreover, Ma et al. (2023) recognized the pivotal role of NIDS performance, which is often influenced by the detection model's effectiveness, learning mechanism, and available training data. To address these concerns, the authors proposed ADCL, a Collaborative Learning-based Detection framework. ADCL harnesses multiple models trained in similar environments to collaborate on intrusion detection, aiming to surpass individual model limitations. Evaluation results showcased ADCL's improved performance over single models in detecting diverse attacks in IoT networks, with significant improvements in adaptability, learning integrity, and model capacity. Similarly, Koroniotis et al. (2020) introduced the Particle Deep Framework (PDF), a novel network forensics framework designed to address vulnerabilities in lightweight and low-power household devices. The PDF encompasses three key functions: extracting and verifying network data flow integrity, dynamically adjusting deep learning parameters using a Particle Swarm Optimization (PSO) algorithm, and developing a Deep Neural Network (DNN) with the PSO algorithm to identify and trace abnormal events in IoT networks. Evaluation using the Bot-IoT and UNSW_NB15 datasets showcased PDF's impressive performance in detecting and tracing cyber-attack events, outperforming other deep learning techniques. Again, Shafiq et al. (2020) proposed a hybrid algorithm and framework for identifying malicious traffic in IoT networks, effectively selecting the most suitable machine learning algorithm for IoT anomaly and intrusion traffic identification. Additionally, Jung et al. (2020) addressed the growing

threat of IoT botnets by leveraging power consumption patterns for classification, with their CNN-based model showing promising results in botnet detection. Furthermore, in the context of securing the physical (PHY) layer in communication technologies, particularly in IoT and fifth-generation (5G) cellular networks, Anajemba et al. (2020) introduced an efficient Sequential Convex Estimation Optimization (SCEO) algorithm for a three-node wireless communication network. Evaluation results demonstrated the optimal performance of the SCEO algorithm and enhanced convergence in transmission.

While the proposed models demonstrated near-perfect results, challenges exist for effective implementation. (1) Integrating the three-tier architecture into existing wireless networks may encounter compatibility issues and potential disruptions. (2) The cost of implementing and maintaining the computational intelligence-based intrusion detection system could be significant, encompassing software development, hardware requirements, and ongoing updates. (3) The IDS's complexity, involving a semi-auto profiling technique and intricate rule implementation, may pose challenges for maintenance, updates, and troubleshooting, requiring careful management for long-term viability. Additionally, the training overhead for mapping network samples to antigens and evolving new antigens may impact system responsiveness, particularly in scenarios demanding quick detection. Furthermore, the algorithm's reliance on behavioral analysis to detect Sybil attacks may present challenges in accurately characterizing normal and malicious behaviors, particularly in dynamic IoT environments.

2.7.7 A Comparison of Related works

Overall, the studies demonstrate the diverse range of approaches and methods employed by different researchers to detect intrusions in IoT systems. Furthermore, Table 2.1 provides an overview of about 80% of the studies that were randomly selected to highlight their evaluation environment, accuracy, precision, recall, F1-score, computation time, and model size.

Approximately 80% of the methodologies outlined in this section have been summarized in Table 2.1. For instance, the outputs of these methodologies were categorized based on several criteria, including evaluation environment (such as Windows system, edge, IoT device, Raspberry Pi, and server), as well as metrics like accuracy, precision, recall, F1 score, execution time, and model size. While reducing computational costs is crucial for IoT-based IDS, many studies solely prioritize overall accuracy, while some delve deeper by incorporating precision, recall, and F1 scores into their assessments. Among the

Table 2.1: A summary of results of the various studies

Reference	*Evaluation Envir*	*Acc*	*Prec.*	*Recall*	*F1*	*Run_time*	*Model_size*
Summerville et al. (2015)	IoT	92.9	X	X	X	X	X
Nguyen et al. (2022)	Raspberry Pi	99.57	X	99.57	X	10,600/s	114.5MB
Siddharthan et al. (2022)	Raspberry Pi	99	X	X	100	0.04s	X
Lee et al. (2014b)	Raspberry Pi	100	X	X	X	0.5s	X
Sudqi Khater et al. (2019)	Raspberry Pi	94	X	95	92	X	X
Zhao et al. (2021)	Edge	98.94	X	X	98.93	X	X
Idrissi et al. (2021)	Edge	96.69	X	X	X	2e-6	2.704KB
Boppana and Bagade (2023a)	Edge	97.3	97.4	97.3	97.3	X	X
Abdel-Basset et al. (2021b)	Edge	99.6	99.48	99.23	99.35	1.1s	X
Parra et al. (2020)	Edge	97.74	95.60	99.91	97.70	X	X
Almiani et al. (2020)	Edge	92.42	90.30	X	X	X	X
Derhab et al. (2019)	Edge	100	X	X	X	0.248s	X
NG and Selvakumar (2020)	Edge	99.75	99.99	99.75	99.87	X	X
Saba et al. (2022)	Edge	99.51	X	X	X	X	X
Sharma et al. (2024)	Edge	99	100	99	100	X	X
Okey et al. (2023)	Edge	100	X	X	X	X	X
Ciklabakkal et al. (2019)	Windows System	99	X	X	X	X	X
Li (2022)	Windows System	98.9	98	98.6	98.3	X	X
Jaafar et al. (2022a)	Windows System	99.34	X	X	X	17.57s	X
Fatima et al. (2023)	Windows System	99.5	99	X	99	0.52s	563MB
Li et al. (2020)	Windows System	92.69	90.85	86.63	88.69	X	X
Wang et al. (2022b)	Windows System	99.44	99.48	99.47	99.46	X	18.1KB
Jaw and Wang (2021)	Windows System	99.99	99.2	99.75	99.3	208s	X
Hanafi et al. (2023)	Windows System	98.21	98.48	98.92	97.25	X	X
Rizvi et al. (2022)	Windows System	99.7	X	X	X	X	X
Li et al. (2022)	Windows System	99.7	99.73	99.96	99.84	138.098s	X
Wang et al. (2019)	Windows System	98.44	98.60	98.47	98.51	X	18.1KB
Kasongo and Sun (2020)	Windows System	90.85	75.50	77.53	X	X	X
Panthong and Srivihok (2015)	Windows System	89.60	X	X	X	X	X
Acharya and Singh (2018)	Windows System	99.09	99.4	X	X	X	X
Halim et al. (2021)	Windows System	99.80	X	X	X	X	X
Fenanir et al. (2019)	Windows System	95	100	X	96	X	X
Shafiq et al. (2020)	Windows System	99.99	100	100	X	X	X
Davahli et al. (2020)	Windows System	99.09	96.31	99.30	97.84	10.90	X
Sanchez et al. (2021)	Windows System	99.33	98.50	99.91	99.20	X	X
Roy et al. (2022)	Windows System	99.11	99.08	99.11	99.08	0.02s	X
Mushtaq et al. (2022)	Windows System	89	88	94	91	X	X
Xu et al. (2020)	Windows System	93.3	X	99.8	96.5	X	X
Jan et al. (2019)	Windows System	98.35	X	X	X	0.047	X
Ullah and Mahmoud (2021)	Windows System	99.96	99.90	99.95	99.93	X	X
Aburomman and Reaz (2016)	Windows System	92.16	X	X	X	X	X
Li and Majd (2023)	Windows System	99.1	X	X	X	1.11s	X
Zhang et al. (2023)	Windows System	99.48	99.54	99.38	99.46	0.995s	X
Huang et al. (2023)	Windows System	X	97.79	X	96.05	8.93s	X
Koroniotis et al. (2020)	Windows System	99.9	100	99.9	99.9	X	X
Priya et al. (2022)	Server	98	X	X	X	X	X

methodologies listed in Table 2.1, only about 34.7% reported the execution time of their models, and a mere 10.86% provided information on the model size. Moreover, some of the execution times and model sizes presented in the table could be further optimized. While there are no set values as to what constitutes a minimal run-time and model size, there is always a need to strive to provide a lower model run-time and size compared to what is currently presented in the literature. Authors have cited various reasons for implementing/deploying their models in specific environments, including:

1. That the constrained IoT problem is partly solved by edge computing, which minimizes the quantity of data sent to the cloud. Bedi et al. (2021); Hasan et al. (2019); Schneible and Lu (2017)

2. Anomaly detection poses a significant challenge due to the sheer volume of data involved. Because excessive throughput, such as multiple accesses during result announcement, has the potential to overload the detection engine unnecessarily Dini and Saponara (2021); Mokhtari et al. (2021); Selim et al. (2021).

3. Effectively handling the copious and intricate data generated by IoT devices poses a challenge, particularly due to the sheer number of attributes that necessitate specialized applications for analysis. Hence, the adoption of dimensionality reduction techniques becomes imperative for feature management Janjua et al. (2019); Wang et al. (2022a).

Several studies acknowledge the inherent limitations of IoT devices in processing and detecting intrusions effectively. Some researchers have suggested that deploying intrusion detection techniques directly on edge devices can address this shortfall in IoT capabilities. Notably, studies whose models were deployed on Windows systems omitted crucial metrics such as model size and computation time. When provided, these metrics tend to be disproportionately large. These measurements are essential for accurately assessing the reduction in computation costs. Therefore, addressing the following gaps and questions is crucial for developing an efficient, lightweight intrusion detection system in the Internet of Things.

2.8 Gaps in the related works

- A number of the proposals have identified the challenges facing intrusion detection systems and their reliance on intelligent algorithms for real-time processing and detection. However, other than prediction accuracy, there is no explicit explanation

nor provisions in the proposals on how the approaches were implemented in the IoT environment to detect the attacks.

- Researchers have explored mainly the implementation of lightweight IDS on fog networks. However, it is important to consider scenarios where IoT devices are deployed in remote and rural areas, where edge devices may not be readily available (as mentioned in Section 2.2). In such cases, it becomes crucial to determine how these approaches can be effectively implemented.

- Several studies reviewed in the literature failed to provide essential information regarding model size and computation time, which are crucial components of a lightweight intrusion detection system. In cases where this information was eventually provided, the sizes reported were found to be a bit larger, which could be improved upon.

Given the identified gaps in the relevant literature and the imperative to develop a computationally effective intrusion detection model for the IoT in critical infrastructure, several pertinent questions arise that demand exploration. These are the basis for the central research question and the subdivided questions (see section 1.2.3).

2.9 Summary

Overall, this chapter explores an array of innovative studies on IoT intrusion detection approaches. These include AI-based lightweight detection, feature selection and feature extraction approaches, ensemble techniques, and fog computing. The approaches demonstrate a keen focus on addressing specific attack types, harnessing advanced algorithms, and emphasizing the significance of relevant methodologies in optimizing intrusion detection efficiency. In addition, this chapter critically examines pertinent questions arising from these studies and highlights key knowledge gaps. However, the feature selection and machine learning approaches used by Roy et al. (2022); Zhang et al. (2022) was explored for further work in other to achieve an effective and efficient intrusion detection model.

Chapter 3

Research Methodology

3.1 Methodology overview

The previous chapter conducted a thorough literature review on intrusion detection in IoT, identifying gaps that informed the research questions. This chapter introduces the methodologies aligned with these questions, focusing on developing a lightweight intrusion detection model for IoT in critical infrastructures. The sequential structure ensures each method's output contributes to subsequent ones, collectively achieving the thesis objectives. Various approaches, including obtaining relevant data, feature selection, augmentation, extraction, pruning, training, quantization, resilience, inferencing, and ethical practices, were employed. Figure 3.1 illustrates the workflow of these methodologies.

3.2 Dataset

This study used publicly available benchmark datasets that are highly relevant to IoT devices. The datasets primarily comprise simulated data related to pipelines, smart grids, and other IoT-specific attacks. The use of multiple datasets in this study is basically to ensure that the feature selection technique adopted in this study is effective across different data regimes. To this end, efforts were made to ensure the datasets were directly aligned with the IoT attacks and environments. The datasets used in this study include: smart grid dataset Pan et al. (2015a), BoT-IoT dataset Koroniotis et al. (2019), gas pipelines and water storage tanks Morris and Gao (2014b); Morris et al. (2011b). Others are: NF-BoT-IoT dataset Sarhan et al. (2021), CIC-IDS2017 dataset Sharafaldin et al. (2018), UNSW18 dataset Moustafa (2019), and MQTT-IoT-IDS2020 Hindy et al. (2020).

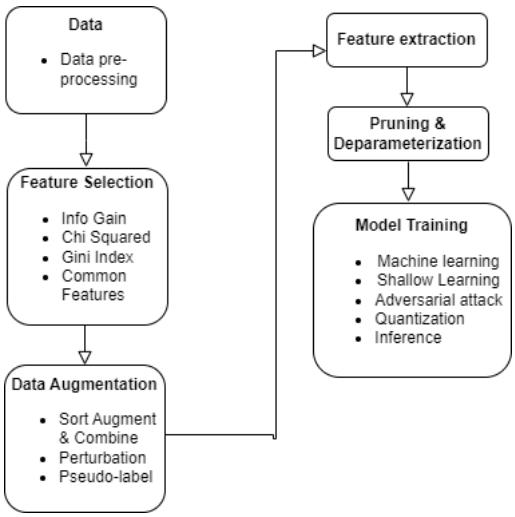


Figure 3.1: A flow of methodologies used in this thesis.

3.2.1 Data preprocessing

For each dataset utilized, rigorous screening was undertaken during data preprocessing to ensure its usability, with a focus on equal contributions from all independent variables. Values such as Infinity and NA were replaced before starting the normalization process. Data normalization, a crucial preprocessing technique, proves beneficial by providing equal weight to all data attributes, thereby expediting the algorithm training phase. The choice of the normalization approach used was based on (a) the dataset used and (b) the necessity to ensure equitable contribution from all attributes. Normalization is indispensable because the measurement unit employed during data collection could impact the analysis, potentially causing certain features to contribute more significantly than others Han et al. (2012). The two normalization approaches employed are as follows:

- **Min - Max Normalization:** This method is a data normalization technique that transforms the data to have a minimum value of 0 and a maximum value of 1. As expressed in equation 3.1, let y_i denote the normalized value of the i^{th} feature. The original values are denoted by x_i, while min_{xi} and max_{xi} represent the minimum and maximum values of the i^{th} feature in the dataset, respectively.

$$y_i = \frac{x_i - min_{xi}}{max_{xi} - min_{xi}} \tag{3.1}$$

- **Standard Normalization:** Standard normalization, also known as z-score normalization, is a technique employed to bring features onto a common scale, especially when dealing with features of different scales and units or when using algorithms sensitive to the distribution and scale of the data. This method is useful even when there are outliers within the dataset. Standard normalization transforms the data based on the mean and standard deviation of each feature. As shown in equation 3.2, let z_j denote the normalized value of the j^{th} feature. The original values are represented by x_j while the μ and σ are the mean and standard deviation values of the j^{th} feature in the dataset, respectively.

$$z_j = \frac{x_j - \mu}{\sigma} \tag{3.2}$$

3.3 Feature Selection Methodology

The feature selection phase in this study emphasizes the identification of relevant features driven by the recognition that high-dimensional features can lead to overfitting and degradation of classification algorithm outputs. In addition, generating a large volume of high-dimensional data by devices can prolong training, generalization, and classification processes. For this reason, feature selection offers the advantage of simplifying the models and enhancing data interpretability. There are several approaches to feature selection, and they include Boruta Dag et al. (2023), Variable Importance from Machine Learning Algorithms, Lasso Regression Guenther and Sawodny (2019), Stepwise Forward and Backward Selection Derksen and Keselman (1992), Relative Importance from Linear Regression, Recursive Feature Elimination (RFE) Yan and Zhang (2015), Genetic Algorithm Siedlecki and Sklansky (1989), Simulated Annealing Abdel-Basset et al. (2021a), Information Gain Win and Kham (2019), Chi-square Test Sikri et al. (2023), Fisher's Score Gu et al. (2012), Missing Value Ratio Yu et al. (2022), and Gini-index Liu et al. (2018). However, several of these techniques would require a longer processing time, thereby adding additional cost to the device. Second, since this work is based on classifying attacks, the correlation between independent variables and the target variable is very important. This is why, in this work, three feature selection techniques were used, and they include Information Gain, Chi-Squared, and Gini Index. These were chosen for this study due to their relevance to intrusion classification, particularly concerning the target variable. Information Gain, Chi-Squared, and Gini Index are valuable techniques for feature selection in cybersecurity datasets as they identify informative attributes that contribute to the accurate classification of network traffic, thereby aiding in developing

robust intrusion detection systems. These techniques address (i) data disorder or anomalies, (ii) the need for precise data classification within the traffic stream, and (iii) the quantification of the value spread in a feature, along with the likelihood of misclassification for a randomly chosen instance. After data preprocessing, the three techniques were applied to rank features based on their importance, and the cumulative variance for each technique was computed until a saturation threshold was reached. This threshold signifies the point at which further increases in variance do not significantly contribute to cumulative variance. Subsequently, subsets of features were selected on the basis of each technique, and a common subset of features present across all subsets was identified. These common features were then selected for the next phase of the study, which involved data compression.

Beyond the selection of non-redundant features, the advantages of combining multiple feature selection techniques include the following:

- Diverse Perspectives: Various feature selection techniques operate based on distinct principles and assumptions. Utilising multiple techniques allows for a range of viewpoints on the significance of features, enabling a comprehensive understanding of the data distribution and relationships.
- Robustness: Various feature selection methods may demonstrate different sensitivities to the characteristics of the dataset. By employing a variety of techniques, the model gains increased resilience, as it becomes less susceptible to the peculiarities of any single technique.
- Enhanced Generalization: Non-redundant features selected by multiple techniques are more likely to be informative and relevant across different scenarios. This helps the model to effectively generalize and to perform optimally on unseen data.
- Addressing Data Heterogeneity: Sometimes, datasets may exhibit inherent heterogeneity, wherein specific attributes may have varying degrees of influence within different subsets of the data. A combination of feature selection techniques aid in capturing these subtle distinctions and selecting features that consistently contribute across different subgroups.
- Reduction of Overfitting:Feature selection mitigates overfitting by exclusively selecting the most informative features. Employing a combination of feature selection techniques reduces the likelihood of overfitting to the peculiarities of a singular approach.

- Model Interpretability: Non-redundant features selected by multiple techniques often correspond to significant patterns in the data. This improves the interpretability of the model, facilitating the understanding of the factors that influence predictions.
- Complementary Information: Various feature selection methods may prioritize distinct aspects of the data, such as statistical significance, information gain, or importance, based on the model. By combining these methods, a more comprehensive perspective on feature importance can be obtained.
- Handling Multicollinearity: When there is a high correlation between features (multicollinearity), employing a combination of feature selection techniques aids in identifying and preserving the most pertinent features, thereby reducing the influence of redundant information.

3.3.1 Feature Selection based on Information Gain

Several algorithms employ diverse heuristic filter criteria to gauge feature importance, aiming to maximize relevance and minimize redundancy Duda et al. (2006). Notably, the information content of a feature is assessed in relation to its correlation with the target class. Information Gain, as a feature selection technique, quantifies the information a feature can provide concerning the target feature. It is very useful in intrusion detection as it identifies non-redundant features that aid detection. In information theory, higher uncertainty implies a lower amount of contained information. Therefore, by computing feature weights and selecting the most relevant ones, information gain reduces the dataset's dimensionality. It also enables the evaluation of how much knowledge about one feature decreases uncertainty regarding the target feature. The merits of information gain that justify its application include the following Appavu et al. (2011):

- **Feature Selection:** Information Gain helps identify a set of attributes that provide the most valuable information for classification tasks. This is very important because it simplifies the model without sacrificing its accuracy.
- **Reduced Overfitting:** By selecting the most relevant features, the risk of overfitting is reduced as the model learns from the data patterns rather than memorizing the data patterns.
- **Improved Model Performance:** Models trained on reduced features tend to perform more efficiently, with faster training times and lower memory use.

Each feature's information gain is calculated in terms of entropy, $E()$, Azhagusundari et al. (2013). To rank the features, the *mutual_info_classif* class of the *sklearn.feature_selection* library in Python was used. The output was generated as a list and sorted in decreasing order of magnitude. Therefore, the output of the process upon completing the computation of the cumulative variance became the subset for subsequent work. The threshold determining the cut-off point is the saturation point during the computation of the cumulative variance. It is the point where further addition of variance results in no increase in the cumulative variance.

$$IG(S, x) = E(S) - E(S|x) \tag{3.3}$$

where:
$IG(S, x)$ is the information gain for the dataset, S
x is a random variable
$E(S)$ is the entropy of S
$E(S|x)$ is the conditional entropy of S given x.

$$E(S) = -\sum_{i=1}^{n} P_i log_2 P_i \tag{3.4}$$

3.3.2 Feature Selection based on Chi-Square

The chi-squared (χ^2) technique is an algorithm designed to assess the relationship between categorical variables. It measures the independence between the two categorical variables and determines whether a significant association exists. In a classification context, χ^2 helps identify the independent (predictor) features relevant to classification purposes. A higher chi-squared value indicates a stronger dependence of the feature on the target, making it more important and suitable for model training Rachburee and Punlumjeak (2015). Using the *SelectKBest* function from the Python *sklearn.feature_selection* library, features were ranked and output as a list in decreasing order of magnitude. After computing the cumulative variance, a subset of features was generated for subsequent analysis.

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{3.5}$$

where:
c is degree of freedom
O is observed values
E is expected values

3.3.3 Feature Selection based on Gini-Index

The Gini Index, introduced by Breiman in 1984, has found widespread application in various algorithms, including feature selection, exhibiting favorable classification outcomes. This statistical method quantifies impurities by measuring the degree to which a specific node in a decision tree is mixed with different classes. In the context of feature selection with decision trees, the Gini Index assesses the importance of each feature in decision-making, guiding the selection of features for data splitting at each node. The reduction in impurity resulting from a split serves as a measure of feature importance. The Gini Index is particularly well-suited for binary and continuous numeric values Manek et al. (2017). In a binary tree, considering a right split denoted as R with a corresponding right Gini node represented as G_R and a left split denoted as L with a left Gini node represented as G_L, the Gini Index (G) and the decrease in impurity (d_{ij}) at a single node can be calculated as follows:

$$d_{ij} = G - (\frac{N_L}{N} G_L + \frac{N_R}{N} G_R) \tag{3.6}$$

Where:
N is the number of units in the dataset
The measure of the feature importance (FI) for the ranking of a feature, K_i in a tree, t, is as defined by Sandri and Zuccolotto (2008).

$$FI_{K_i}(t) = \sum_{j \in J} d_{ij} I(K_i \, splits \, at \, node \, j) \tag{3.7}$$

3.3.4 Common Features Subset

The term "Common Features subset" refers to features that are shared or intersect among different feature selection techniques. The selection of this common feature subset is derived from the subsets of features obtained through the ranking and cumulative

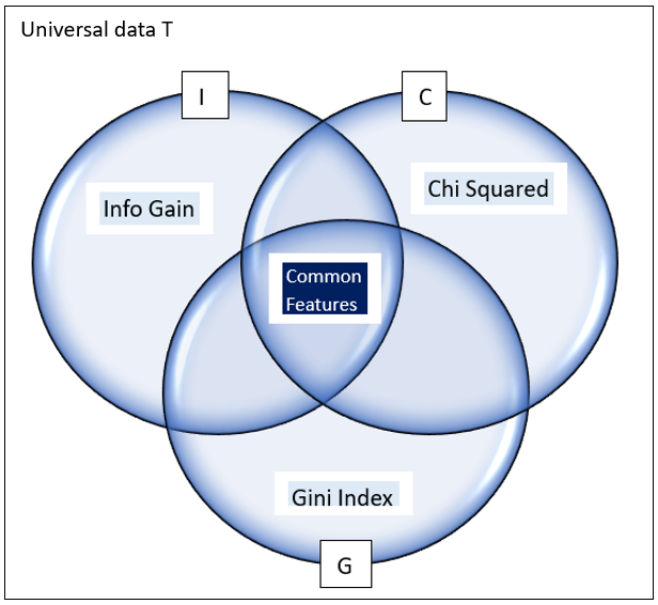


Figure 3.2: Features and the area showing common features - intersection of features.

variance computation of the three feature selection techniques. The cumulative variance determines a threshold, indicating the point at which additional feature values cease to contribute to any significant increase in variance. Once the threshold is established, three subsets are created for each technique. The process of selecting the common subset of features begins, comprising only those features present in all three subsets, while non-common features are discarded. This selection is based on the premise that the selected features possess essential properties for effective use in subsequent processing, particularly during training by more learning algorithms.

Consider a dataset, T, which contains features $(v_1, v_2, v_3, ..., v_n)$ that were ranked using information gain, Chi-squared, and Gin-index feature selection techniques. Let I, C, and G be T subspaces. In order words, the subspaces are the outcome of the feature selection strategies. This implies that $I \in T$, $C \in T$, and $G \in T$ are valid, and a representation of the set and subspaces is shown in Figure 3.2.

The area indicated as common features in Figure 3.2, is the point of interest in this phase. This is because it is the region shared by all subspaces (i.e. I, C, and G) of a universal vector space T. The common features are important because they help in the identification the features which would be eventually used to shrink the original

subspaces into a much smaller common features space that is customised for further processing.

Let $\overrightarrow{i}, \overrightarrow{c}, \& \overrightarrow{g}$ represent the common elements from the subspaces. This therefore imply that - $\overrightarrow{i} \in I,\ \overrightarrow{c} \in C,\ \& \ \overrightarrow{g} \in G$.

From set theorem using the subspace test, three axioms of closure, addition and scalar multiplication must hold for the common features to be representative of the original vector spaces.

1. For closure
   Since I, C and G are subspaces of T , they each contain $\overrightarrow{0}$ which by definition of intersection implies
   $\{0\} \in (I \cap C \cap G)$
   Similarly,
   $\overrightarrow{i} \in (\overrightarrow{I} \cap \overrightarrow{C} \cap \overrightarrow{G});\ \overrightarrow{c} \in (\overrightarrow{I} \cap \overrightarrow{C} \cap \overrightarrow{G});\ \overrightarrow{g} \in (\overrightarrow{I} \cap \overrightarrow{C} \cap \overrightarrow{G})$
   Therefore, $(\overrightarrow{i} \cap \overrightarrow{c} \cap \overrightarrow{g}) \in (I \cap C \cap G)$

2. For addition
   $\{0\} + (\overrightarrow{i} \cap \overrightarrow{c} \cap \overrightarrow{g}) = (\overrightarrow{i} \cap \overrightarrow{c} \cap \overrightarrow{g}) \in (I \cap C \cap G)$
   Also since,
   $\overrightarrow{i}, \overrightarrow{c}, \overrightarrow{g} \in (I \cap Y \cap G)$,
   we can draw from it that
   $\overrightarrow{i} + \overrightarrow{c} + \overrightarrow{g} \in I;\ \overrightarrow{i} + \overrightarrow{c} + \overrightarrow{g} \in C;\ \overrightarrow{i} + \overrightarrow{c} + \overrightarrow{g} \in G$
   $\implies \overrightarrow{i} + \overrightarrow{c} + \overrightarrow{g} \in (I \cap C \cap G)$

3. For scalar
   Let α be a scalar of 1 (unit vector) such that $\alpha \in (I \cap C \cap G)$
   Considering that $I, C \& G$ are subspaces of T, then
   $\alpha(\overrightarrow{i}) \in I;\ \alpha(\overrightarrow{c}) \in C;\ \alpha(\overrightarrow{g}) \in G$
   $\implies \alpha(\overrightarrow{i}\overrightarrow{c}, \overrightarrow{g}) \in (I \cap C \cap G) \in T$

Algorithm 1 feature selection approaches

Input: $Labeled - data$, T_d, $Number\,of\,iterations\,N_t$
Output: $Common_{feature}, C_f$
for $i = 1\,in\,f_n$: **do**
    $rank\,features \;\; IG_R \leftarrow f(x)(f_i, f_{ii}, f_{iii}, ..., f_n)$
    $rank\,features \;\; CS_R \leftarrow f(y)(f_i, f_{ii}, f_{iii}, ..., f_n)$
    $rank\,features \;\; GI_R \leftarrow f(g)(f_i, f_{ii}, f_{iii}, ..., f_n)$
end for
order (IG_R, CS_R, GI_R) $\;\; order\,magnitude\,in\,descending\,order$
for $a_i\,in\,(IG_R, CS_R, GI_R)$ **do**
    compute $Cum_{var}IG \leftarrow Cumulative_{var}(IG_R)$
    compute $Cum_{var}CS \leftarrow Cumulative_{var}(CS_R)$
    compute $Cum_{var}GI \leftarrow Cumulative_{var}(GI_R)$
    while $Cum_{var}(i) + Var(a_i) \not\approx Cum_{var}(i)$ **do** ◁ where i takes IG_R, CS_R, & GI_R
        $Cum_{var}(i) \leftarrow Var(a_{i+1})$
        $A_i \leftarrow Cum_{var}(i)$ ◁ save selected features
    end while
end for
for $f_i \;\; in \;\; (A_i, A_{ii}, A_{iii})$ **do**
    if $f_i \in (\,A_i \cap A_{ii} \cap A_{iii})$ **then** ◁ finding common features
        $select\,f_i$
        $save\,Y \leftarrow f_i$
    end if
end for
Return(Y)

In summary, three distinct feature selection techniques were applied to rank dataset features based on their importance. The computation of cumulative variance facilitates the selection of feature subsets for each technique. Subsequently, a subset comprising features common to the initial subsets was chosen for the subsequent phase of the study. The rationale behind employing the three feature selection techniques was to obtain a concise set of features that could enhance generalization and mitigate overfitting. Using a single feature selection method carries the risk of generating suboptimal feature subsets, potentially affecting the detection performance of the learning method. Conversely, employing a combination of feature ranking from multiple selection techniques and choosing a shared subset of features can improve classification and overall accuracy. This ensemble approach leverages the strengths of each technique, creating a more robust and effective feature subset Hoque et al. (2018); Rodríguez et al. (2007). The steps for achieving feature selection and identifying common features are outlined in Algorithm 1.

3.4 Data Augmentation Methodology

Data augmentation was incorporated into this study to address the prevalent issue of imbalanced classes in datasets Zhu et al. (2017). This is a common challenge in security data, especially in situations where the majority of the data is malicious. Various resampling techniques, including synthetic oversampling, undersampling, cluster-based undersampling, cost-sensitive learning, and instance weighing, have been employed to address this challenge by adjusting the size of the minority class. Studies by Dina et al. (2022) emphasize the effectiveness of using synthetic data for oversampling the minority class, resulting in improved classification. However, many oversampling techniques lack the underlying structural distribution of the original data, leading to issues like overfitting. Effective data augmentation, which enhances generalization and classification, requires the generated synthetic data to mirror the density and distribution of the original dataset Meurisch et al. (2020); Tang and He (2015). In this thesis, three distinct approaches to data augmentation were employed to oversample the minority classes in the dataset, improve generalization, and enhance classification. Two of the approaches were based on the Sort-Augment-Combine (SAC) algorithm Otokwala et al. (2021), while the third was based on the pseudo-data generation technique.

3.4.1 Sort-Augment-Combine (SAC)

In this technique, the dataset was split into subsets of the class labels and synthetic data were generated from each subset. The synthetic data were then used to oversample the minority classes in order to enhance the generalization of the learning algorithm. There are two approaches to the SAC technique, with the first being the use of the *synthop* package Nowok et al. (2016) for generating synthetic data. The second augmentation approach is the use of feature perturbation. In both cases, however, the data is Sort, Augmented and then Combined to form the new augmented dataset. The three stages involved in the SAC technique are as follows.

1. **Sort:** At this stage, the data are sorted into subsets of the instant classes. In other words, a binary class dataset will be sorted into two subsets: malicious and benign data. Given a data frame, S and consisting of classes: $A, B, C, ..., n$, the power set can be represented as, $P(S) = A, B, C, ...,$
   where,
   $A \in S$, $B \in S$, & $C \in S$.
   The expression implies that A, $B, \& C$, which are the instant classes of the dataset, S, and which can be further represented as a set as shown in equations 3.8 - 3.10.

$$A = a_1,\ a_2,\ a_3, ... \tag{3.8}$$

$$B = b_1,\ b_2,\ b_3, ... \tag{3.9}$$

$$C = c_1,\ c_2,\ c_3, ... \tag{3.10}$$

   Where $a_1, ..., b_1, ..., c_1, ..., n$ are elements of the subsets $A, B, \& C$

2. **Augment:** After dividing and sorting the data into subsets of the instant classes, a synthetic data function generator is used to generate synthetic data, which are then used to augment the original minority class(es). More importantly, the synthetic data helps maintain the distribution behind the original data variables. On the basis of this, for example, a new set of synthetic data values is generated to form equations 3.11, 3.12, and 3.13 from equations 3.8, 3.9, and 3.10.

$$\bar{A} = \bar{a_1}, \bar{a_2}, \bar{a_3}, ... \tag{3.11}$$

$$\bar{B} = \bar{b_1}, \bar{b_2}, \bar{b_3}, ... \tag{3.12}$$

$$\bar{C} = \bar{c_1}, \bar{c_2}, \bar{c_3}, ... \tag{3.13}$$

The minority classes are then supplemented independently. The process of augmentation is carried out by combining the generated synthetic data with the original subsets i.e.: equations (3.8) & (3.11); (3.9) & (3.12); and (3.10) & (3.13) are combined to form the augmented subsets of $A_{augmented}$, $B_{augmented}$, & $C_{augmented}$. As a result, the new augmented subsets are:

$$A_{augmented} = a1, a2, a3, \bar{a_1}, \bar{a_2}, \bar{a_3} \tag{3.14}$$

$$B_{augmented} = b1, b2, b3, \bar{b_1}, \bar{b_2}, \bar{b_3} \tag{3.15}$$

$$C_{augmented} = c1, c2, c3, \bar{c_1}, \bar{c_2}, \bar{c_3} \tag{3.16}$$

3. **Combine:** At this stage, the new augmented classes are combined to form a balanced new training dataset. In other words, combining equations (3.14), (3.15), and (3.16) gives us the new training dataset $P(S) = \{\} + A_{augmented} + B_{augmented} + C_{augmented}$. Because of the co-sharing of the variables by the subsets, the combination of the augmented subsets is achieved through row-binding.

$$New_{data} = combine(A_{augmented}, B_{augmented}, C_{augmented}) \tag{3.17}$$

Approach 1: Using Synthetic data library

In this approach, a function generator, Syn(), was used to synthesize the data value from the original dataset's latent space to increase the size of minority classes. The generator uses sequential regression modeling to synthesize each variable one after the other in the dataset. It fits the data to the assumed distribution and obtains estimates of its parameters on the basis of conditional distributions from which synthetic values are derived. For example, consider a dataset of variables $(Z_1, Z_2, ..., Z_n)$. Synthesis is

performed such that the first variable to be synthesized is Z_1 however, because it lacks predictors before it, its synthetic values are therefore generated through random sampling with replacement from its original values. The distributions of the succeeding variables are then estimated and synthesized on the basis of the conditional distributions of the preceding variables Nowok et al. (2016). In addition, the function generator was used in conjunction with predefined parameters to generate high-quality synthesized data. For instance, each class subset was passed to a function with m = 1 (the number of synthetic versions of the observed data) and k (the number of cases in the synthesized data), taking different values according to the size of the synthetic data to be generated. The variables inherited by the subset from the common feature set are essentially preserved during generation because other subsets share the variables (data co-location).

Approach 2: Using Feature Perturbation

This feature perturbation approach was applied at the augment stage of the SAC technique. The data perturbation involves the introduction of variations or disturbances, such as noise, into a dataset. The purpose of adding noise and augmenting the data is to create a more robust and diverse dataset, which in turn improves the generalization and performance of the algorithm Rebuffi et al. (2021). In this approach, a function called *combine_samples* was created. This function takes two samples (sample1 and sample2) and randomly combines the feature values. The combination is achieved by taking the average of the corresponding feature values from both samples. Essentially, for each feature, the function calculates (feature1 + feature2) / 2. The input for the function is two samples, and it generates a new sample by element-wise averaging their features. The objective is to generate synthetic samples by blending the features of existing minority class samples. Additionally, another function called *perturb_features* was also created. This function takes a sample and an additional parameter called magnitude. Gaussian noise is then introduced into the features by generating random numbers with a mean of 0 and a standard deviation equal to the magnitude. The amount of noise added to the features is determined by the magnitude parameter, which controls the standard deviation of the Gaussian distribution. The main purpose of introducing noise is to add variability to the features of the samples, thereby increasing their diversity.

In addition, the original sample is loaded, and the minority class is extracted by filtering. The desired number of new synthetic samples to be generated is determined, and an empty list is initialized to store the generated samples. A loop is initiated to generate new synthetic samples. In each iteration of the loop, two indices are randomly

Algorithm 2 Minority class augmentation through data perturbation

Input: $Minority\_class$
Output: $Oversampled\_minority\,class$
functn $combine\_samples(sample1,\ sample2)$
$new\_sample = (sample1 + sample2)/2$
return new_sample
functn $perturb\_features(sample,\ magnitude = x)$
$noise = random(0, magnitude,\ sample.shape)$ ◁ Gaussian noise introduce
$perturbed\_sample = sample + noise$
return $perturbed\_sample$
$NewSamples = n$
$NovelSamples = [\ ]$
for $i\ in\ range(NewSamples)$: **do** ◁ Randomly select two samples
    $X1,\ X2 = rand(minority\_data.shape[0], size = 2, replace = False)$
    $sample1,\ sample2 = minority\_data[X1],\ minority\_data[X2]$
    $new\_sample = combine\_samples(sample1, sample2)$ ◁ Combine samples
    $perturbed\_sample = perturb\_features(new\_sample)$ ◁ Perturb features
    $NovelSamples.append(perturbed\_sample)$
end for
$NovelSamples = array(NovelSamples)$ Convert the list to a numpy array
$oversampled\_minority\_class = vstack((minority\_data,\ NovelSamples))$
$New\_data = concat(majority\_class,\ oversampled\_minority\_class)$ ◁ new data

selected without replacement (replace=False) from the range of indices corresponding to the length of the minority class array. These indices represent two different samples from the minority class. The selected samples are then combined using the previously defined *combine_ samples* function to create a *new_ sample*. Subsequently, the new_sample is perturbed using the *perturb_features* function. The perturbed sample is added to the list of *novel_ samples*. Upon completion of the loop, the list of *novel_ samples* is converted into an array. The original minority class data and the newly generated synthetic samples are then vertically stacked and concatenated to create an oversampled minority class dataset. This oversampled dataset can now be used for further processing.

In summary, this approach creates new synthetic samples for the minority class by combining the features of existing samples and introducing random noise. This is very useful for addressing class imbalance in a dataset, especially when the minority class is underrepresented. The steps for achieving the perturbation techniques are provided in Algorithm 3, and the implementation is in Section 5.5.

In summary, the Augment-Combine (SAC) technique was used to address the problem of class imbalance in datasets. The data augmentation approach uses synthetic data to oversample the minority class(es). The technique was applied to both binary and multiclass datasets. The algorithm for the processes used in the SAC data augmentation is outlined in Algorithm 2, and the implementation is in Section 5.4.

Algorithm 3 Minority Oversampling through Synthetic data using Sort-Augment-Combine (SAC)

Load $D \leftarrow dataset$
Sort $dataset(D)$ $into$ $classLabels(x_i,\ x_{i+1}, , , n)$
for i in $1 : ncol(x_i,\ x_{i+1},\ n)$: **do**
    Load x_i
    $\bar{x_i} \leftarrow G_x(x_i)$ $\triangleleft$ generate synthetic data using synthetic function generator
    Augment $A_i \leftarrow (x_i + \bar{x_i})$
end for
Combine $augClassLabels \leftarrow (A_{augmented} + B_{augmented} + C_{augmented})$
Combine $New_{data} \leftarrow (augClassLabels + D)$
return(New_{data})

3.4.2 Pseudo-label data augmentation

This was another data augmentation approach adopted to enhance classification. This is a novel approach that is done as an inline process during the fitting of a semi-supervised learning model. This approach is necessary as the SAC algorithm mentioned earlier cannot be effectively implemented during the model's training at this point. In this approach, the dataset was split into training and testing subsets to ensure that there was no data leakage. Subsequently, the training data were further partitioned into three subsets: X_labeled, y_labeled, and X_unlabeled. Then, a model, denoted as mod_L, was trained on the labeled data (X_labeled and y_labeled) for 200 epochs, employing a batch size of 64. Following on, the model (mod_L) was then used to predict the X_unlabeled data, thereby generating pseudo-labels known as y_pseudo. To create a combined dataset for training, both X_labeled and X_unlabeled data were concatenated, and similarly, y_labeled and y_pseudo were concatenated and shuffled to ensure a balanced mixture of labeled and pseudo-labeled samples, thus enhancing generalization and classification. The implementation is in Section 7.3.

3.5 Dimensionality Reduction

Feature extraction is an approach used to compress data and reduce its dimensionality. According to Sayood (2017), data compression involves representing information in a concise form by identifying and using internal data structures while retaining the information content. In this context, a neural network is employed to compress a common feature dataset. The purpose of compression is to address the resource requirements of deep learning models, which often pose challenges in IoT devices because of their significant computing, memory, and power demands. To tackle this challenge, the Long Short-Term Memory Autoencoder (LSTM-AE) approach is used for data compression. Various feature reduction techniques exist, including Principal Component Analysis (PCA), Factor Analysis (FA), Linear Discriminant Analysis (LDA), Singular Value Decomposition (SVD), t-Distributed Stochastic Neighbor Embedding (t-SNE), and Auto-encoder. However, the choice of LSTM-AE is based on the fact that it is very effective for capturing temporal dependencies and patterns in sequential data while also learning compact representations of the input data. In the context of intrusion detection or cybersecurity, particularly, an LSTM autoencoder can be used to compress and reconstruct sequences of network traffic data, enabling the detection of anomalies or malicious activities based on deviations from normal behaviour patterns. In addition, time is a critical issue in cybersecurity. Past occurrences play a significant role in comprehending present and potential future threats. LSTM addresses such challenges as long-term dependencies issues, allowing for a more comprehensive understanding of temporal patterns. Thus, utilising an LSTM autoencoder aims to facilitate the analysis of historical events over time, enhancing the model's ability to capture temporal dynamics effectively. Therefore, after the initial implementation of the PCA technique and the issues arising from the curse of dimensionality (Chapter 5), the LSTM autoencoders were adopted because they not only reduce the dimensionality of input data and minimize redundancy but also preserve data density and structure. The architecture of the LSTM autoencoders includes a small bottleneck layer, which compels the network to learn a concise representation and generate an intermediate feature vector for each data point. This representation enables flawless retrieval of the original data points. One advantage of LSTM autoencoders is their ability to overcome the "long-term dependency problem" commonly encountered in neural networks. This problem refers to the difficulty faced by traditional neural networks, including regular autoencoders, in retaining and using information across extended sequences or periods. LSTM autoencoders address the long-term dependency problem by learning the structure of sequences in a dataset over multiple time steps. This enables

them to retain information for prolonged periods. Consequently, they can train and encode a concise representation at the bottleneck layer. According to Baldi (2012); Le et al. (2015), LSTM autoencoders involve two fundamental operations. First, the input data are compressed into a lower-dimensional representation using the encoder. Then, the original input is reconstructed from this lower-dimensional representation using the decoder.

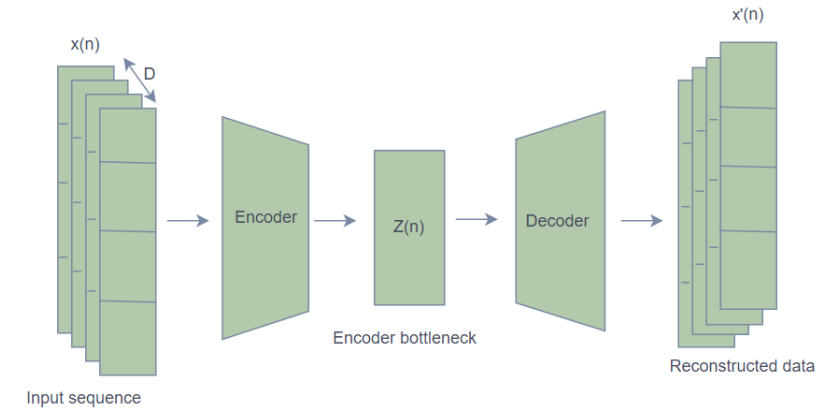
During this phase of the project, the primary focus was on data compression, which is a critical function of the encoder. Therefore, utilizing the encoder's capabilities, the dataset consisting of the common features identified during the feature selection phase was inputted into the LSTM-AE (Long Short-Term Memory Autoencoder). Initially, the output of the bottleneck layer in the autoencoder was limited to nine nodes before being further restricted to five nodes, providing a concise representation (refer to Figure 3.3 (a and b) for a visual depiction of the LSTM-Autoencoder architecture and the structure of the gates). Interestingly, while there was variation in the output of the encoder, there were minimal differences in model accuracy, but there was a substantial reduction in model size and run-time.

Mathematically representing an LSTM-AE data compression model can be complex because it involves multiple layers, each containing LSTM cells. Each LSTM cell is equipped with three gates: the input gate (i_t), output gate (o_t), and forget gate (f_t), which regulate the flow of information within the cell. The input gate determines how much new information should be added to the cell state, the forget gate determines how much of the previous state should be ignored, and the output gate controls the amount of the current state to be outputted. As shown in Figure 3.4, the *tanh* and *sigmoid* functions are used as activation functions. The hyperbolic tangent *tanh* and the *sigmoid* function enable LSTM to effectively control the flow of information and sequential data. They do this by allowing the network to selectively remember or forget information over time using the input, forget, and output gates.

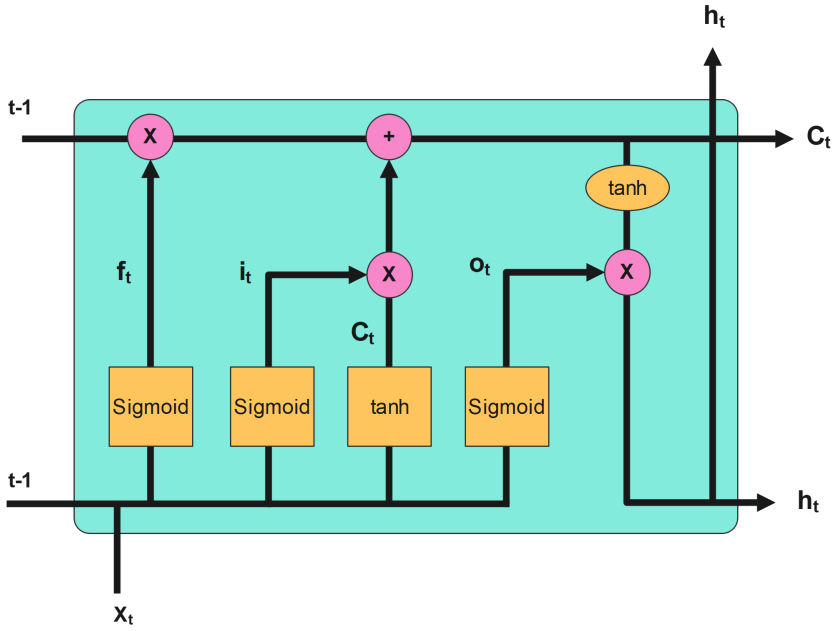
The cell state of an LSTM-AE is updated using the following equation:

$$C_t = f_t * C_{t-1} + i_t * g_t \tag{3.18}$$

where C_t is the cell state at time, t, f_t is the forget gate at time, t, i_t is the input gate at time, t, and g_t is the input modulation at time, t. The output of an LSTM cell is computed using the following equation:



(a)



(b)

Figure 3.3: Figure showing LSTM Autoencoder Trinh et al. (2019)

$$h_t = o_t * tanh(C_t) \tag{3.19}$$

where h_t is the output of the cell at time, t, o_t is the output gate at time, t, and $tanh$ is the hyperbolic tangent activation function.

3.6 Pruning and Deparameterization

3.6.1 Pruning

Considering the goals and objectives of this research, this thesis also aims to emphasize the utilization of necessary approaches to remove less effective connections and zero values that may have been introduced in order to achieve the desired outcomes. This has become necessary because model training often comes with increased computation and parameter storage costs Li et al. (2016). Pruning and deparameterization were employed to address these overhead costs while maintaining classification accuracy Zhu and Gupta (2017). Pruning was utilized to reduce these costs by compressing the weights of various layers without compromising original accuracy. Consequently, the size of the neural network model was reduced by removing less important connections. As a result, a more efficient model was obtained without significant loss in performance. In this study, the Python library *TensorFlow-keras-cloned-model* was utilized to clone the model and retain only 50% of the weights. The steps for pruning are outlined below:

1. **Clone the Original Model:** First, a clone of the original neural network model was created and stored in a variable. The new model has the same architecture and configuration as the original model.

2. **Weights to the Pruned Model:** Then the weights (parameters) from the original model are copied to the pruned model. This essentially initializes the pruned_model with the same weights as the original model.

3. **Define Pruning Parameters:** Thereafter, the pruning parameters are defined using the function, *pruning_schedule* which specifies the sparsity (fraction of weights pruned) to be pruned beginning from a step. In this case, the constant sparsity was set to 50% (0.5) with pruning starting from step 0. This is with a view to reducing the weights and enhancing the effectiveness of the learning algorithm. Also, the 50% was arrived at after several trials and, more importantly, to ensure that the right weights were used subsequently.

4. **Apply Pruning to the Model:** After setting the pruning parameters, it was time to apply them, and magnitude-based weight pruning was performed on the cloned model (pruned_model) using the specified pruning parameters. With this setting, weights with the lowest magnitudes using the pruning schedule based on the sparsity level were determined and set to zero (0), thereby reducing its overall size while retaining important features.

Mathematically, weight pruning can be represented as follows:

Let W be the weight matrix of a neural network with dimensions (m, n), where m represents the number of neurons in the dense layer and n indicates the number of neurons in the current layer.

Let θ be the pruning threshold, a value between 0 and 1 that determines the percentage of weights to be pruned. For example, if $\theta = 0.5$, then 50% of the weights will be pruned. To perform weight pruning, the weights in W_{ij} are ranked based on their magnitude from smallest to largest. The lowest w_{ij}% of weights are then removed and set to zero, and in this case, 50% of it was set to zero. The weight pruning operation for a specific weight w_{ij} in the weight matrix W can be defined as:

$$w_{ij} = \begin{cases} 0, & \text{if } |w_{ij}| < \theta \\ w_{ij}, & \text{otherwise} \end{cases} \tag{3.20}$$

Where:

- w_{ij} represents the weight connecting neuron i to neuron j in the dense layer.
- θ is the pruning threshold.

From equation 3.20:

- If the absolute value of the weight w_{ij} is less than the pruning threshold θ, the weight is set to zero, effectively pruning the connection between neurons.
- If the absolute value of the weight w_{ij} is greater than or equal to the pruning threshold θ, the weight remains unchanged.

Equation 3.20 can be applied iteratively across the weight matrix W, resulting in a pruned weight matrix where many connections below the pruning threshold (θ) have

been set to zero (0).

Pruning helps to reduce memory requirements slightly, accelerates inference time, and makes the network more efficient; however, the sparsity (0) values introduced by pruning are also unnecessary weights. This is because, upon the validation of the pruned model, it was discovered that the model size was still large (refer to Tables 7.3 & 7.4 in Chapter 7). Therefore, stripping the structured sparsity patterns introduced during the pruning process was necessary to reduce the memory footprint further. This is achieved through the deparameterization of the model.

3.6.2 Deparameterization

Deparameterization is a method that removes or reduces specific weights and biases associated with neural network connections Huang et al. (2022). Following the pruning of magnitude-based weights, which sets some weights to zero, it becomes essential further to eliminate these weights for a more streamlined model. In essence, the pruned_model, even after pruning, retains information related to pruning, necessitating the creation of a deparameterized_model. The deparameterized_model mirrors the architecture of the original model but lacks pruning-related parameters. This parameter reduction, particularly those introduced by sparsity, accelerates inference time, which is crucial for real-time applications where response time is paramount. TensorFlow's *Keras-pruning* method was employed for deparameterization, not only diminishing memory requirements but also simplifying subsequent optimization steps and enhancing the manageability of the deparameterized model. The integration of pruning and deparameterization effectively addressed the issue of high model size while maintaining or even improving performance. Figure 3.5 illustrates a schematic of the model transformation from the original model through the pruned model to the deparameterized model.

3.7 Model Training, Adversarial Attacks and Inferencing

For model training, two approaches were used, namely, traditional Machine Learning (ML) and Deep Learning (DL).

3.7.1 Machine Learning

While investigating and fitting the model for the classification of attacks, some machine learning algorithms were used for model training and evaluation. The algorithms used

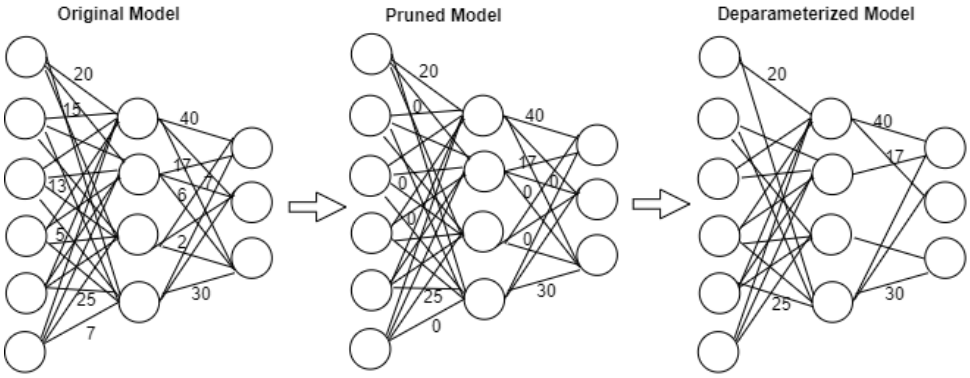


Figure 3.4: Figure showing original, pruned and deparameterized model.

include: Random Forest (RF) Breiman (2001), Linear Discriminant Analysis (LDA) Xanthopoulos et al. (2013), and Support Vector Machine (SVM) Pisner and Schnyer (2020). The choice of these machine learning approaches is based on the fact that they can handle high-dimensional data. In addition, they are also robust to overfitting. Furthermore, random forest and LDA are powerful techniques for dimensionality reduction, which therefore provide the incentive for the techniques to use their internal mechanism to select the best features and output a better classification.

3.7.2 Shallow Deep Learning (SDL)

A shallow deep learning dense model Ge et al. (2021); Meir et al. (2023), also referred to as a fully connected neural network, was employed in this study. Its architecture features a limited number of hidden layers with interconnected neurons. Each neuron in a layer is linked to every neuron in the preceding layer, and collectively, these neurons contribute to the model output. Given that the primary objective of this research is to develop a lightweight Intrusion Detection System (IDS) that is computationally efficient and cost-effective, the adoption of a Shallow Deep Learning (SDL) model was deemed imperative for the following reasons:

1. **Simplicity and Interpretability:** Shallow models exhibit a simpler architecture with fewer parameters than deep neural network models. This simplicity renders them suitable for less complex data because the reduced layer complexity facilitates easier comprehension and interpretation of learned representations, enhancing decision-making processes within the model Gilpin et al. (2018).

2. **Computational Efficiency:** Training shallow models incurs lower computational

costs than deep learning models. This advantage is particularly valuable when dealing with devices possessing limited computational resources, where faster training times are essential Janiesch et al. (2021).

3. **Data Efficiency:** Shallow models can perform well on datasets with limited samples, effectively compressing reduced datasets. In contrast, deep models excel with large datasets, leveraging their capacity to learn intricate representations and generalize effectively Janiesch et al. (2021).

4. **Avoiding Overfitting:** Shallow models are less prone to overfitting because of the simplicity of the model. Conversely, deep learning models with numerous parameters may memorize noise in training data, leading to overfitting Pasupa and Sunhem (2016).

5. **Reduced Risk of Vanishing or Exploding Gradients:** Shallow models are less susceptible to vanishing gradient problems because they have fewer layers. In contrast, deep learning models may encounter vanishing gradient challenges during backpropagation, thus impeding training Gustineli (2022).

6. **Resource Constraints:** Shallow models are highly practical in scenarios with resource constraints, such as memory or processing power limitations Tian et al. (2019).

There are so many constraints that the IoT devices and among the constraints are: Limited Processing Power, Restricted Memory, Physical Size and Form Factor. Therefore, it is important to note that the decision to use a shallow deep model was made because the data has been reduced to a less complex form that fits the available computational resources.

The shallow learning model comprises three layers. The input and hidden layers conducted the bulk of the computations, whereas the output layer served as the final layer responsible for generating the model's predictions. The output of the deparameterized model served as the input for training the dense model. Within the dense layer, each neuron underwent a linear transformation of its input, followed by the application of a non-linear activation function. The linear transformation entailed computing a weighted sum of inputs from the preceding layer, with each input multiplied by an adjustable weight parameter. The outcome of this linear transformation was then passed through the activation function, introducing non-linearity into the model. The Rectified Linear Unit ($ReLU$) activation function was applied to both the input and hidden layers. This

function sets negative inputs to zero and leaves positive inputs unchanged. For the output layer, the *sigmoid* activation function was utilized, mapping inputs to a range of 0–1 suitable for binary classification. A simple structure of a Shallow Deep Learning with the input layer, hidden layer and the output layer is shown in figure 3.6

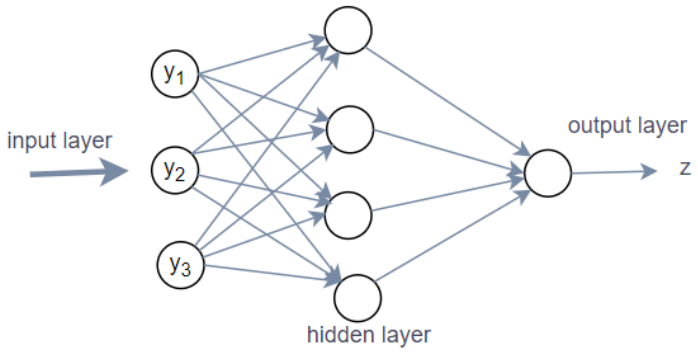


Figure 3.5: Figure showing a shallow deep learning

Mathematically, the dense model can be represented as a function that takes an input vector y and produces an output vector z. The function encompasses L layers, including an input layer, one or more hidden layers, and an output layer. Each layer l in the model consists of N_l neurons and is connected to the preceding layer $l-1$ and the subsequent layer $l+1$. The processes of transformation are indicated in Equations 3.21 - 3.23.

The output of the neurons in layer l is given by:

$$z_l = W_l \times a_{l-1} + b_l \tag{3.21}$$

where W_l is a matrix of weights with dimensions (N_l x N_{l-1}), a_{l-1} is the output of the previous layer, and b_l is a vector of biases with dimensions (N_l x 1). The operation * denotes matrix multiplication.

The output of the neurons in layer l was then passed through an activation function g_l to introduce non-linearity into the model. The output of layer l is given by:

$$a_l = g_l(z_l) \tag{3.22}$$

The output of the final layer L is the output of the training model y, which is a function

of the output of the previous layer a_{L-1}:

$$y = f(a_{L-1}) \tag{3.23}$$

where f is the activation function of the output layer.

However, during the training process of the data using the dense model, the model undergoes weight and bias updates using an optimization algorithm and a loss function to minimize the error. Therefore, the Adam optimizer and binary cross-entropy were used as the loss function to create stability. The loss function quantifies the difference between the model's predictions and the true outputs, while the optimization algorithm adjusts the weights and biases to decrease this difference. To achieve a fair amount of training to enhance the effective learning process, the model was trained and evaluated using an epoch of 300 and a batch size of 64. After the training stage, pruning was performed to reduce the model size and unnecessary weights.

3.7.3 Bayesian Optimisation for Hyperparameter Tuning

Bayesian optimization is an effective strategy for hyperparameter tuning in deep learning models Garnett (2023). It employs Bayesian inference and optimization techniques to systematically explore and discover optimal hyperparameter configurations. In contrast to conventional methods such as grid search, Bayesian optimization optimizes the time and computational complexities associated with tuning numerous hyperparameters. Its noteworthy adaptability in accommodating varying hyperparameter requirements for accurate predictions across different datasets aligns well with the diverse nature of this study. This adaptability facilitates the exploration of different hyperparameter configurations, thus maximizing the model performance. The study focused on tuning critical hyperparameters, including learning rate, batch size, number of layers, number of neurons, regularization, and activation functions, which are all pivotal for achieving optimal model performance. Significantly, the process of finding the most suitable hyperparameter configurations resulted in minimized cost and loss through weight adjustment. Overall, Bayesian optimization provides a systematic and efficient framework for hyperparameter tuning, unlocking enhanced model performance while minimizing computational complexities. Refer to Algorithm 4 for a detailed explanation of the steps from the data compression technique to model training.

Algorithm 4 for data compression and model fitting

Input: $Common - features$, C_f, $Sequence$, $pruning$, $Deparameterization$
Output: $Encode_{data}$
for $each\ X_j\ in\ X_{Train}$: **do**
  while $x_i \leq time - step$: **do** *input layer of LSTM-AE*
    $AE \leftarrow AutoEncoder(X_{Train})$
    $C_t = f_t * C_{t-1} + i_t * g_t$ *calculate* C_t $(Eqn.6)$
    $h_t = o_t * tanh(C_t)$ *update state* h_t $(Eqn.7)$
    $Encode_{data} \leftarrow compress\ AE\ to\ 5\ nodes$
  end while
end for
$train\ Encode_{data}$
while $Mod_{size} = high$: **do**
  $Pruned_{mod} \leftarrow Prune(Mod)$
  $Depar_{Mod} \leftarrow Deparameterize(Pruned_{mod})$
end while
for $y\ in parameters$: **do**
  $perform(BayesianOptimisationSearch)$
end for
$Train(training_{data})$

3.7.4 Adversarial attacks

Adversarial attacks manifest in various forms, including poison attacks, in which an attacker manipulates the training data to mislead the learning algorithm and induce incorrect classifications. This type of attack significantly impairs system performance Paudice et al. (2019). Addressing such attacks and ensuring model robustness requires effective training and learning strategies. To counter label poisoning attacks, Peri et al. (2020) proposed a deep k-NN approach, while Aghakhani et al. (2021) introduced a scalable and transferable clean-label poisoning attack. Similarly, Zakariyya et al. (2023) proposed a non-perturbation approach that relies on the efficiency of resource-efficient models to withstand label adversarial attacks. In this study, two approaches were integrated into the training process to mitigate the impact of label poisoning: (a) the use of an outlier detection method within the robust learning technique. This method identifies and removes poisoned examples from the training set by employing the *EllipticEnvelope* class from the *scikit-learn* library Kramer and Kramer (2016) in Python for outlier detection. (b) Robust training through the generation of pseudo-labels to enhance the training process of our proposed Lightweight Intrusion Detection System (LIDS) (refer to section 3.4.2). This approach facilitates effective generalization.

3.7.5 Quantisation

Quantization is a crucial technique in deep learning that decreases the memory and computation demands of neural network models while maintaining accuracy Liang et al. (2021); Wu et al. (2020). This process involves mapping continuous value ranges to discrete values, thereby reducing the number of bits required to represent each parameter or activation in the network. Quantization comes in three forms: float 8 bits, float 16 bits, and float 32 bits.

The process of float 8 quantization involves reducing the precision of floating-point numbers to 8 bits. This means each numerical value is represented using 8 bits, significantly reducing memory usage compared to the conventional 32-bit floating-point representation. Float 8 is commonly applied in scenarios with stringent memory limitations, such as implementing models on edge devices or systems with restricted resources. On the other hand, Float 16 quantization employs a 16-bit format for encoding each floating-point value, offering improved precision compared to Float 8 while maintaining memory efficiency. Float 16 is often used in situations requiring a balance between precision and memory efficiency, making it suitable for mobile devices or applications with moderate resource constraints. The standard floating-point representation, Float 32, uses 32 bits for each numerical value, providing superior precision but at the cost of increased memory requirements. Float 32 is commonly used in the training stage of machine learning models, especially when high precision is essential. In contrast, the default byte quantization approach explores optimization options using float 8 bytes, float 16 bytes, and float 32 bytes, selecting the most effective optimization approach. The TensorFlow Lite model is employed to preserve the model weight. Quantization can be performed after model training or during training (Quantization Aware Training - QAT). QAT was chosen over post-training quantization to seamlessly integrate the quantization process into training, optimizing the model for quantization from the outset and avoiding potential accuracy issues associated with post-training quantization.

The deparameterized model underwent training and subsequent saving. Afterward, the preserved deparameterized model was converted into a TensorFlow Lite model, employing the float Default Byte quantization approach. The *tensorflow-model-optimization* library Liao et al. (2022) facilitated this process by incorporating a quantize_model function to generate a quantization-aware version of the Keras model (model). The quantization-aware model underwent compilation, using Adam as the optimizer, binary_crossentropy as the loss function, and accuracy as the metrics. Subsequently, the quantization-aware

model underwent a 30-epoch training on the provided data.

Quantization yields several advantages, including reduced memory and computation demands, minimal accuracy loss, method flexibility, and seamless integration into the deep learning workflow. Following quantization, the model transitions into the TensorFlow Lite (TFLite) format, ready for deployment and inference on the test data through the TensorFlow Lite interpreter, ensuring effective implementation and result evaluation. The quantization process is delineated in Figure 3.6.

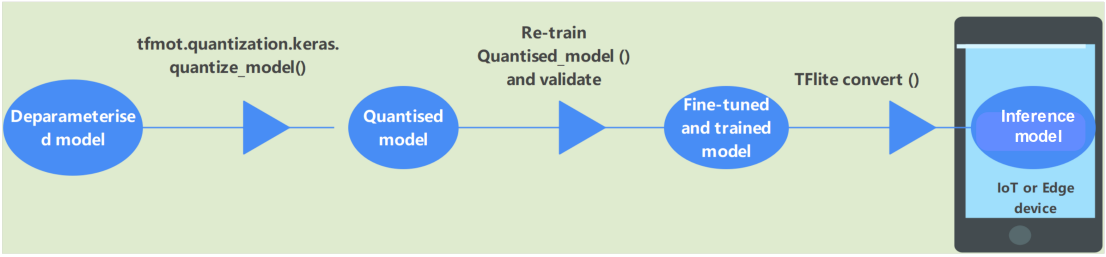


Figure 3.6: Quantization process showing the flow from one process to another.

3.7.6 Model Deployment and Inferencing

To assess the effectiveness of the proposed model, the quantized version was tested on two distinct devices. The first device, a Windows 11 Desktop machine, used Google Colab for Python code execution. This system featured a single-threaded CPU with an Intel Core i7-1065G7 processor operating at 1.30–1.50 GHz, 16 GB of RAM, and a 500 GB hard disk. The second device was a Raspberry Pi 4 with 8 GB RAM running on the Bullseye Debian-based OS with aarch64 architecture (refer to Figure 3.7 for the experimental setup). While the Windows system used Python Notebook as its IDE, the Thorny IDE was employed in the Raspberry Pi experiment. TensorFlow and other essential Python libraries were installed on the Windows Desktop machine, and TFLite, a lightweight TensorFlow version for inference operations, was used on the Raspberry Pi 4. The TFLite model was loaded using the tf.lite.Interpreter class and memory for the input and output tensors were allocated using the allocate_tensor method. Details of the input and output tensors of the lightweight model were obtained using the get_input_details and get_output_details methods. Next, the input data shape was defined, and a random numpy array was generated to simulate the input data. This array was assigned as the input data using the set_tensor method. After configuring the input tensor, the interpreter was invoked using the invoke method to perform inference and obtain the output tensor. A threshold value of 0.5 was applied to the output tensor to convert

the probability scores below and above the threshold into binary predictions of (0 and 1). Subsequently, a classification report was generated by comparing the numpy form of the data with the predicted labels. In addition, the model size and computation time were computed. The steps for quantization, deployment, and inference are outlined in Algorithm 5.

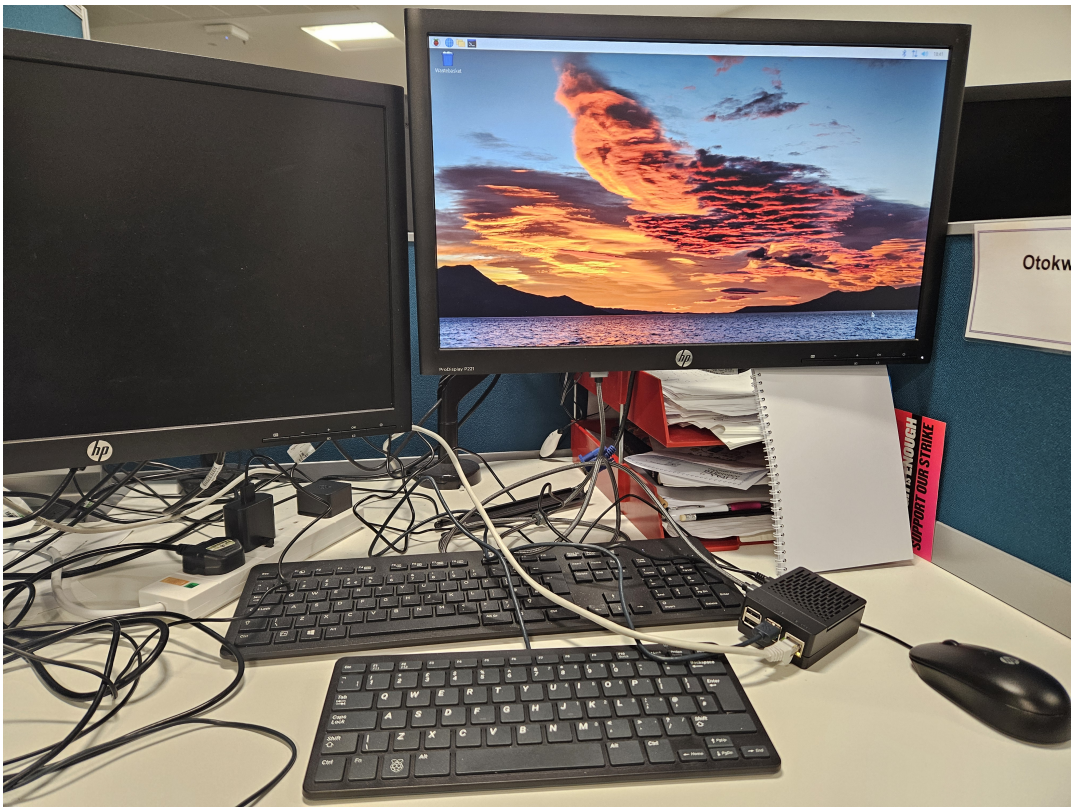


Figure 3.7: Experimental setup consisting of a Raspberry Pi 4 and Windows systems.

Algorithm 5 for quantization, deployment and inferencing

```
Mod ← Model
if Mod = deparameterised then
    Mod.h5                save model
end if
GetModelSize()
Convert to tflite file
if Mod = tflite then
    quantize_model(model)
end if
get input & output tensors
while Tensor ≠ 0 do
    Resize tensor
    interpreter.invoke()
end while
Classification
classification Report()
getValidationTime()
getModelSize()
```

3.8 Performance Metrics

The metrics used for the evaluation of the performance of models in this work are: Accuracy, Precision, Sensitivity, Specificity, F1-score, and Receiver Operating Characteristics.

Accuracy:

This is the average proportion of correctly classified instances (both TP and TN) out of the total number of instances by a model. It is computed thus.

$$Accuracy(ACC) = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.24}$$

Sensitivity (Recall):

Sensitivity or recall is the proportion of TP instances of a class that are correctly predicted. A higher recall rate indicates low FN rate. It is calculated as thus:

$$DetectionRate(DR) = \frac{TP}{TP + FN} \tag{3.25}$$

Precision:

It evaluates the proportion of TP predictions out of all positive predictions made by a model. It is computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3.26}$$

F1-score:

F1-score is also called the harmonic mean of Precision and Recall and it gives a better measure of the incorrectly classified cases than the Accuracy especially for imbalanced classes in a dataset.

$$F1 - score(F1) = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{3.27}$$

Receiver Operating Characteristic (ROC) Curve:

The ROC curve is a graphical representation of a binary classification model's performance. It also shows the Area Under the Curve which quantifies a model's accuracy.

3.9 Ethical Practice

While conducting this research, as outlined in this thesis, strict adherence to the university's Research Governance and Integrity Policy was maintained to uphold and foster ethical conduct throughout the study. Furthermore, no data pertaining to humans or animals was collected. All utilized data were obtained from publicly available sources, and proper attribution to the original authors was ensured through citation.

3.10 Summary and thesis methodological structure

The methodologies outlined in this chapter were designed to fulfil the objectives specified in this thesis. The process commenced with the identification of relevant datasets and the selection of suitable machine learning algorithms for classification. Following this, three feature selection techniques were employed to rank the features. Subsequently, the selected datasets were compressed using the Long-Short-Term-Memory Autoencoder (LSTM-A). Training of the model was performed using a fully connected dense model

before pruning and deparameterization were applied to reduce associated weights. The data further underwent quantization to reduce model weight before inferencing. An overview of the structure of this thesis methodology is provided in Figure 3.8 below.

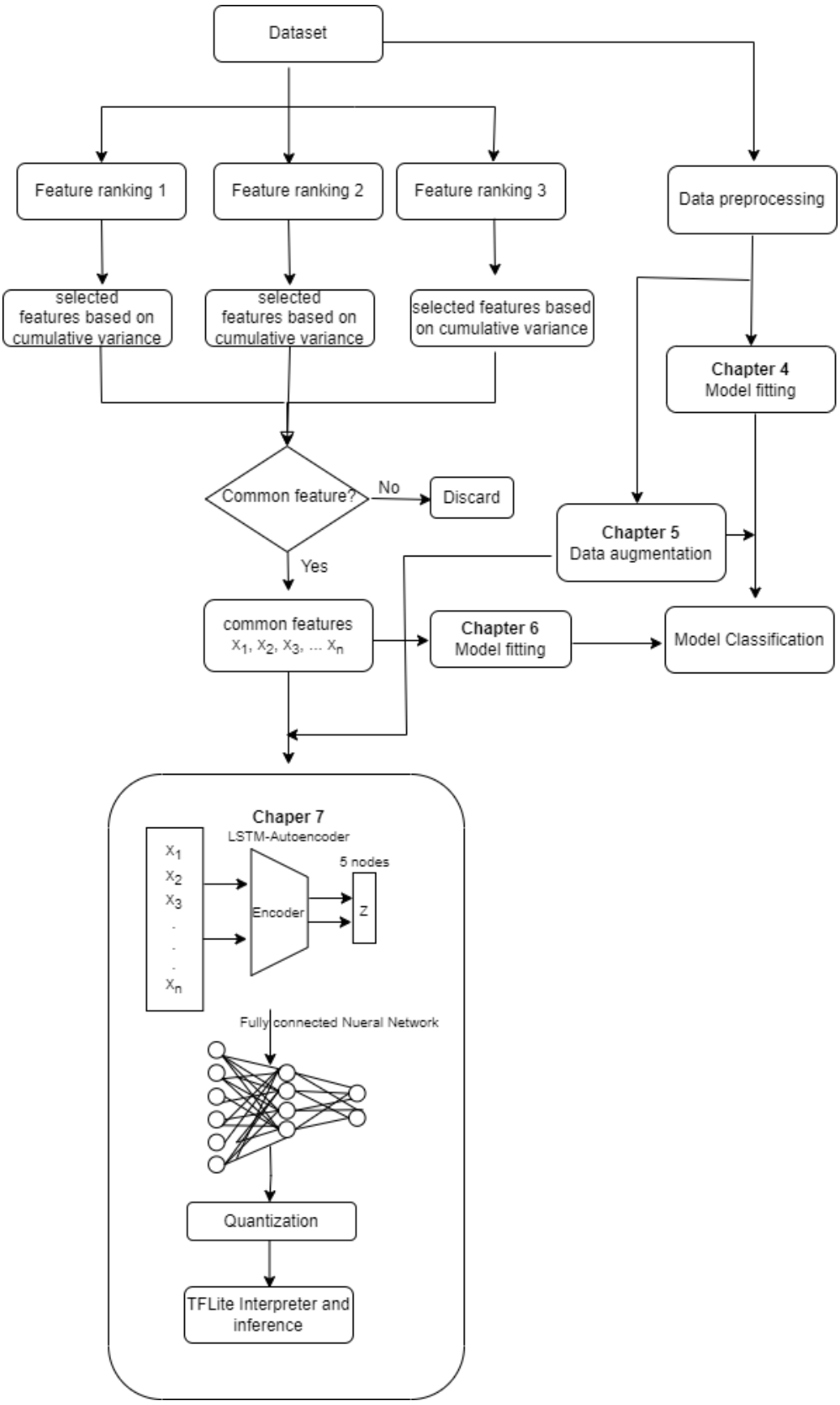


Figure 3.8: A flow of methodologies used in this thesis.

Chapter 4

Effective Detection of Cyber Attack in a Cyber-Physical Power Grid System

4.1 Overview

In Chapter 2 of this thesis, the relevant literature that focused on IoT intrusion detection and lightweight intrusion detection in IoT devices was discussed. The literature review revealed the vulnerabilities in IoT devices and how they have been exploited by both state and non-state actors to compromise the IoT and critical systems. Additionally, the review identified gaps, including the need for effective attack classification in the IoT. In this chapter, IoT attacks are addressed, specifically attacks on intelligent electronic devices (IEDs) in smart grid networks. Building upon the concerns raised in Chapter 2, the use of machine learning algorithms to accurately detect and classify attacks on IoT devices was explored. To enhance the generalization and effectiveness of attack classification, various machine learning algorithms were employed to analyze and evaluate the dataset. The objective of this study was to identify measures that contribute to improved generalization and effective attack classification.

4.2 Introduction

The Purdue model for Industrial Control Systems (ICS) has played a crucial role in merging Information Technology (IT) and Operation Technology (OT) by integrating Wireless Sensor Networks (WSN) and robots. This convergence has significantly advanced the cyber-physical power grid system, commonly referred to as the smart grid. The incorporation of Intelligent Electronic Devices (IED) and other internet-enabled devices into its infrastructure has not only improved monitoring capabilities but has also added substantial value to its operations Escudero et al. (2018). It is noteworthy that the next generation of electric power grid systems, including critical infrastructures, is expected to heavily depend on advanced technologies such as industrial automation control systems, error diagnostics, preventive maintenance, automatic safety switching, advanced metering infrastructure, and synchrophasor systems, as emphasized by Pan et al. (2015b).

Despite these technological strides, the smart grid system faces an increased risk of cyber-attacks aimed at undermining its functionality and disrupting its crucial role in society. Unauthorized users exploit vulnerabilities in devices, such as weak passwords, unpatched firmware, weak encryption, and insecure web links, to gain access to internet-enabled devices within the smart grid system Gilchrist (2017). In some cases, hackers target older firmware versions with known vulnerabilities. The shift from isolated, proprietary software to Commercial-off-the-Shelf (COTS) components in power grid infrastructure systems has exposed them to cyber threats Dondossola et al. (2008); Morris et al. (2011a). Reports indicate that these attacks often succeed due to the insufficient resilience of COTS components and the lack of properly hardened, maintained, or updated safeguards against cyber threats Haber and Haber (2020). In the past, cyber-attacks primarily targeted the IT infrastructure of critical organizations. However, the merging of OT and IT infrastructure has led to a notable increase in cyber-attacks aimed at OT systems Maglaras et al. (2018). These breaches can cause disruptions, such as resetting phasor parameters, system shutdowns, and disturbances to the power grid system. Despite Operating Systems (OS) historically offering abstraction and support mechanisms for hardware and application protection Mollus et al. (2014), cyber-attacks, particularly those perpetrated by non-state actors, have grown increasingly sophisticated. This highlights the critical importance of effectively detecting and preventing cyber-attacks on smart power grid system Conteh and Royer (2016).

4.2.1 Structure of a smart grid system

Figure 4.1 below provide a simple structure of a cyber-physical power grid system with intelligent devices (IoTs) connected.

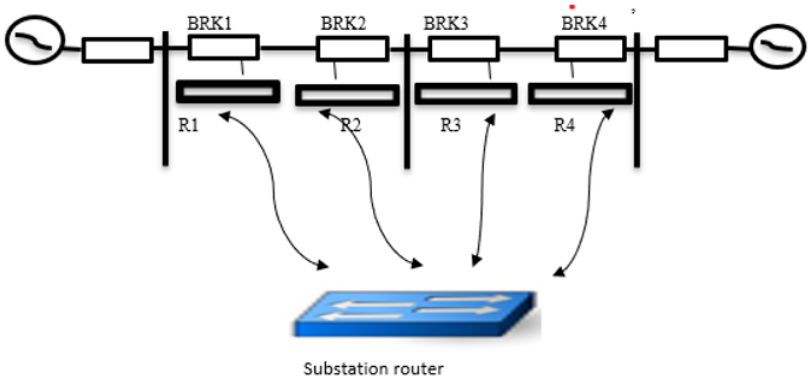


Figure 4.1: Structure of a Cyber-physical Power Grid System Morris and Gao (2014a).

A typical power grid system structure features power generators located at both ends to supply electricity to the grid. Intelligent Electronic Devices (IEDs) labeled as R1, R2, R3, and R4 are connected to circuit breakers BRK1 through BRK4. These IEDs play a crucial role in monitoring grid events and controlling the circuit breakers by either switching them on or off. According to the authors Morris and Gao (2014a), there are two events that can trigger the tripping of circuit breakers:

(a) An alert within the line segments that could prompt the IEDs to initiate the breakers' tripping.

(b) Operators manually issuing a command to the IEDs to break the circuit.

In both scenarios, intelligent devices utilize a distance protection algorithm, allowing the circuit breakers to trip regardless of the command's validity, whether it is a valid or invalid cause. The following is a list of event scenarios derived from the two instances mentioned above that can lead to line tripping. They include Short-circuit faults, Line maintenance, Relay setting changes, and Data Injection. These scenarios produce the dataset used in this study, and it is evident that successful attacks on the power system can lead to its incapacitation, rendering the power grid system incapable of delivering efficient power. Given these vulnerabilities and the limited capacity of the smart power grid system to address cyber challenges Mo et al. (2011), there is a pressing need to detect cyber-attacks and secure the infrastructure of the power grid system.

4.2.2 Objective and contribution

The aim of this study is to develop an efficient cyber-attack detection model by employing various machine learning classifiers on the dataset of the smart power grid system. Subsequently, the obtained results will be thoroughly compared, and the model demonstrating the highest effectiveness will undergo further testing using diverse metrics. The evaluation of the model's performance metrics and the identification of observed gaps in the results will constitute valuable contributions to the enhancement of intrusion detection for cyber-attacks in the smart power grid system, paving the way for future studies in this domain.

4.3 Methodology

4.3.1 Dataset

This study utilizes a dataset sourced from the power system simulation Pan et al. (2015a,b). Comprising 128 variables, of which 128 variables are predictors and a response variable consisting of three classes. The dataset is made up of 52,885 observations and captures measurements of electric transmission within a smart power grid system. These measurements were conducted using four synchrophasors, each measuring 29 features of events in each Phasor Measurement Unit (PMU), resulting in a total of 116 features and 12 derived features. Synchronization of the measurements and features was achieved using a common time source. The dataset categorizes features into attacks and benign data, with benign data including normal traffic and NoEvents obtained through measurements using snort, a simulated control panel, and relays. Parameters such as the voltage phase angle, voltage phase magnitude, current phase angle, and current phase magnitude were measured on the basis of possible scenarios impacting a smart grid. In addition, parameters such as the zero-voltage phase angle, zero-voltage phase magnitude, zero-current phase angle, and zero-current phase magnitude were recorded. Other dataset parameters include frequency for relays, frequency delta, appearance impedance for relays, appearance impedance angle for relays, and status flag for relays. The dataset also provides descriptions of the fault location, line maintenance, and load condition. This comprehensive setup measured both normal traffic transmission in the grid and potential cyber intrusion attacks impacting the power grid system.

4.3.2 Dataset class distribution

To visualize the distribution of the instant classes in the response variable, a barplot of the values is created. The distribution of the classes in numerical terms was Attack (37,851), Natural (Normal) (11809), and NoEvents (3225). See the plot in Figure 4.2.

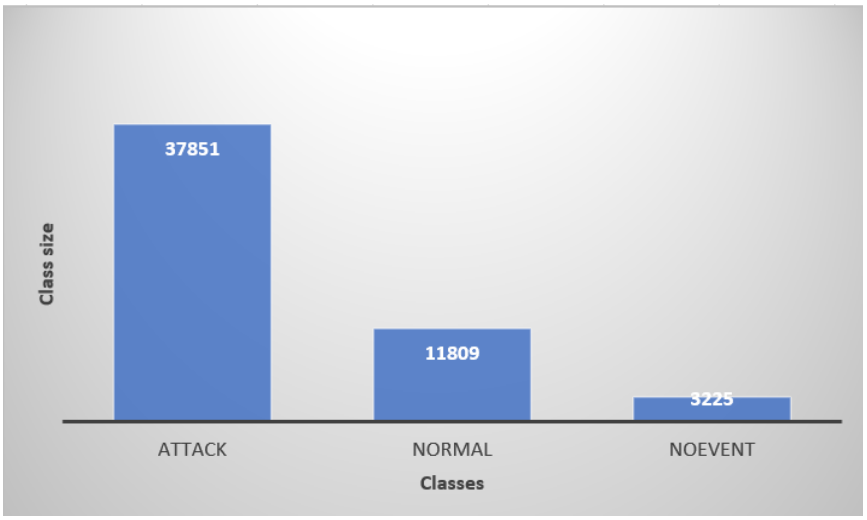


Figure 4.2: Barplot showing the distribution of the instances of the response variable.

From the plot, the malicious class that constitutes the attack is the majority class, with the ratio between the minority and majority classes being: for attack : Natural (1 : 3) and for attack : NoEvent being (1 : 11). The distribution indicates an imbalanced class that may impinge on the classification with a bias for the majority class.

4.3.3 Data pre-processing

Data cleaning and pre-processing are crucial steps in preparing a dataset for analysis. The goal is to ensure that all data points are unbiased and contribute effectively to the model. This involves removing outliers, selecting relevant features, and normalizing the data using the min-max approach in order to ensure that all the features contribute equally to the model output. In our case, min-max scaling was the normalization method for multivariate analysis. Scaling was applied to variables from the first to the 128th, excluding the response variable, which is a factor variable. Once the scaling process was complete, the target variable was appended, followed by the application of the classifier for modeling. During data inspection, outliers were identified and removed outliers in the form of positive and negative infinity (INF and -INF). Before removing them, the dataset was investigated further, and it was discovered that the anomalies were caused

by faults on Line 1's relay, resulting in "Inf" values. Similar outliers were found in Line 2 and relays number 3 and 4 of the power line. In addition, the analysis revealed that the outliers could be attributed to either the disabling of a single relay for line maintenance, a remote tripping command of a single relay, or a fault occurrence. To better visualize the data points that deviated significantly from the others, a boxplot was used to depict the outliers in the dataset Aggarwal and Aggarwal (2017). In Figure 4.3, the outliers, represented as "Inf," were observed in the following variables: "R1.PA.Z," "R2.PA.Z," "R3.PA.Z," and "R4.PA.Z."

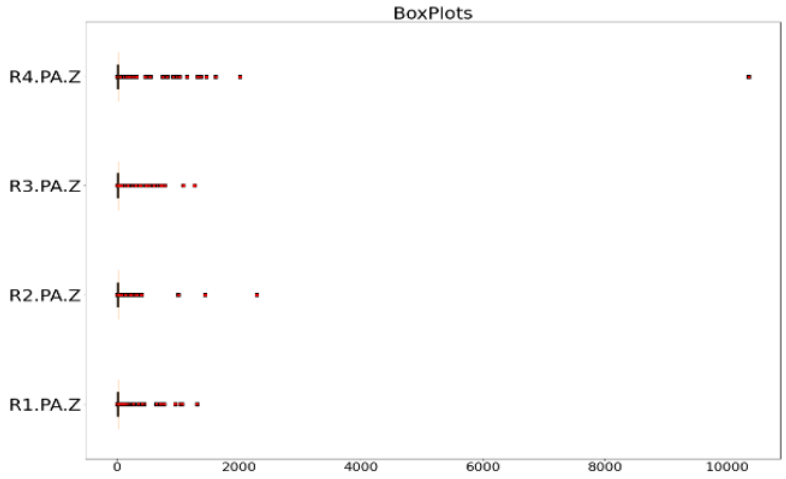


Figure 4.3: Boxplot representation of values and outliers.

Figure 4.3 provides an overview of the range of values in the datasets. From the boxplot, we could observe the outliers that need to be resolved as part of data preprocessing before the fitting of models can be effectively done.

4.4 Model fitting and performance evaluation

In this stage of the experiment, a Windows 10 computer with Intel Core i5 processors and RStudio IDE was utilized. Subsequently, various machine learning algorithms were applied to the dataset. The algorithms employed were Linear Discriminant Analysis (LDA), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Random Forest (RF). The purpose of using multiple models was to compare their results and determine which model achieved the highest accuracy, sensitivity, and specificity. Furthermore, both linear and non-linear classifiers were used to observe potential bias toward the majority class in the output of some classifiers. The dataset consisted of 52,885 observations and

129 features. To conduct the analysis, the dataset was divided into training and testing data. Approximately 70% of the observations, corresponding to 37,000 instances, were allocated for training the classifier. The remaining 30% was utilized for validation. Once the data partitioning was complete, the models were fitted using the different classifiers.

4.5 Model result comparison and discussion

The output of each classifier's confusion matrix was computed, and the classes' values were tabulated in Table 4.1. The metrics as contained in the confusion matrices are overall Accuracy, Sensitivity, and Specificity.

Table 4.1: Outputs of the confusion matrix of each of the models

		Sensitivity			**Specificity**		
Model	**Accuracy**	**Attack**	**Natural**	**NoEvent**	**Attack**	**Natural**	**NoEvent**
LDA (%)	71	99	1	6	3	99	99
SVM (%)	72	99	0	4	1	99	99
Tuned-SVM (%)	77	94	28	70	39	65	98
KNN (%)	71	100	0	0	0	100	100
RF (%)	92	98	68	91	73	98	99

The values in Table 4.1 are a condensation of the output of the confusion matrices of the LDA, SVM, KNN, and RF models. The metrics captured were overall accuracy, sensitivity, and specificity. The values show a varying degree of classification rate, which are explained further in sections 4.5.1 through 4.5.5.

4.5.1 Linear Discriminant Analysis (LDA)

The initial classifier applied to the dataset was Linear Discriminant Analysis (LDA) Tharwat et al. (2017). LDA is renowned for its strong linear classification capabilities and its effectiveness in dimensionality reduction. The approach involves partitioning the data space into N distinct regions, assuming a Gaussian distribution with each attribute demonstrating a variance close to the mean. Despite its robust nature, using the default setting, the classifier achieved an accuracy of 71%, accompanied by a noticeable misclassification rate. Notably, while the sensitivity for the attack class was 99%, the sensitivity for the Natural and NoEvent classes was only 1% and 6%, respectively. Similarly, in

terms of specificity, only 3% of the Attack class was accurately classified, compared to the 99% classification accuracy for the Natural and NoEvent classes. This indicates a case of false alarms likely caused by the imbalanced classification.

4.5.2 Support Vector Machine (SVM)

The Support Vector Machine (SVM) stands out as a robust classifier suitable for both regression and classification tasks Scholkopf and Smola (2018). Known for its high accuracy and efficient use of computational resources, SVM employs decision boundaries to categorize data points based on their proximity to a hyperplane. This hyperplane's position and orientation are influenced by the data points, maximizing output and margin. The results from fitting the SVM model using the default setting, as presented in Table 4.1, reveal an overall accuracy of 72%. However, sensitivity values of 99%, 0%, and 4% were achieved for the attack, natural, and NoEvent classes, respectively. This indicates a poor classification performance for classes other than malicious one. Similarly, the model's specificity indicates that only 1% of the malicious class was correctly classified, while the remaining classes showed a 99% classification rate. Therefore, fine-tuning kernel parameters is necessary to improve performance and reduce misclassification rates.

4.5.3 SVM Tuning

Considering the high rate of misclassification and the relatively low accuracy of the SVM (Support Vector Machine) model, there was a need to fine-tune the SVM kernel parameters. The objective was to improve accuracy and reduce the Cost Matrix Scholkopf and Smola (2018). In an SVM, kernels take data points as inputs and produce similarity scores that influence the class boundaries. The proximity of the data points to the hyperplane determines their similarity score. The closer they are, the higher their score. To achieve better classification results with our SVM model, it is necessary to find the right kernel parameter values that provide an optimal measure of data point proximity. To this end, different values for gamma and cost were experimented with, aiming to find the right combination that would yield improved accuracy and recall rates. In addition, various kernels were tested, including radial, polynomial, sigmoid, and linear kernels. After thorough experimentation, the optimal values for gamma and cost were found and set to 0.1 for gamma and the cost parameter to 20 in the radial kernel to produce the best results. With these tuned values, the kernel parameters effectively improved accuracy while slightly reducing the misclassification rate. Overall, the accuracy increased from 72% to 77%. However, despite this improvement, the misclassification rate was still

relatively high, prompting the need to explore other non-linear classifiers. Table 4.1 lists the sensitivity and specificity values achieved during the tuning process.

4.5.4 K–Nearest Neighbour (KNN)

The K-Nearest Neighbor (KNN) Thanh Noi and Kappas (2017) served as an alternative non-linear classifier for modeling the dataset. KNN used Euclidean distance to assess the proximity between individual data points and their neighbors. With respect to the dataset size, the appropriate value for K was determined to be either 192 or 193 (nearest neighbor). Subsequently, using the default setting, the model was trained, and the output of the confusion matrix was recorded (see Table 4.1). The overall accuracy rate of the fitted KNN model was observed to be 71% with a specificity classification value of 100% for the malicious class. Interestingly, a 0% sensitivity value for the other two classes indicates that the imbalance needs to be oversampled at best for effective classification. A 0% specificity value was also observed for the malicious class against the perfect values of 100% for the Natural and NoEvent classes.

4.5.5 Random Forest

The Random Forest (RF) algorithm Van Essen et al. (2012) uses randomly generated decision trees from selected data samples to make predictions, and the best solution is determined through a voting mechanism. The robustness of the forest increases with a larger number of trees by employing ensemble and divide-and-conquer methods for data splitting. Each tree is constructed using an attribute selection indicator. Application of the Random Forest classifier significantly enhanced accuracy to 92% within a 95% confidence interval. The model exhibited improved sensitivity and specificity, making it highly effective in detecting attack instances in the multiclass dataset under consideration. In addition, the balanced accuracy across the three instances was notably high, underscoring the suitability of the classifier for the experiment. Notably, the model excelled in identifying and detecting attack classes, as evidenced by a Kappa value of 82%. The Kappa statistic, which measures agreement while considering chance agreements, falls within the range of 0.81–1.00, indicating perfect agreement. This characteristic is particularly beneficial in subjective assessments or scenarios with imbalanced distribution. See Table 4.1 for detailed output information.

Table 4.2: Confusion Matrix of the Random Forest classifier

		Actual Values		
		Attacks	Natural	NoEvents
Predicted	Attacks (%)	11202	984	56
	Natural (%)	142	2592	6
	NoEvents (%)	9	4	890

4.5.6 Confusion Matrix of the best model (RF)

Upon evaluating the values presented in Table 4.1, it becomes evident that the Random Forest model outperformed all other classifiers, yielding the most favorable results. Furthermore, the RF model demonstrated the lowest misclassification rate compared to all other models, as indicated by the confusion matrix in Table 4.2. The diagonal elements of the matrix correspond to accurate decisions, while the numbers located on either side of the diagonal signify errors, commonly referred to as misclassifications across different classes. In comparison, this result is an improvement when compared with the benchmark result in Pan et al. (2015a,b), which had an overall accuracy of 90.4%.

4.5.7 Recall for the RF model

Recall, also referred to as Hit Rate or sensitivity, is a key metric used to evaluate the performance of a model. It is calculated as the number or proportion of correctly predicted positive values divided by the total number of actual positive values ($TP/(TP + FP)$). False positives represent instances where the model incorrectly classified negatives as positives. From Table 4.2. the recall is 0.98, which indicates that out of all instances that should have been labeled as "Attack," the Random Forest (RF) model successfully identified 98%.

4.5.8 Precision for the RF model

This is a performance metric used to evaluate the accuracy of a classification model. It measures the proportion of true positive predictions (correctly identified positive instances) out of all positive predictions made by a model. From the RF model result, the precision of the class "Attack" is 0.92 (92%), and it signifies the proportion of instances that were accurately predicted.

4.5.9 F–score

The F-score, or F1, serves as another metric for evaluating the accuracy of a classifier, particularly in datasets where the distribution of classes is slightly skewed toward the majority class. Given that the dataset under consideration falls into this category, it becomes essential to calculate the F-score for the Random Forest (RF) model. The F-score is defined as the harmonic mean of precision and recall, making it a widely employed metric for assessing performance in uneven or imbalanced classification problems.

$$\begin{aligned} F1 &= 2 * \frac{Precision * Recall}{Precision + Recall} \\ &= 2 * \frac{92 * 98}{92 + 98} = 2 * 47.45 = 94.9\% \end{aligned} \tag{4.1}$$

An F-score value of 1 signifies that the variance among the class mean aligns precisely with expectations based on within-class variance and not due to random chance. Thus, given our model's F-score approaching 1, it can be deduced that the Random Forest (RF) model effectively classified and detected attacks. Moreover, considering a 95% Confidence Interval with a significance level of 0.05, the computed P-value ($p < 2.2e\text{-}16$) is less than the significance level, indicating statistical significance. This supports the conclusion that the RF model is well-suited for attack detection.

4.5.10 Cut-off value

The ROC curve serves to identify the optimal cut-off value, illustrating the trade-off between true positives and false positives across different threshold levels. In essence, it assesses the hit rate and false alarm rate at various thresholds, as depicted in Figure 4.4. Fawcett (2006).

From Figure 4.4, it can be observed that the accuracy of the RF model tends to increase with an increase in the cutoff values. However, at a maximum threshold value before the default cutoff (0.5), the model was able to achieve the maximum accuracy.

4.5.11 Receiver Operating Characteristic (ROC) Curve

The ROC curve serves as a valuable tool for visualizing and assessing the accuracy of classifier performance, and it remains independent of the class distribution. A ROC curve that trends toward the top-left corner of the graph signifies superior performance. In the

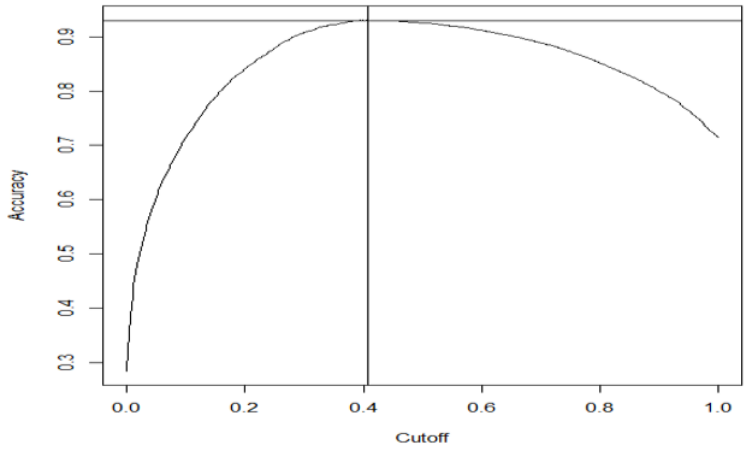


Figure 4.4: Plot showing the overall Accuracy values against several Cutoff values of the RF model.

case of the RF model, the ROC curve in Figure 4.5 demonstrates a tendency toward the top-left corner, indicating the model's proficiency in predicting true positive rates accurately.

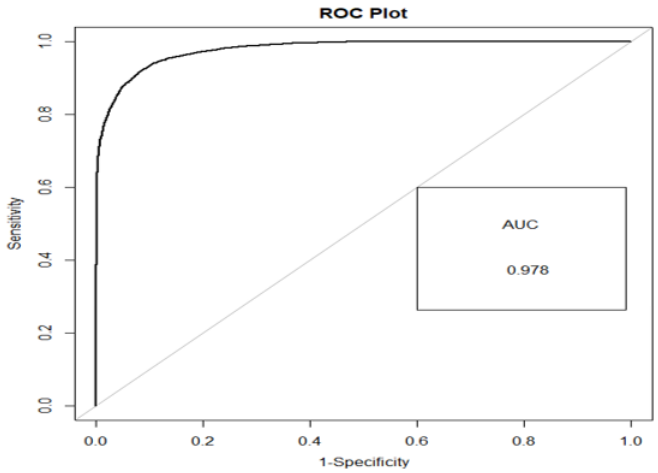


Figure 4.5: Showing ROC graph of sensitivity against 1-specificity and the area under the curve.

4.5.12 Area Under the Curve (AUC)

The ROC graph covers an area of 1, with a scale ranging from 0.0 to 1.0. To assess a model's predictive accuracy, it's crucial to compute the AUC of the curve, representing

the probability that a randomly chosen value is a positive instance of higher rank. An AUC of 0.5 indicates that the ROC curve aligns with the baseline (the diagonal), where False Positive Rate (FPR) equals True Positive Rate (TPR), suggesting less accurate predictions or, at best, detection by chance. However, with an AUC of 0.978 for the RF model, it signifies a higher likelihood of detecting high true positives.

4.6 Conclusion

Addressing the increasing integration and complexity of cyber-physical smart grid systems and other critical infrastructures requires an effective strategy for detection, monitoring, optimization, and, crucially, securing these systems from cyber attacks. This study proposes a robust anomaly detection method designed to safeguard smart power grid systems from cyber threats. Given the varied distributions within the target class of the dataset, multiple machine learning models were employed for fitting, with essential data cleaning and preparation conducted beforehand. The classifiers utilized included Linear Discriminant Analysis, Support Vector Machine, K-Nearest Neighbor, and Random Forest. Interestingly, the Random Forest model consistently outperformed others, exhibiting superior accuracy and a heightened detection rate for true positives, including improved specificity. The RF model's performance was further scrutinized using metrics such as precision, recall rate, F-score, ROC, and Area Under the Curve, all of which consistently supported the RF model as highly proficient in anomaly detection within a smart grid system.

However, the imbalanced class distribution in the dataset posed challenges to effective generalization and classification, as evidenced by zero (0), 1, and other marginal classifications in the output of the RF and other models. Ineffective generalization arises when there is insufficient data for robust learning, resulting in false alarms and low classification rates. To address this issue, the next chapter of this thesis extensively explores the enhancement of intrusion detection through data augmentation of the minority class.

Chapter 5

Improving Intrusion Detection Through Data Augmentation

5.1 Overview

Aligned with the primary objective of developing a lightweight intrusion detection model tailored for the IoT, Chapter 4 delved into the realm of efficiently detecting cyber-attacks on an IoT-enabled smart grid network. Notably, the application of machine learning models revealed that the Random Forest exhibited commendable classification performance. However, the inherent challenge of an imbalanced class distribution within the dataset hindered the effective generalization of the models, as shown in the results in Chapter 4. This chapter refocuses on the overarching goal of achieving computational efficiency by employing robust data augmentation and feature selection methodologies. These strategies bolster data generalization and classification, particularly in scenarios involving imbalanced datasets. Research Question One (RQ1) guides the exploration: *How can effective generalization be achieved in order to improve intrusion detection of attacks on IoT devices?* To answer this pivotal research question, a comprehensive examination of data augmentation and feature selection is presented in this chapter. The emphasis is on fortifying the intrusion detection model's performance, specifically tailored for resource-constrained IoT devices. Leveraging two benchmark datasets representing IoT-enabled smart grid and BoT-IoT datasets, this study taps into real-world scenarios to evaluate the proposed techniques. The core objective is to scrutinize the effectiveness of data augmentation and feature selection in achieving superior classification outcomes

while simultaneously minimizing computation costs.

5.2 Introduction

Data augmentation, as proposed by Tanner and Wong (1987), emerges as a valuable technique for enhancing observed data, thereby fortifying the generalization capabilities of learning algorithms. This augmentation can be applied to the entire dataset or selectively to training data or minority class(es), proving particularly advantageous in intrusion detection classification scenarios. The effectiveness of classification, especially in the context of IoT and critical infrastructure attacks like DoS and DDoS, hinges on the robust and efficient generalization of the learning algorithm. However, these attacks frequently result in imbalanced datasets, posing a challenge to optimal generalization due to the dominance of the malicious class as the majority. In scenarios where the majority-to-minority class ratio exceeds 1:3, imbalanced classification becomes prominent, hindering effective generalization Ramyachitra and Manikandan (2014). While there isn't a definitive minimum ratio for an imbalanced dataset, analysis of the IoT Botnet dataset Koroniotis et al. (2019) suggests that ratios of 1:3 or higher contribute to imbalanced classification. In such situations, the classifier becomes biased towards the majority class during training, leading to imbalanced classification Sun et al. (2009).

Insufficient data during machine learning classifier training hinders generalization, resulting in a bias towards overrepresented majority classes and leading to misclassification LATA (2019); Lemley et al. (2017). This ineffective generalization and misclassification significantly impact intrusion detection models, compromising their ability to detect and prevent attacks. While data augmentation has been successful in various domains like image classification, its effectiveness varies based on the dataset and algorithm application Fadaee et al. (2017); Perez and Wang (2017). It's important to note that using synthetic data with a different distribution complicates the model's ability to analyze and classify data effectively O'Ree and Obaidat (2011). Given these considerations, two SAC (Sort, Augment, and Combine) approaches were employed for the augmentation and oversampling the minority class(es). These include (a) using library-generated synthetic data and (b) utilizing feature perturbation, as discussed in Chapter 3.

5.2.1 Contribution

To address the challenges of class imbalance and low data regime in datasets as highlighted in section 5.2, the contributions of this study are hereby presented as follows:

– A data augmentation strategy for class imbalance in datasets that can be used with both binary and multiclass datasets. This proposed novel data level data augmentation technique employs a Sort, Augment, and Combine (SAC) minority oversampling approach to address the problem of class imbalance in a dataset.

– A synthetic data that is of high quality and has the same distribution as the original data. This is to enhance effective blending and generalization.

– To demonstrate the effectiveness of feature perturbation such that beyond its concept of noise, there is a useful technique for the oversampling of a minority instant class and hence improve generalization and the performance metrics such as sensitivity, specificity, and overall accuracy. This is due to the fact that improved generalization leads to better classification, which is critical for intrusion detection.

5.3 Approach 1 - using SAC library generated synthetic data

The first of the two SAC approaches for addressing the problem of class imbalance in datasets focuses on data augmentation through synthetic oversampling of the minority class(es). The method is based on the Sort–Augment– Combine (SAC) data augmentation technique, which has been clearly discussed in section 3.5.1 with the steps indicated in Algorithm 2. Here, the dataset was split into a ratio of 70:30 training to testing. The training data were then sorted into instant classes before the generation of the synthetic. Subsequently, the concatenation of the subset's instant classes and the original training data was performed using the SAC technique.

5.4 Dataset and Model Fitting

Two datasets were used in this study, and they are the BoT-IoT dataset Koroniotis et al. (2019) and the Smart grid dataset Pan et al. (2015a). Both are multiclass datasets with class imbalance. The justification for using more datasets here is to ensure that the approach is not limited to only a single dataset but can be applied to more datasets. More importantly, the approach requires datasets with imbalanced classes which makes it imperative for more imbalanced datasets to be considered.

5.4.1 Using Smart Grid Dataset

This dataset was produced from a controlled laboratory experiment involving the measurement of electrical signals on transmission lines using synchrophasors. Key parameters, including voltage, current, frequency, cyber-attack impedance, and normal traffic, were measured as part of the experiment Pan et al. (2015a). The dataset was categorized into three groups: binary, triple, and multiclass, each comprising 15 sets of 37 event scenarios. In this study, the triple-class category was selected, resulting in a dataset with 10,035 observations and 128 features. The target feature consisted of three class labels: Attack, Natural, and NoEvents. After performing the necessary cleaning and preprocessing on the data, Principal Component Analysis (PCA) was applied to reduce the feature dimensionality from 128 to 25 Principal Components (PCs). Subsequently, the dataset was partitioned into a 70:30 ratio for training and validation, and the class distribution relative to the largest class in the dataset is presented in Table 5.1.

Table 5.1: Distribution of the instant classes in original and augmented Smart grid data

Class type	Original numb. of Observ.	Ratio	After Augmentation
Attack	6890	1:1	4790
Natural	1919	1:3	4760
NoEvents	495	1:13	4789

From Table 5.1, it could be observed that the relationship between the Attack and Natural classes exhibits a closed to balanced distribution, whereas a slight imbalance is observed in the relationship between the Attack and NoEvents classes. Given this observation, the model was fitted on the original dataset, and also on the augmented Natural and NoEvents classes.

Table 5.2: Distribution of the instant classes in original and augmented Smart grid data

		Sensitivity			Specificity		
Data	Overall Acc.	Attack	Natural	NoEvents	Attack	Natural	NoEvents
Original Data (OD)	91	97	72	85	76	97	99
Augmented Data(AD)	95	90	96	99	98	95	99
Difference between (OD - AD) (%)	4	-7	24	14	22	-2	0

Remarkably, post-augmentation of the minority classes, the classifier achieved a commendable 90% accuracy in correctly classifying the Attack class. Similarly, it demonstrated high accuracy in classifying the Natural and NoEvents classes, achieving 96% and 99%, respectively. However, there was a 7% decrease in the classification accuracy of the Attack class after augmentation, as summarized in Table 5.2. A 4% increase in overall accuracy counterbalanced this decrease. Notably, the increase in the overall accuracy is attributed to the improved classification of benign classes. In the context of intrusion detection, enhanced classification of the benign class contributes to a reduction in false alarms, thereby mitigating Type 2 errors. Furthermore, there is also an improvement in the classification of the benign classes, as shown in Table 5.2, with a 24% in the Natural class and 14% in the NoEvents class. Similarly, for specificity, there is also an improvement in the attack class with an increase in classification by 22% and decreased by 2% for the benign Natural class. Notwithstanding the decrease in classification, a lower specificity indicates a high number of false negatives, which has the propensity to increase false alarms.

5.4.2 Comparing SAC-1, ROSE Augmented, SMOTE Augmented and SAC-1 + ROSE Augmented datasets Using Binary dataset

To assess the performance of the library-generated synthetic augmentation (SAC-1) compared to other widely used oversampling techniques in binary classification, the class distribution in Table 5.1 was transformed into a binary dataset. This involved merging the Natural and NoEvents classes into a single benign class. The resulting dataset comprised attack and benign classes, with counts of 6890 and 2414 instances, respectively, representing 74% and 26% of the total. Subsequently, a model was trained on both the original and augmented datasets. The augmentation of the minority class in the binary dataset employed the SAC-1 synthetic technique, along with Random Over-sampling Examples (ROSE) and the Synthetic Minority Oversampling Technique (SMOTE) Demir and Şahin (2022). The use of these diverse oversampling methods aims to facilitate a comprehensive comparison, shedding light on how the SAC-1 synthetic approach performs in relation to other popular minority oversampling techniques.

The oversampling techniques, ROSE and SMOTE, have been extensively applied to address imbalanced minority classes in binary datasets. For instance, SMOTE addresses the class imbalance in datasets, particularly in machine learning tasks where one class is significantly underrepresented relative to the others. It generates synthetic samples for the minority class by interpolating between existing ones. It creates new instances

along the line segments joining k minority class nearest neighbors. It helps to balance class distribution through the oversampling of the minority class, thereby improving the classifier's performance. However, it may also introduce noise and perform poorly with overlapping classes. ROSE, in a similar way, is used to correct class imbalance through the oversampling of the minority class. It oversamples by randomly selecting the minority class samples and duplicates them to augment the dataset. Unlike SMOTE, ROSE does not generate synthetic samples but replicates existing ones. As a result, it is simple to implement and computationally less intensive compared to SMOTE. However, one of the demerits of this technique is that it does not have the potential to address the problem of overfitting due largely to the duplication of values. Consequently, it becomes crucial to assess the effectiveness of the SAC-1 synthetic approach in comparison to the result of the model on the SMOTE and ROSE oversampled data. To accomplish this, a K-Fold cross-validation using Random Forest (with k=5) was conducted, with the specified parameters outlined below. Subsequently, the model's outputs on the original dataset, ROSE-augmented dataset, SMOTE-augmented dataset, and SAC-1 augmented dataset were compared. The parameters were configured as follows:

TrainControl

This parameter facilitates the specification of the number of repetitions for cross-validation. In this study, the *repeatedcv* approach was used to ensure a consistently repeated training/testing split. The value k=5 was employed for resampling iterations, and the *random* search was utilized as the tuning parameter for the search process.

Train

This parameter configuration is instrumental in facilitating the model fitting process and contributes to refining the tuning for improved outcomes. In this context, the Random Forest (RF) algorithm was employed. A *tuneLength* value of 10 was utilized, along with an "ntree" setting of 1000.

Subset

Fine-tuning this element improves the automatic selection of the *bestTune*, enabling the identification of the optimal tune value from the coefficients. Subsequently, the determined bestTune value is used as the value for *mtry* parameter.

After configuring the parameters, the dataset underwent augmentation using the ROSE

and SMOTE packages. Subsequently, model fitting was carried out, and the outcomes are presented in Table 5.3.

Table 5.3: Comparison of the overall accuracy, sensitivity and specificity of the confusion matrix of original data, ROSE, SMOTE, SAC-1 and SAC+ROSE Augmented data

Dataset	Overall Accuracy	Sensitivity	Specificity
Original data	91	98	71
ROSE Augmented	97	96	98
SMOTE Augmented	94	96	90
SAC-1 Augmented	**93**	**91**	**94**
SAC-1 + ROSE	**98**	**98**	**98**

The findings in Table 5.3 suggest that models trained on data augmented with ROSE and SMOTE techniques were slightly better in overall accuracy and sensitivity when compared to data augmented using the SAC-1 library-generated synthetic technique. Recognizing the strengths of both ROSE and SAC, we decided to merge the augmented datasets from both techniques (SAC-1 augmented + ROSE augmented) to create a new training dataset. Training a model on this combined data resulted in improved overall accuracy, sensitivity, and specificity, surpassing the performance of other models in terms of overall accuracy, sensitivity, and specificity.

5.4.3 BoT-IoT dataset

This dataset is the result of a laboratory simulation of IoT Botnet traffic with various types of attacks. It is relevant to the course of this study, especially given that IoTs are involved in the attack and the potential for IoT devices to be used as bots to attack other devices. According to the authors, this benchmark dataset was developed as a stop-gap measure for cybersecurity researchers and, more importantly, to enhance the understanding of modern evasive attacks. The authors also added that the dataset has gained a wider acceptance over the years because of its advantages over other benchmark datasets because of the following reasons: (a) redundant records leading to biased detection Mahoney and Chan (2003), (b) several missing records as factors in some datasets McHugh (2000), and (c) data unbalancing among constituent observations Tavallaee et al. (2009). The dataset has 82,332 observations and 42 features consisting of 10 classes in the target feature. Table 5.4 and Figure 5.1 show the size and ratio of the classes relative to the largest class (Normal class) and a plot of the distribution of the classes.

The analysis of the class distribution, presented in Table 5.4 and Figure 5.1, highlights

Table 5.4: Original data size, ratio and distribution of instant classes

Class	Number of Observation	Ratio to largest class
Analysis	677	1:54
Backdoor	583	1:63
DoS	4089	1:9
Exploits	11132	1:3
Fuzzers	6062	1:6
Generic	18871	1:2
Normal	37000 (largest class)	1:1
Reconnaissance	3496	1:10
Shellcode	378	1:97
Worms	44	1:840

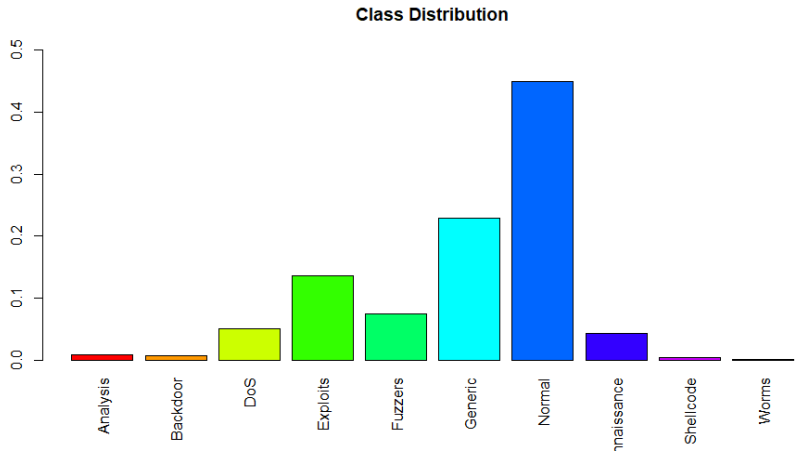


Figure 5.1: showing a bar plot of class distribution in the original dataset

a clear pattern. Normal traffic exhibits the highest number of observations, twice as many as the closest attack classes, namely Generic and Exploits. While the Generic, Exploits, and Fuzzers classes maintain a ratio of 1:2, 1:3, and 1:6, respectively, relative to the Normal class, the remaining classes representing attack types are significantly underrepresented in comparison to the benign class. In addition, a number of the features in the dataset are categorical, so it became imperative to treat the class imbalance with minority oversampling and to also use one-hot encoding to expand the features. This encoding technique increased the dataset's features from 42 to 187. With the expanded feature set, it became necessary to rank the features and retain only those deemed relatively important. The feature importance function of the Random Forest algorithm was employed for this purpose. This function measures the decrease in node impurity (using the Gini index) as the nodes are split. The ranking revealed that the Mean Decrease Gini values ranged from 0.4512 to 2822. On the basis of these ranks, less important features were filtered out, reducing the feature count from 187 to 53. Additionally, Principal Component Analysis (PCA) was conducted to obtain Principal Components (PCs) that captured at least 90% of the variance in the dataset.

5.4.4 Model fitting with original dataset using Random Forest

A Random Forest (RF) model was fitted on the original dataset with K-fold cross-validation (k=5) to provide a platform for the comparison of the confusion matrix when oversampling was eventually performed. The confusion matrix following the fitting of the model to the original dataset is displayed in Table 5.5.

Table 5.5: Output of random forest on the original dataset before minority class augmentation

	Analysis	Backdoor	DoS	Explot	Fuzzer	Generic	Normal	Recon	Shellcode	Worm
Analysis	42	8	8	47	39	0	0	0	0	0
Backdoor	8	0	10	27	22	2	0	3	0	0
DoS	123	45	1737	1939	249	62	20	226	3	1
Explot	316	285	1799	7531	883	350	173	392	45	29
Fuzzer	185	213	237	765	3442	38	635	124	25	1
Generic	0	3	30	58	1	18323	13	5	6	3
Normal	1	24	209	637	1376	83	35957	363	115	5
Recon	0	4	41	200	41	4	179	2373	88	1
Shellcode	0	1	17	28	9	7	22	10	96	0
Worm	0	0	1	0	0	2	1	0	0	4

The confusion matrix of the model on the original data is displayed in Table 5.5. However, it also reveals a significant rate of misclassification. It must be noted that in predictive learning algorithms, it is typically assumed that models classify equally among categories and that prediction errors are consistent across all classes. Interestingly, this assumption only holds in ideal scenarios and does not account for imbalanced class distributions. In such cases, misclassifications tend to result in Type 1 and Type 2 errors Wankhade et al. (2013).

To improve classification accuracy and minimize false alarms, it is essential to address imbalanced data. To tackle this issue, first, Approach 1 of the SAC strategy was implemented to oversample the minority classes. The size of the Generic class was used as the foundation for generating synthetic data. This approach had two main objectives: first, to prevent the model from overgeneralizing during training, and second, to address the relatively balanced ratio between the Generic and Normal classes (1:2), which does not signify a significant imbalance. Additionally, the Generic class exhibited a low misclassification rate of only 3% (refer to Table 5.4). The augmented dataset and the ratios are displayed in Table 5.6.

Table 5.6: Ratio of classes to largest class after augmentation of dataset

Class	Original size	Ratio	Augmentation size	New Ratio
Analysis	677	1:54	18956	1:2
Backdoor	583	1:63	18073	1:2
DoS	4089	1:9	17992	1:2
Exploits	11132	1:3	18367	1:2
Fuzzers	6062	1:6	18186	1:2
Generic	18871	1:2	18871	1:2
Normal	37000	1:1	37000	1:1
Reconnaissance	3496	1:10	18528	1:2
Shellcode	378	1:97	18144	1:2
Worms	44	1:840	18084	1:2

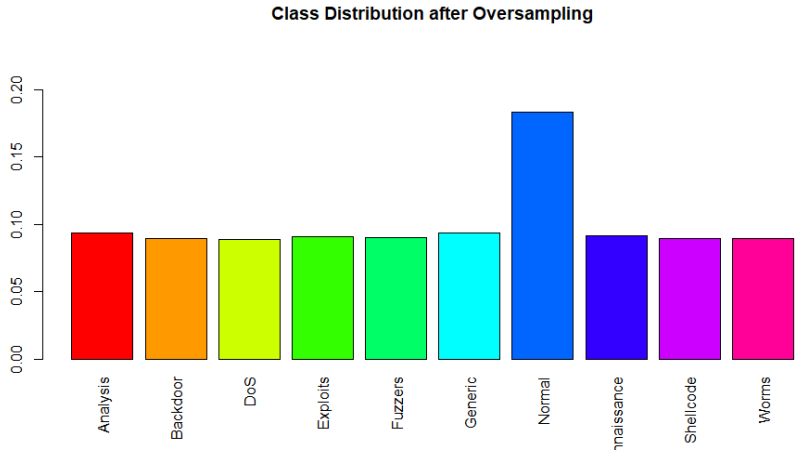


Figure 5.2: Plot of the class distribution of augmented classes

The classes augmented are Analysis, Backdoor, DoS, Exploits, Fuzzers, Reconnaissance, Shellcode, and Worms. Table 5.6 shows the new distribution. Similarly, Figure 5.2 shows the plot of the class distribution of augmented classes.

Furthermore, it was crucial to ensure that the synthetic data values were similar to the distribution of the original dataset before augmenting the minority classes with the generated data. To verify this, a comparison plot between the synthetic data and the shape of the original data was created (refer to Figure 5.3). The plot illustrates a side-by-side analysis of the original and synthetic distributions for the selected classes. The dark color represents the observed original data, whereas the light color represents the generated synthetic data. Notably, the plot demonstrates that the structural distribution

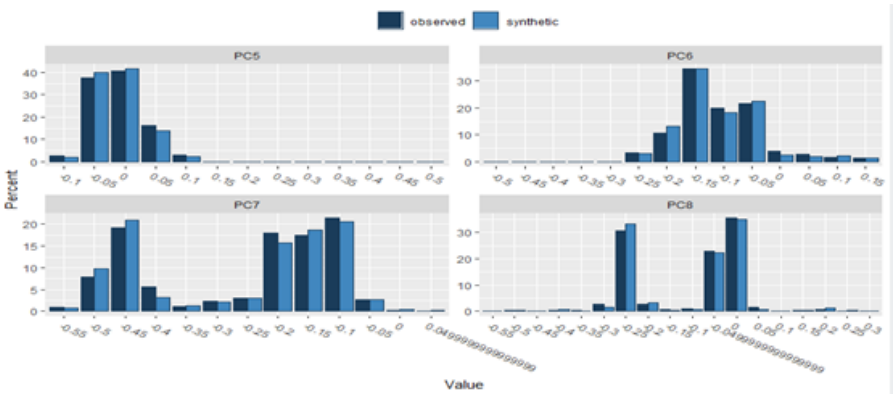


Figure 5.3: A comparison between the original data and the synthetic data.

of the original class was accurately preserved in the synthetic data.

5.4.5 Model fitting with augmented minority classes dataset using Random Forest Model

After augmenting the minority classes, the dataset's number of rows increased from 82,332 to 202,200 while maintaining its 24 features (principal components). Subsequently, it became crucial to fit a model to observe the impact of augmentation. Fitting a model on the augmented dataset aimed to improve generalization and classification, aligning with the objective of addressing the challenges posed by code obfuscation and ensuring robust detection capabilities. The fitting process replicated the approach used for the original data, employing the Random Forest model with K-Fold cross-validation while maintaining other parameters. This consistency in methodology ensures a meaningful comparison between the performance of the model on the augmented dataset and the original data.

Table 5.7: Output of random forest on the original dataset after minority class augmentation

	Analysis	Backdoor	DoS	Explot	Fuzzer	Generic	Normal	Recon	Shellcode	Worm
Analysis	18066	260	350	323	655	4	4	204	0	0
Backdoor	238	17146	312	216	492	3	0	54	0	0
DoS	148	56	14008	2459	398	105	61	441	2	1
Exploit	320	335	2274	12936	970	339	195	315	2	2
Fuzzer	182	170	358	1015	13912	50	1308	379	51	1
Generic	0	2	14	37	2	18277	9	4	5	0
Normal	1	15	182	509	1220	52	34602	272	33	3
Recon	1	27	178	563	332	9	586	16599	126	1
Shellcode	0	62	309	275	201	30	232	258	17925	1
Worm	0	0	6	34	4	2	3	2	0	18075

In Table 5.7, the results of a random forest model on the oversampled minority class dataset are displayed. The results demonstrate the output of the model on the augmented minority oversampled data, which achieved an overall accuracy of 89%. A comparison with the previous output (Table 5.5) reveals an enhancement in classification performance, particularly in terms of sensitivity and specificity (see Table 5.8). This is in addition to the overall accuracy. Given the improved classification, it was necessary to compare the sensitivity and specificity of the original and augmented datasets. The comparison can be found in Tables 5.9 & 5.10.

Table 5.8: Overview of classification and misclassification of classes before and after augmentation.

Class	Augmented Classification	Aug. Misclass	Original Misclass
Analysis	95.3	4.7	94
Backdoor	94.8	5.7	100
DoS	77.8	22.2	58
Exploits	70.4	29.6	33
Fuzzers	76.4	23.6	34
Generic	96.8	3.2	3
Normal	93.5	6.5	3
Reconnaissance	89.5	10.5	33
Shellcode	98.7	1.3	75
Worms	99.9	0.01	91

Table 5.9: Comparison of the Sensitivity of Confusion Matrix of Original and Augmented datasets

Data	Avg. Acc	Analysis	Backdoor	DoS	Explot	Fuzzer	Generic	Normal	Recon	Shellcode	Worm
Original data (OD)	84	6	0	42	67	56	97	97	67	25	9
Augmented data (AD)	89.7	95	94	77	70	76	97	93	89	98	99
Difference btw (OD - AD) (%)	5.7	89	94	35	3	20	0	-4	22	73	90

Table 5.10: Comparison of the Specificity of Confusion Matrix of Original and Augmented datasets

Data	Avg. Acc	Analysis	Backdoor	DoS	Explot	Fuzzer	Generic	Normal	Recon	Shellcode	Worm
Original data(OD)	84	99	99	96	93	97	99	93	99	99	100
Augmented data (AD)	89.7	99	99	98	97	98	99	98	99	99	99
Difference btw (OD - AD) (%)	5.7	0	0	2	4	1	0	5	0	0	-1

Table 5.9 uses sensitivity for comparisons, where sensitivity represents the classifier's accuracy in correctly identifying the positive class (True Positives). In Table 5.8, the misclassification rates for classes like Analysis, Backdoor, Worms, Shellcode, and DoS, were exceptionally high in the original dataset model. However, the misclassifications witnessed a significant reduction to 4.7%, 5.7%, 0.01%, 1.3%, and 22.2%, respectively after the augmentation of the minority classes in the dataset. This improvement was also extended to classes such as Fuzzers and Reconnaissance. Except for the Generic and Normal classes, which experienced a slight drop in classification, the model's performance on the minority-augmented dataset displayed substantial improvement, which is particularly crucial in intrusion detection scenarios. Curiously, the drop in the recall of the two classes could be attributed to the synthetic values used to train the model.

Table 5.6 shows that the augmented dataset has slightly higher specificity than the original dataset. This is especially important considering modern attacks that employ evasive techniques to avoid detection.

5.5 Approach 2 - Feature Perturbation approach

The second approach used for the augmentation and oversampling of the minority class(es) is the perturbation approach, which was explained in section 3.4.1 (Approach 2) with steps in Algorithm 2. Using the smart grid and the BoT-IoT datasets described in Chapter 3, the datasets were sorted into the different subsets were augmented using the SAC technique, where the datasets were first sorted into the instant classes before the generation of synthetic data and subsequent augmentation of the minority class(es). The combined phase of the techniques was then implemented with the original dataset to obtain an augmented new dataset.

5.5.1 Using Smart Grid Dataset

During preprocessing, the dataset underwent augmentation to address the class imbalance. The initial class distribution showed 59,529 instances for the Attack class and 23,814 instances for the Natural class. To introduce more variability into the Natural data, a function called *perturb_features* was created. This function applies Gaussian noise to the features, allowing control over the magnitude of the noise using a parameter which defaults to 0.1. The function returns the perturbed sample. Given the significant difference of 35,715 instances between the Attack and Natural classes, it was decided to generate synthetic data to augment the Natural class. 33,000 new samples were randomly generated and perturbed by introducing Gaussian noise to the features. Next, the newly generated synthetic samples were concatenated with the original Natural data, creating an augmented Natural subset. To create a unified dataset for further analysis or modeling, the augmented Natural class was then combined with the entire dataset. This resulted in a new dataset with a class distribution of 59,529 instances for the Attack class and 56,814 instances for the Natural class. By augmenting the Natural class and combining it with the original dataset, a more balanced distribution was achieved, providing improved data for subsequent modeling and analysis.

Table 5.11: Comparison of the overall accuracy, sensitivity and specificity of the confusion matrix of original data and the Perturbation Augmented data

Dataset	Overall Accuracy	Sensitivity	Specificity
Original data	91	98	71
ROSE augmented	97	96	98
SMOTE augmented	94	96	90
Perturbation augmented	**95**	**96**	**92**

Table 5.11 compares the results obtained from the random forest model. The comparison includes the original dataset and three augmented datasets obtained using oversampling techniques such as ROSE, SMOTE and the feature Perturbation technique. The comparison reveals that the model trained on the perturbation-augmented dataset achieved better overall accuracy and specificity classification when compared with the results obtained from the model trained on the original and SMOTE-augmented datasets. However, the ROSE augmented dataset outperformed the three other oversampling techniques. Perhaps it is interesting to state that while SMOTE and the perturbation techniques generated and used synthetic data, ROSE does not generate synthetic data but only resamples the already existing data. The ROSE technique approach duplicates the values, which, in effect, is prone to overfitting Zhu et al. (2017).

5.5.2 Using BoT-IoT dataset

The data shown in Table 5.12 compares the performance of the random forest model on the BoT-IoT dataset. The dataset exhibits a high level of imbalance, with 585,710 instances classified as Anomaly and 40,073 as the benign class (Normal), resulting in a ratio of 1:14 (malicious: benign). The comparison includes the benchmark, ROSE, SMOTE, and a newly proposed perturbation-augmented model. Remarkably, the perturbation-augmented model achieved the most outstanding performance among the three oversampling techniques, delivering a near-perfect result.

Table 5.12: Comparison of the overall accuracy, sensitivity and specificity of original, ROSE, SMOTE, SAC and SAC+ROSE Augmented data

Dataset	Overall Accuracy	Sensitivity	Specificity
Original data	99.89	98.54	99.98
ROSE Augmented	99.89	98.70	99.97
SMOTE Augmented	99.90	98.92	99.97
Perturbation augmtd	**99.90**	**99.90**	**99.90**

5.6 CONCLUSION

Intrusion detection in the IoT poses challenges, particularly regarding class imbalances that can result in biased classification and false alarms. This chapter addresses these challenges by proposing two data augmentation strategies based on the SAC algorithm. The SAC strategy involves sorting, augmenting, and combining the instant classes of a dataset. The first approach uses a library to generate synthetic data, which is then used to oversample the minority class of the target feature. The second approach involves perturbing the features to generate synthetic data values, which are then used to oversample the minority classes in the dataset. In both approaches, the dataset is initially divided into subsets based on instant classes. Synthetic data are then created from these independent subsets. The synthetic data are combined with each independent class label, resulting in a new training dataset. Notably, features are ranked on the basis of importance using a single feature selection technique. In addition, PCA is employed to further reduce the dimensionality of the data. Subsequently, a random forest model is fitted to binary and multiclass datasets. The results demonstrate improvements in classification compared with the original data and the synthetic data generating approach like SMOTE for both datasets. There is also an improvement over the ROSE technique in the second dataset. The two results indicate that the proposed techniques offer promising solutions

for addressing class imbalances and improving intrusion detection in the IoT.

However, in the following chapter, additional feature selection techniques are adopted to use an ensemble approach. This approach aims to identify the most optimal features because relying solely on a single feature selection technique may result in suboptimal feature sets that may not work effectively with certain learning algorithms. This comprehensive feature selection process is particularly significant and critical in the context of intrusion detection. Furthermore, the proposed approach has the potential to be replicated in other benchmark datasets in the future, thereby facilitating further validation and enhancing its applicability.

Chapter 6

Ensemble Common Features Technique for Lightweight Intrusion Detection in Industrial Control System

6.1 Overview

Chapter 5 of this thesis focuses on achieving effective classification in the IoT using data augmentation and a single feature selection technique. However, relying solely on a single feature selection strategy, as employed in the previous chapter, may not consistently result in the selection of the best features. Different feature selection approaches often rank features differently, which can lead to suboptimal feature subsets, as highlighted by Zhang et al. (2021). Consequently, the optimal features required for effective classification were not always attained, compromising the learning algorithm's performance. This chapter introduces an ensemble feature selection approach and learning algorithms to address this limitation. It recognizes that relying solely on a single feature selection technique may not yield the most effective classification outcomes. By incorporating an ensemble of feature selection techniques and adopting various learning algorithms, the aim is to enhance the overall performance of the classification process, run time and a lower model size in the IoT. Therefore, this chapter aims to answer research question two (RQ2): *In what ways can dimension reduction techniques be employed to obtain optimal*

features for constructing a lightweight intrusion detection model for resource-constrained IoT devices?. To this end, by exploring a combination of feature selection techniques in an ensemble approach, this study aims to identify the optimal features that effectively reduce dimensionality and enhance classification accuracy while minimizing computational costs.

6.2 Introduction

Feature selection is a reliable approach for improving classification and overall accuracy. In intrusion detection, a lightweight technique based on feature reduction is used to select relevant features and discard redundant ones. The approach used in the study is the filter feature selection method. The justification for using this approach is because it is independent of the classifier and it is also resistant to overfitting. , uses statistical techniques to assess the relationship between independent and dependent variables and select features based on statistical scores. This approach is crucial due to the large volume of data and the time-consuming nature of training and classification algorithms Rachburee and Punlumjeak (2015); Rodríguez et al. (2007); Wang et al. (2019). Feature selection is crucial due to the increasing data volume of data being captured during attacks. A larger data would most likely result in longer training time which could then impact a model performance and accuracy. For this study, an ensemble approach that combines three feature selection approaches was used in order to achieve and efficient and computational lightweight model for the IoT.

By adopting a multiple-feature selection approach, computational complexity can be reduced, and classification performance can be improved. Relying on a single feature selection technique may lead to a high false positive rate and longer processing time. References indicate the advantages of combining multiple feature selection techniques, including the selection of non-redundant features and improved performance Ben Brahim and Limam (2018); Mohammadi et al. (2019).

6.2.1 Contribution

- The study achieved a lightweight intrusion detection model based on Common Features Techniques from multiple feature selection techniques.
- The approach can achieve the same high overall accuracy, sensitivity and specificity with reduced features as would with more features.
- The approach is capable of achieving high classification at a significantly reduced

Type of Attacks	Abbreviation
Normal	Normal(0)
Naïve Malicious Response Injection	NMRI(1)
Complex Malicious Response Injection	CMRI(2)
Malicious State Command Injection	MSCI(3)
Malicious Parameter Command Injection	MPCI(4)
Malicious Function Code Injection	MFCI(5)
Denial of Service	DOS(6)
Reconnaissance	Recon(7)

Figure 6.1: target class types

CPU computational cost - in terms of memory usage and computation time.

6.3 Methodology and Dataset

6.3.1 Dataset

The datasets used in this study are laboratory-simulated cyber-attacks on Industrial Control System Network Traffic on gas pipelines and water storage tanks Morris and Gao (2014b); Morris et al. (2011b). The other dataset used are: BoT-IoT dataset Koroniotis et al. (2019), NF-BoT-IoT dataset Sarhan et al. (2021), CIC-IDS2017 dataset Sharafaldin et al. (2018), UNSW18 dataset Moustafa (2019). While the first two datasets were processed and modeled in their multiclass form, the subsequent datasets were condensed into binary classes. The essence of using multiple datasets at this stage is that incorporating multiple datasets serves the purpose of validating the ensemble common features technique's consistency across various datasets. Ensuring that the model produces consistent results across multiple datasets is crucial for establishing it as a reliable method for dimensionality reduction in lightweight intrusion detection systems.

Gas pipeline dataset

This dataset consists of 8 instant classes, 7 different kinds of malicious classes, and 1 benign traffic. The attack types and normal traffic are provided in Figure 6.1, and the distribution of the classes is displayed in Table 6.1.

Table 6.1: Instant classes distribution for Gas pipeline dataset

Classes	Normal	NMRI	CMRI	MSCI	MPCI	MFCI	DOS	Recon
Rows	6671	335	1664	93	842	41	189	783

Water Tank dataset

This is another dataset originating from laboratory simulation, and it contains seven different types of malicious as well as benign traffic. The distribution of the classes are displayed in Table 6.2.

Table 6.2: Instant classes distribution for water tank dataset

Classes	**CMRI**	**DoS**	**MFCI**	**MPCI**	**MSCI**	**NMRI**	**Normal**	**Recon**
Rows	1457	135	155	410	209	1198	19503	4132

Tables 6.1 and 6.2 reveal a heavily skewed distribution towards the Normal class, resulting in a highly imbalanced dataset. To address this issue, the SAC approach was employed to augment the minority classes (see Algorithm 2). The minority classes were oversampled almost up to the threshold set by the number of observations in the benign class. The original dataset had 27,199 observations (rows) and 23 features for the IoT-enabled water tank dataset. However, after augmenting the minority classes, the total number of observations increased to 155,675. Similarly, for the gas pipeline dataset, the original dataset had 10,618 observations and 27 features, with the Normal class being the majority. After oversampling the minority classes based on the benign class as the baseline, the total number of observations in the dataset increased to 53,359. Subsequently, a min-max normalization approach was applied to scale the values, ensuring that all features contribute equally and eliminating bias.

6.3.2 Methodology

After normalizing the data, both datasets were subjected to feature selection techniques to rank and select non-redundant subsets of features. In Section 3.3 and Algorithm 1, three feature selection techniques were employed: Information Gain, Chi-Squared, and Gini-Index. Each dataset's features were prioritized based on their importance, resulting

in three feature subsets. To eliminate redundant and less significant features, the cumulative variance of each technique's values was computed, considering a threshold where further additions would not yield additional increments. This process generated three datasets from the Gas pipeline and water tank datasets, named after the technique used: (a) the Information Gain dataset, (b) the Chi-Squared dataset, and (c) the Gini-Index dataset. In addition, a fourth dataset called the Common Features Technique (CFT) dataset was created by selecting features shared by the three other datasets. For the Gas pipeline dataset, 15 features were retained in each of the Information Gain, Chi-Squared, and Gini-Index datasets after ranking and eliminating redundant features. In the Common Features Technique (CFT) dataset, 11 common features were selected (see Appendix 3 for feature ranking). Similarly, for the water tank dataset, after removing redundant features, each of the Information Gain, Chi-Squared, and Gini-Index datasets contained 17 features. The CFT dataset also comprised 14 common features. These common features were particularly interesting because they exhibited robustness and played a prominent role across the selection techniques. They captured distinct attributes that consistently contributed to various subspaces, thereby promoting effective model generalization and optimal performance. Furthermore, the model's output on the common feature subset was compared with its output on the other datasets. Figure 6.2 provides an illustration of the feature selection strategies. As mentioned in Section 6.1, after generating the subsets, four datasets were obtained from each original dataset. Subsequently, three learning algorithms were applied to each dataset, and the resulting output was recorded. The learning algorithms used were Random Forest (RF), Support Vector Machine (SVM), and k-Nearest Neighbors (KNN). Employing various models aimed to ensure consistency and test the technique's effectiveness.

Before fitting the models, the datasets were divided into a training and validation set in an 80:20 ratio. Using a script, 20 iterations were conducted, each time randomly selecting 70% of the training dataset with replacement. During each iteration, various metrics, such as overall accuracy, sensitivity, specificity, computation time, and memory consumption, were recorded. The models were evaluated on datasets that included Information Gain, Chi-Squared, Gini-Index, and the Common Features Technique (CFT) feature selection techniques. In total, 80 iterations (20 iterations on each dataset) were performed. The average performance metrics for each model on each dataset were calculated and summarized in a tabular format for comparison. In addition, the values underwent a hypothesis test using the two-sample t-Test.

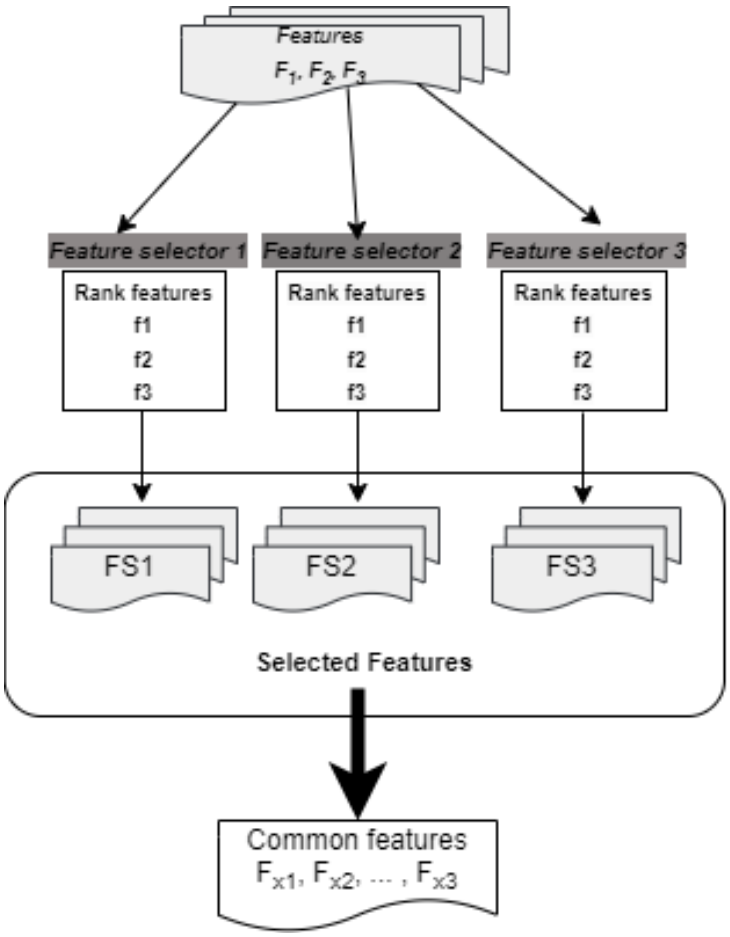


Figure 6.2: Multiple feature selectors

$$(x - min(x))/(max(x) - min(x)) \tag{6.1}$$

6.4 Model Fitting and Discussion

Tables 6.3 to 6.8 present the average metrics values for Random Forest (RF), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) models on the four datasets derived from the original gas pipeline and water tank datasets.

For the gas pipeline dataset, Tables 6.3, 6.4, and 6.5 summarize the average accuracy, sensitivity and specificity of the models using Random Forest, Support Vector Machine, and K-Nearest Neighbors, respectively. For the water tank datasets, Tables 6.6, 6.7, and 6.8 show the average accuracy, sensitivity and specificity values of the models using Random Forest, Support Vector Machine, and K-Nearest Neighbors, respectively. All the tables compare the output of the models on the Common Feature Technique (CFT), Information Gain (Info_Gain), Chi-Squared (Chi_Sqd), and Gini Index (Gini_Index) datasets. The average values are displayed in the tables below. .

Table 6.3: Summary of Average sensitivity and specificity values with SVM model on gas pipeline datasets

	Accuracy	Sensitivity								Specificity								Time (sec)	Memory
		CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon	CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon		
CFT (%)	87.10	99.50	60.70	95.50	98.55	92.08	96.60	57.34	100	99.70	100	100	99.95	100	88.68	94.47	100	5.01	27.57
Info_Gain (%)	88.70	99	68.12	95.30	99.03	93.39	96.49	56.64	100	99.79	100	100	99.93	99.99	92.47	94.91	100	6.04	30.26
Chi_Sqd (%)	89.83	99.30	77.82	95.40	99.11	93.33	97.38	57.15	100	99.73	100	100	99.93	99.99	93.42	95.35	100	5.97	29.85
Gini_Index (%)	88.72	99.10	71.31	95.4	98.69	92.15	96.17	55.98	100	99.75	100	100	99.94	99.99	92.45	94.96	100	6.11	29.93

From the result displayed in Table 6.3, it could be observed that the overall accuracy of the SVM model on the CFT dataset is 87.10%, which is slightly lower than the results achieved by the model on the other datasets. The lower classification, as displayed in the table, could be traced to the sensitivity classification of the MPCI class and the specificity value for the NMRI class. The lower classification by the SVM model could be attributed to the fact that its classification is influenced by the number of data points in the dataset. In general, SVM tends to perform well with a moderate to large amount of data, as it relies on finding the optimal hyperplane that best separates different classes. With a small dataset that the CFT offers, SVM would struggle to find a clear decision boundary, leading to overfitting or poor generalization of the SVM model.

From the summary of the results displayed in Table 6.4, the RF model on the CFT dataset

Table 6.4: Summary of average sensitivity and specificity values using Random Forest model on gas pipeline datasets

	Accuracy	Sensitivity								Specificity								Time (sec)	Memory
		CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon	CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon		
CFT (%)	93.09	99.6	81.53	86.8	98.99	98.14	93.32	97.53	100	99.67	100	100	99.88	90.07	90.27	99.02	100	0.12	15.57
Info_Gain (%)	95.76	99.9	84.39	96.2	98.75	93.67	94.73	98.31	100	99.79	98	100	99.93	99.99	97.85	99.4	100	0.15	18.32
Chi_Sqd (%)	96.76	99.6	92.63	96.2	98.56	93.8	95.37	98.27	100	99.79	98	100	99.93	99.93	98.91	99.34	100	0.15	18.11
Gini_Index (%)	95.82	99.7	86.39	95.5	99.19	92.69	94.91	97.77	100	99.7	98	100	99.96	100	98.13	99.36	100	0.16	18.35

achieved an overall accuracy of 93.09%. Again, this is slightly lower than the model's result on the other datasets, though it achieved a lower computation time and model size. This is, however, very interesting, considering that the RF model works effectively both with smaller and larger datasets. Possibly, the reduction in the variations and lower diverse samples in the dataset may have caused the model not to build enough multiple decision trees.

Table 6.5: Summary of average sensitivity and specificity values using KNN model on gas pipeline datasets

	Accuracy	Sensitivity								Specificity								Time (sec)	Memory
		CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon	CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon		
CFT (%)	89.40	98.23	88.67	100	99.51	92.62	63.05	78.57	100	99.90	97	99.30	99.81	99.85	96.98	93.68	100	5.96	20.86
Info_Gain (%)	89.81	99.89	100	100	99.82	99.87	97.07	93.72	100	99.89	100	100	99.82	99.87	97.07	93.72	100	6.54	23.79
Chi_Sqd (%)	88.07	99.89	97	99.1	99.82	99.87	97.06	93.72	100	99.89	97	99.1	99.82	99.87	97.06	93.72	100	7.37	23.71
Gini_Index (%)	87.87	99.89	97	99.1	99.82	99.87	97.06	93.72	100	99.89	97	99.1	99.82	99.87	97.06	93.72	100	7.46	23.73

Table 6.5 shows the result of the fitting of the KNN model all the datasets. It could be observed that the model on the CFT dataset achieved 89.40% overall classification which is almost at par with the model's results obtained on the other datasets. More importantly, the model on the CFT dataset achieved the classification in 5.96s using 20MB of memory.

Table 6.6: Summary of average sensitivity and specificity values using Random Forest model on water tank datasets

	Accuracy	Sensitivity								Specificity								Time (sec)	Memory
		CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon	CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon		
CFT (%)	98.18	89.95	94.58	100	99.94	96.85	98.13	99.97	100	99.29	90.56	100	100	99.99	99.98	98.76	100	0.38	44.30
Info_Gain (%)	99.09	92.48	99.23	100	99.97	96.93	98.65	100	100	99.45	99.87	100	100	100	99.99	98.90	100	0.40	48.93
Chi_Sqd (%)	98.90	90.83	99.34	100	99.95	97.10	99.05	100	100	99.69	99.60	100	100	100	99.98	98.83	100	0.45	49.10
Gini_Index (%)	98.97	90.83	99.33	100	99.99	96.96	98.52	99.97	100	99.69	99.50	100	100	100	99.99	98.75	100	0.45	48.87

Similarly, for the water tank dataset, the RF model fitted on the four datasets, as shown in Table 6.6, indicates that the model result on the CFT dataset achieved 98.18% overall

accuracy, which is almost at par with the model's result on the Chi-squared and the Gini-index dataset. In the same vain, except for the TP classifications on the CMRI and DoS classes, and lower specificity value for the DoS class, the model on the CFT data can be said to be effective as the overall accuracy was achieved in 0.38s using 44.30MB of memory.

Table 6.7: Summary of average sensitivity and specificity values using Support Vector Machine model on water tank datasets

	Accuracy	Sensitivity								Specificity								Time (sec)	Memory
		CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon	CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon		
CFT (%)	94.52	92.95	96.58	95.20	96.94	96.85	93.13	81	100	91.29	90.56	100	96.58	89.56	92.98	98.76	100	4.37	42.10
Info_Gain (%)	94.54	92.48	96.23	94.30	97.87	96.93	94.65	80	100	90.45	91.87	100	96.70	90.40	91.99	98.90	100	5.45	48.73
Chi_Sqd (%)	94.08	90.83	95.34	92.67	95.95	97.10	94.05	80.5	100	90.69	91.60	100	95.72	90.10	91.98	98.83	100	5.47	48.20
Gini_Index (%)	94.06	90.83	95.33	93.56	96.91	97.36	93.52	79.97	100	90.69	91.30	100	95.80	90	90.99	98.75	100	5.46	47.77

In this table (Table 6.7), the model achieved a commensurate classification in terms of overall accuracy with a 94.52% for the CFT dataset. This feat was achieved in 4.37s using 42.10MB of memory. Interestingly, the model utilising the CFT data also performed well for the TP with values that are at with the model's result on the other datasets.

Table 6.8: Summary of average sensitivity and specificity values using k-Nearest Neighbour model on water tank datasets

	Accuracy	Sensitivity								Specificity								Time (sec)	Memory
		CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon	CMRI	DoS	MFCI	MPCI	MSCI	NMRI	Normal	Recon		
CFT (%)	99.00	91.95	99.38	100	99.94	96.85	98.13	99.97	100	99.49	99.56	100	100	99.99	99.98	98.78	100	3.39	46.42
Info_Gain (%)	99.08	92.48	99.23	100	99.97	96.93	98.65	100	100	99.55	99.87	100	100	100	99.99	98.60	100	4.41	49.83
Chi_Sqd (%)	99.02	90.83	99.34	100	99.95	97.10	99.05	100	100	99.69	99.60	100	100	100	99.98	98.83	100	5.43	50.10
Gini_Index (%)	98.98	90.83	99.33	100	99.99	96.96	98.52	99.97	100	99.71	99.60	100	100	100	99.99	98.75	100	5.44	48.82

Table 6.8 provides another overview of the model's result on the water tank dataset. The KNN was the model used, and the output clearly shows that the model's result on the CFT dataset was 99.00%, which was near perfect overall accuracy. This classification was achieved in 3.39s using 46.42MB of memory.

Overall, while some classical machine learning models have exhibited improved performance using the CFT model compared to the other datasets, other models have performed poorly due to the size of the data. However, the computation time and memory utilized make the CFT approach very interesting as it makes further work on improving the classification through neural networks and optimization more desirable.

6.4.1 BoT-IoT dataset

There were 625,783 observations and 70 features in this dataset. However, the target class was condensed into a binary class such that only classes that correlated with Mirai and Normal were selected. As a result, the total number of observations was reduced to 455,750. After preprocessing and normalization, feature selection techniques were applied, resulting in the CFT dataset having 11 features and the other approaches (Information gain, Chi-squared and Gini-index) having 24 features each (See Appendix 4 for ranked features). Tables 6.9, 6.10, and 6.11 summarize the average results from the SVM, RF, and LDA models after 30 iterations on each dataset. The summary table displays the average values for overall accuracy, sensitivity, specificity, computation time, and memory usage.

Table 6.9: Summary of average values of performance metrics on BoT-IoT dataset using SVM model

Metrics of Performance	**Summary of average values**			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	**98.18**	98.15	98.20	98.00
Sensitivity (%)	**99.9**	99.87	99.88	99.89
Specificity (%)	80.35	80.11	80.47	78.75
Comp. Time (s)	**11.3**	18.17	21.27	18.87
Memory (MB)	**80**	147	146	147

Table 6.10: Summary of average values of performance metrics on BoT-IoT dataset using Random Forest model

Metrics of Performance	**Summary of average values**			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	**99.84**	98.81	99.770	99.68
Sensitivity (%)	**99.99**	99.99	99.99	99.99
Specificity (%)	**98.22**	97.9	97.44	96.37
Comp. Time (s)	**1.2**	1.6	1.8	1.6
Memory (MB)	**75**	110	116	116

From the results presented in Table 6.9, the average overall accuracy achieved through multiple iterations with the CFT approach stands at an impressive 98.18%. This figure is on par with the accuracy obtained using the information gain technique (98.15%), the Chi-squared technique (98.26%), and the Gini-index technique (98.6%). In terms of sensitivity, all strategies effectively identified over 99% of true positives, indicating a robust detection of malicious traffic. When assessing the benign class (normal traffic), which

Table 6.11: Summary of average values of performance metrics on BoT-IoT dataset using LDA model

Metrics of Performance	**Summary of average values**			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	96.68	96.45	**96.81**	96.35
Sensitivity (%)	99.17	98.82	99.44	99.31
Specificity (%)	70.84	71.43	69.08	66.10
Comp. Time (s)	**0.11**	0.14	0.13	0.12
Memory (MB)	**31**	48	46	45

constitutes the true negatives, the CFT approach demonstrated an average specificity classification of 80.35%, as depicted in Table 6.15. This performance aligns closely with the average specificity values observed using other approaches. Examining computation time, the average time displayed in the table illustrates that the CFT approach requires only 11.3 seconds on average. This is a substantial reduction compared with the averages for the other techniques, which are 18.17, 21.27, and 18.87 seconds. It's noteworthy that SVM exhibits significantly higher run-time complexity, potentially contributing to its slightly longer validation time across the techniques. In terms of space complexity (memory usage), the CFT approach used 80MB of RAM for validation, which is a notably lower requirement compared to other techniques.

Furthermore, the output of the Random Forest model on the same dataset is presented in Table 6.10. Notably, the CFT approach achieved a remarkable 99.8% classification rate in just 1.2 seconds, using 75MB of memory. In contrast, the alternative methods required more memory and a longer average time. Similarly, fitting an LDA model on the dataset with constant size and dimensions yielded the results displayed in Table 6.11. The CFT approach used 31MB of memory and achieved an average overall classification rate of 96% in just 0.11 seconds. The sensitivity and specificity classifications were 99% and 70%, respectively. These findings underscore the efficiency and effectiveness of the CFT approach in achieving high classification accuracy with reduced computational resources.

6.4.2 CIC-IDS2017 dataset

This dataset contains 78 features and 225,745 observations. After data preprocessing and feature ranking in order to remove redundant features, the CFT approach dataset features came down to 13, whereas the other approaches each had 22 features in the datasets (See Appendix 2 for feature ranking). Tables 6.12, 6.13, and 6.14 show the

summary of the average metrics values after several iterations for the RF, SVM, and LDA models upon fitting the models.

Table 6.12: Summary of average values of performance metrics on CIC-IDS2017 dataset using RF modell

Metrics of Performance	**Summary of average values**			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	99.96	99.98	99.98	99.97
Sensitivity (%)	99.95	99.97	99.97	99.96
Specificity (%)	99.99	99.99	99.99	99.99
Comp. Time (s)	1.25	2.13	1.97	1.41
Memory (MB)	87	129	132	131

Table 6.13: Summary of average values of performance metrics on CIC-IDS2017 dataset using SVM

Metrics of Performance	**Summary of average values**			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	99.37	99.45	99.51	99.74
Sensitivity (%)	99.84	99.81	99.79	99.87
Specificity (%)	98.78	98.96	99.14	99.56
Comp. Time (s)	1.5	1.6	1.5	1.7
Memory (MB)	84	130	133	133

Table 6.14: Summary of average values of performance metrics on CIC-IDS2017 dataset using LDA

Metrics of Performance	**Summary of average values**			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	96.9	97.11	96.86	94.77
Sensitivity (%)	99.89	99.66	99.27	99.92
Specificity (%)	92.98	93.79	93.75	88.06
Comp. Time (s)	1.20	1.18	1.3	1.91
Memory (MB)	30	42	43	42

The output of the RF, SVM, and LDA models on the sub-datasets obtained from the CIC-IDS2017 dataset is shown in Tables 6.12, 6.13, and 6.14. Using the RF, SVM, and LDA models on the CFT approach, the models achieved an overall accuracy of 99, 99, and 96% in 1.25, 1.5, and 1.2 seconds, respectively, while using 87, 84, and 30MB of memory. In all three models, the CFT approach achieved an impressive classification of 99% of the true positive. The other approaches also achieved impressive classification;

however, these classifications came at a higher computational cost in each of the cases.

6.4.3 UNSW18 dataset using Deep Learning

This dataset comprises 999,999 observations and 44 features. Following feature selection, the CFT approach dataset was refined to 11 features, while other approaches retained 16 features each. Notably, the benign class was represented by only 32 observations, a stark contrast to the 999,967 observations in the malicious class. This resulted in a highly imbalanced dataset with a ratio of 31,248:1 (malicious to benign). Recognizing the imbalance, oversampling techniques, as detailed in Chapter 5, were applied to boost the minority class. This oversampling increased the dataset size to 1,999,934. After dividing the data into training and testing sets through a script, a random sample equivalent to 70% of the observations was selected for each of the 30 iterations across the sub-datasets. Given the use of deep learning, a model with 10 epochs was employed. The architecture featured two dense layers with eight input units, utilizing ReLU (Rectified Linear Unit) as the activation function. Additionally, a batch size of 32 was implemented to ensure swift training iterations and quick convergence. Table 6.15 presents a summary of the average metric values.

Table 6.15: Summary of average values of performance metrics using the deep learning model on UNSW18

Metrics of Performance	Summary of average values			
	CFT	***Info. Gain***	***Chi-Square***	***Gini-Index***
Accuracy (%)	100	100	100	100
Sensitivity (%)	100	100	100	100
Specificity (%)	100	100	100	100
Comp. Time (s)	0.39	0.42	0.39	0.41
Memory (MB)	104	104	106	105

The outcome of the Shallow Learning applied to the four sub-datasets reveals a flawless 100% classification accuracy across all datasets. The table displays values for overall accuracy, sensitivity, and specificity, indicating consistent model performance across different approaches. Notably, the computation time for these approaches demonstrates a substantial reduction, ranging from 0.39 to 0.41 seconds, compared to traditional machine learning models. Interestingly, while the memory utilization size was higher, ranging from 104MB for CFT and Information Gain to 106MB for the Chi-Squared dataset, these values underscore the efficiency of the Shallow Learning approach in achieving optimal classification accuracy.

6.4.4 Measure of Significance - Hypothesis testing

The hypothesis under consideration is the acceptance of the CFT model's approach for a lightweight intrusion detection model. This stems from its capability to reduce computational costs while maintaining an equivalent classification rate to traditional approaches. A statistical test was employed to assess the validity of this hypothesis and determine whether the null hypothesis should be rejected or accepted. Given the multiple iterations performed (20 for multiclass and 30 for binary class datasets), resulting in a sample size of $n = 20\,\&\,30$, the t-test emerged as the most suitable, as it is applicable for sample sizes not exceeding 30. This inferential statistical test evaluates whether a statistically significant difference exists between the means of two variables.

To conduct this hypothesis test, a two-tailed t-test was executed (refer to Equation 6.2) to ascertain if a significant difference exists between the means of computation time and memory for the four data approaches, with the CFT computation time and memory serving as the reference. Tables 6.16 to 6.19 showcase the p-values obtained from the average metric values in Tables 6.3 through 6.6. These tables correspond to the average values achieved using information gain, chi-squared, Gini index, and CFT as variables. The computed p-values were derived to compare the means of computation time and memory usage. The comparisons were conducted as follows: Common Features Technique vs Information Gain (CFT vs Inf-Gain), Common Features Technique vs Chi-Squared (CFT vs Chi-Squared), and Common Features Technique vs Gini-Index (CFT vs Gini-Index). The calculated p-values in these tables signify the significance of the difference in average memory usage and computation time between the CFT and other approaches. These comparisons aim to assess how the computation time and memory utilization of the CFT model compare with those of the other models.

The Two-Sample t-Test
$H_0 : \mu_1 - \mu_2 = \Delta_0$
$H_0 : \mu_1 = \mu_2$
$H_1 : \mu_1 \neq \mu_2$

$$Two\ test\ value : t = \frac{\bar{x} - \bar{y} - \Delta_0}{\sqrt{\frac{S_1^2}{m} + \frac{S_2^2}{n}}} \tag{6.2}$$

where

$\bar{x}$ & $\bar{y}$ are sample mean
μ_1 is population mean 1
μ_2 is population mean 2
m & n are number of samples
S_1 & S_2 are standard deviations

The p-value is a statistical measure that indicates the probability of accepting or rejecting an observed outcome within a 90% or 95% confidence interval (CI). A lower p-value suggests stronger evidence in the data sample to reject the null hypothesis in favor of the alternative hypothesis. A p-value of 0.05 or less is considered statistically significant, indicating that the observed outcome is unlikely to occur by chance.

Hypothesis testing - multiclass datasets

Table 6.16: P-values for time and memory for Tables 6.3

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini_In**
Time	4.41E-09	5.78E-09	7.52E-09
Memory	2.20E-16	2.20E-16	2.20E-16

Table 6.17: P-values for runtime and memory for Tables 6.4

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini**
Time	2.23E-04	2.76E-04	3.03E-06
Memory	2.20E-16	2.20E-16	2.20E-16

Table 6.18: P-values for runtime and memory for Tables 6.5

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini**
Time	5.33E-11	5.17E-14	2.20E-16
Memory	6.81E-02	2.20E-16	2.20E-16

Table 6.19: P-values for runtime and memory for Tables 6.6

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini**
Time	8.12E-03	2.55E-08	4.60E-09
Memory	2.20E-16	2.20E-16	2.20E-16

Hypothesis testing - binary class datasets

The following tables provide the p-values corresponding to the average computation time and memory usage presented in Tables 6.9 to 6.14. These p-values indicate the likelihood that the observed outcomes can be accepted or rejected based on a confidence interval of 90% or 95% acceptability using a statistical measure. It is worth noting that a p-value of 0.05 or lower is generally considered statistically significant. In this study, we are particularly interested in comparing the computational cost of the models using the CFT approach with the other approaches. To achieve this, we calculate the p-values using a two-tailed t-test, specifically for the average time of the CFT approach versus Information Gain, the CFT approach versus Chi-squared, and the CFT approach versus Gini-index. The computed p-values can be found in Tables 6.20 to 6.22. Due to space limitations, only some of the model's outputs are displayed in these tables.

Table 6.20: P-values for runtime and memory for Table 6.12

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini**
Time	2.2e-16	1.319e-11	1.607e-08
Memory	2.2e-16	2.2e-16	2.2e-16

Table 6.21: P-values for runtime and memory for Table 6.10

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini**
Time	1.76E-02	0.5157	1.04E-09
Memory	2.20E-16	2.20E-16	2.20E-16

6.4.5 Measure of significance discussion

Tables 6.3 and 6.4 showcase the SVM model's consistent overall accuracy, surpassing 88%, when using information gain, chi-squared, and Gini index feature selection approaches. In contrast, the model employing the CFT feature selection technique achieved a slightly

Table 6.22: P-values for runtime and memory for Table 6.15

	P-values		
	CFT-vs-Inf_Gn	**CFT-vs-Chi_Sq**	**CFT-vs-Gini**
Time	5.92E-05	0.4765	0.09655
Memory	0.4593	1.58E-04	0.03372

lower overall accuracy of 87%. With the exception of the Normal class, robust classification is evident across classes, as indicated by the sensitivity (True Positives) values in the tables. The p-values presented in Table 6.9, within a 95% confidence interval, reveal a statistically significant difference in computation time and memory usage between the CFT feature selection technique and other approaches. This suggests that the CFT technique achieves classification performance comparable to traditional feature selection approaches while significantly reducing computational resource consumption. Similarly, the results in Tables 6.4 shows that the random forest model, which uses the CFT feature selection technique, achieves an overall accuracy of 93%. The p-values in Table 6.10 indicates a statistically significant difference in computation time and memory usage compared with other feature selection techniques, which achieved an overall accuracy of 95%. Consequently, it can be concluded that the random forest model trained on the CFT dataset achieves high classification, similar to other feature selection approaches, at a lower computational cost.

Moreover, when fitting the KNN model to datasets from the four feature selection approaches (Tables 6.5), the CFT approach achieves a classification rate exceeding 89%, which is comparable to the other three techniques. Notably, while the statistical significance of the computation time difference between the CFT approach and other approaches (see Table 6.11) was observed at a 95% confidence interval, the p-value for memory (0.0681) indicated a significant difference between the CFT and information gain approaches. This warrants rejection at 95% confidence intervals but acceptance within 90% confidence intervals. Hence, it can be concluded that the model using the CFT feature-generated dataset achieves classification rates similar to those of other techniques with a lower computational cost. Similarly, as evident in Tables 6.6, the random forest model using the CFT technique boasts a 98% classification rate, which is at per with the performance of other approaches that also achieved 98%. The p-values in Table 6.12 indicate that the CFT technique could achieve high intrusion classification at a low computational cost, as both computation time and memory usage p-values were less than 0.05.

Table 6.20 provides the p-values representing the comparison of average computation time and memory usage across 30 iterations for Information gain, Chi-Squared, and Gini index techniques with respect to the Common Feature Technique (CFT). The comparisons between the p-values for computation time of CFT and Information gain, CFT and Chi-Squared, and CFT and Gini index are 2.2e-16, 1.319e-11, and 1.607e-08, respectively. These values are statistically significant ($p < 0.05$) and fall within the 95% confidence interval. Therefore, considering the average classification results in Table 6.12, it can be concluded that the CFT technique performed better with significantly lower computational cost. Similarly, Table 6.21 presents the p-values for computation time and memory comparisons based on average values in Table 6.10. The computed p-values for CFT vs Information gain, CFT vs Chi-Squared, and CFT vs Gini index techniques are 1.76e-02, 0.5157, 1.04e-09, and 2.20e-16, respectively. This indicates that the CFT approach outperformed the Information gain and Gini index techniques with statistical significance within the 95% confidence interval. However, for the comparison between CFT and Chi-Squared techniques, the difference is not statistically significant ($p > 0.05$ or 0.1). Notably, the memory usage values were acceptable within the 95% confidence interval. In both cases (Tables 6.20 and 6.21), the null hypothesis is rejected, suggesting that the model using CFT techniques performed optimally with lower computational cost than the other techniques, as the p-values are less than 0.05, indicating statistical significance within the 95% confidence interval.

Conversely, Table 6.15 represents a deep learning model with perfect classification across the models. The comparison of computation time between CFT and Gini index approaches in this table has a p-value of 0.09655, which lies within the 95% confidence interval and is considered statistically insignificant. However, at the 90% confidence interval, the difference becomes statistically significant. Additionally, the p-value for memory usage between the two procedures is 0.03372, indicating statistical significance. Therefore, it can be inferred that the CFT strategy performs comparably to other approaches for the dataset used in the experiment, as demonstrated in Table 6.15, with the added advantage of lower computational cost. Regrettably, it is worth noting that the memory usage based on the average output in Table 6.15 is relatively high despite it being statistically significant within the acceptable range at the 95% confidence interval.

6.5 Conclusion

In this study, a lightweight intrusion detection technique based on the Common Features Technique (CFT) was introduced. This approach involves prioritizing features using various selection techniques and selecting a common subset based on a defined threshold for cumulative variance. The result is a subset of common features across the sub-datasets. This study employed both traditional machine learning algorithms and deep learning models, including Random Forest, Support Vector Machine, K-Nearest Neighbour, and Linear Discriminant Analysis models. The models' outcomes on the four sub-datasets were assessed using a two-tailed t-test for statistical significance. Remarkably, models using the CFT technique achieved high classification accuracy with a reduced feature set compared with techniques such as information gain, chi-squared, and Gini-index. Crucially, the CFT technique achieved this classification with a reduced computational cost.

Noteworthy is the effective classification demonstrated by traditional machine learning models trained on the CFT dataset, which exhibited comparable performance to models using the complete set of data features. Conversely, the deep learning model achieved perfect classification in terms of overall accuracy, sensitivity, and specificity. Furthermore, this perfect classification was accomplished with a shorter computation time. However, memory usage remained relatively high, necessitating further reduction to accommodate resource-constrained devices such as the IoT. As a result, the succeeding chapter of this thesis delves into additional compression techniques for the CFT dataset. It also explores further advancements in classification through deep learning methodologies and optimization processes involving pruning, deparameterization, quantization, and inferencing. The overarching goal is to further diminish the computational cost, particularly in terms of computation time and memory usage.

Chapter 7

Optimized Common Features Selection & Deep-Autoencoder (OCFSDA) For Lightweight Intrusion Detection in Internet of Things

7.1 Overview

Chapter 6 of this thesis initiated a dimensionality reduction approach to develop a lightweight intrusion detection model. Employing the common features technique (CFT), this method focuses on selecting non-redundant features to enhance effective classification. Notably, fitting various learning algorithms on the CFT dataset yielded commendable classifications while reducing computational costs, both in terms of computation time and memory utilization. However, the deep learning model applied to the CFT dataset achieved perfect classification within a shorter computation time, albeit with relatively higher memory usage. Building on the perfect classification obtained by the deep learning model in the preceding chapter, this chapter extends the exploration. Here, further dimensionality reduction of the CFT data was delved into through feature extraction. This process involves optimization, pruning, deparameterization, and inferencing, all of which aim to achieve a computationally efficient and lightweight intrusion detection

model tailored for the Internet of Things (IoT).

7.2 Introduction

The focus of this chapter is to answer the research questions and more specifically research questions (RQ2 & RQ3) which goes thus: RQ2 - *In what ways can dimension reduction techniques be employed to obtain optimal features for constructing a lightweight intrusion detection model for resource-constrained IoT devices?* and RQ3 - *n the realm of IoT security, how can the strategies outlined in Question 2 (RQ2) be optimized to bolster resilience, efficiency, and overall performance of the intrusion detection model?*

As clearly shown in Chapter 2 of this thesis, the IoT has transitioned from being a paradigm that revolutionizes the acquisition, processing, and use of data for the enhancement of activities and value addition to a source of concern, especially for attacks due to their vulnerabilities. In fact, for many, it meant safety and efficiency in the acquisition and delivery of data, assisted decision-making, and ease of operation. This chapter focuses on achieving a lightweight intrusion detection model that is computationally efficient and inexpensive for detecting attacks. A number of the approaches highlighted in Chapter 3 of this thesis were used, and two benchmark datasets were used. Furthermore, the Shallow Deep Learning approach was used in view of the computational and resource requirements that might be needed if the traditional deep learning model were to be used, especially when the resource-constrained nature of the Internet of Things (IoT) is considered.

7.2.1 Contribution

We present a novel Optimized Common Features Selection and Deep-Autoencoder (OCFSDA) model for lightweight intrusion detection in the IoT.

1. The proposed model was able to effectively reduce the CPU computational cost, such as memory usage and processing time. This reduction was achieved through effective optimization processes and improvement in the algorithm when it was applied to two benchmark datasets.

2. The proposed model is resilient against adversarial attacks such as label poisoning attacks. The resilience of the proposed model was achieved through robust learning and augmentation to enhance effective learning and detection.

3. The proposed model, in addition to reducing memory footprint and execution time, also shows effectiveness across different metrics of measurement. These were compared with other related works on lightweight intrusion detection systems and the proposed model showed immersed improvement.

7.3 Methodology

The approaches used in achieving the objectives of this chapter are as provided in Chapter 3 of this thesis. The methods and their applications are as follows:

7.3.1 Dataset and preprocessing

The datasets used in this Chapter are two openly accessible IoT network datasets: the MQTT-IoT-IDS2020 Hindy et al. (2020) and the CICIDS2017 Sharafaldin et al. (2018) datasets. The utilization of these datasets is crucial as it aligns with the intended implementation of the model on an IoT platform. Given the IoT context, the use of IoT-based datasets becomes essential. Furthermore, the datasets have been converted into binary class datasets to facilitate the modeling process.

The MQTT-IoT-IDS2020 was generated by simulation of the Message Queuing Telemetry Transport (MQTT) protocol. The MQTT protocol is employed in machine-to-machine communication within the Internet of Things (IoT) domain. In the setup of the simulation, various components, such as sensors, a broker, a simulated camera, and an attacker, were used, and five scenarios were captured during the simulation. The scenarios include normal and malicious operations. The malicious activities included aggressive scans, UDP scans, Sparta SSH brute-force attacks, and MQTT brute-force attacks. Pcap files were then generated and stored, and features were extracted from these files. The dataset comprises 225,711 observations and 68 features. As mentioned earlier, the malicious classes were consolidated into a single class, and the class distribution was as follows: 128,025 instances of malicious traffic and 97,681 instances of benign traffic (see Figure 7.1). Similarly, the CIC-IDS2017 dataset, which was used in this study was created by the Canadian Institute of Cybersecurity. The simulation was a stopgap measure to generate datasets that would serve the purpose of addressing the lack of benchmark datasets and offer insights into traffic diversity, volumes, known attacks, and anonymized packet payloads, all of which reflect real network infrastructure trends Sharafaldin et al. (2018). The simulation setup was partitioned into two segments. Part 1 encompassed four machines responsible for executing various attacks, while the second part involved

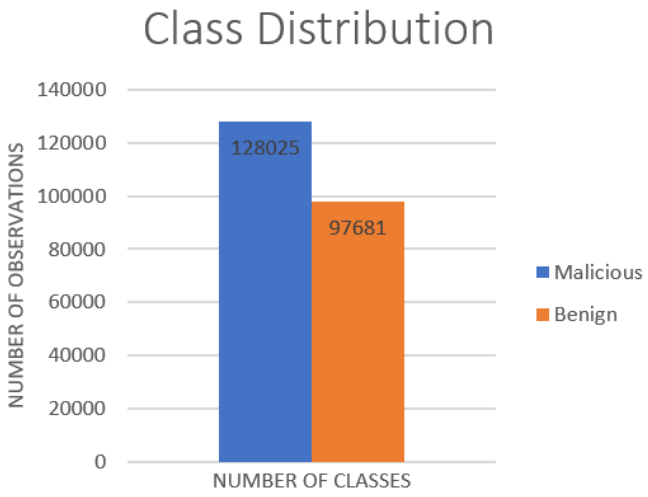


Figure 7.1: Figure showing Class distribution for MQTT-IoT-IDS2020 dataset.

ten machines susceptible to vulnerabilities. In the end, the dataset had 78 features and 286,467 observations, with the instant classes representing the attack class comprising 158,930 observations, while the benign class accounted for 127,537 observations (see Figure 7.2).

During the data preprocessing stage, we found that 10 features in the CIC-IDS2017 dataset either contained only zero values or had extremely marginal values. After normalization, these features resulted in NaN values. Consequently, we excluded these 10 features from the dataset. We then applied the min-max normalization technique to the remaining independent variables to prevent numerical instabilities in some machine learning algorithms. This normalization ensures that the input values fall within the manageable range of 0 to 1, thereby improving the performance of the learning algorithms and facilitating faster convergence.

Evidently, data characterized by high dimensions can degrade and diminish the overall accuracy and training efficiency of learning algorithms Nguyen et al. (2020); Zebari et al. (2020). Therefore, after the normalization of the dataset, the process of selecting informative feature subsets that could enhance the performance of the learning algorithms was applied.

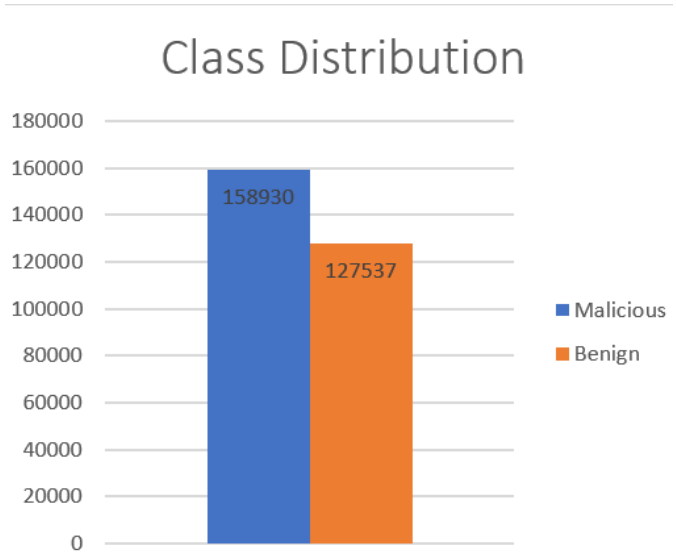


Figure 7.2: Figure showing Class distribution for CIC-IDS2017 dataset.

7.3.2 Feature Selection and Common Features subset

At this stage, relevant features were selected, but before then, the features were ranked using the 3 feature selection techniques explained in Section 3.0.5. The ranking of the features in decreasing order of magnitude is shown in Appendix 1 & 2. Upon the conclusion of ranking and the computation of the cumulative variance, a selection of features up to the threshold was performed and tabulated for each of the 3 feature selection techniques. This resulted in 3 different subsets of feature selection techniques. Following on from the formation of the 3 subsets of features, a common feature subset was generated, as shown in Tables 7.1 and 7.2 for the 2 datasets. As previously stated in section 3.3.4, the Common Features subset pertains to features shared among the 3 subsets of the feature selection techniques. The area indicated as common features is the point of interest such that features found in this subspace are then used as input for the next phase of the study. More importantly, the common features serve as the optimal features that are amenable to more learning algorithms and classification.

In summary, three feature selection techniques were employed to rank the features of the datasets based on their importance. Cumulative variance computation was used to select feature subsets for each technique. Subsequently, a subset of features that were common to the initial subsets was chosen and used as input for the next phase of the study.

Table 7.1: Showing MQTT-IoT-IDS2020 dataset common features

S/N	***MQTT-IoT-IDS2020 Common Features***
1	Total.Length.of.Fwd.Packets
2	Subflow.Fwd.Bytes
3	Average.Packet.Size
4	Bwd.Packet.Length.Mean
5	Fwd.Header.Length.1
6	Fwd.Header.Length
7	Bwd.Packet.Length.Max
8	Init_Win_bytes_forward
9	Fwd.Packet.Length.Mean
10	Avg.Fwd.Segment.Size
11	Fwd.Packet.Length.Max
12	Total.Fwd.Packets
13	Subflow.Fwd.Packets
14	Init_Win_bytes_backward
15	act_data_pkt_fwd
16	Bwd.IAT.Total
17	Fwd.Packet.Length.Std

Table 7.2: Showing CIC-IDS2017 dataset Common Features

S/N	***CIC-IDS2017 Common Features***
1	Packet Length Mean
2	Packet Length Std
3	Average Packet Size
4	Init_Win_bytes_backward
5	Init_Win_bytes_forward
6	Fwd Packet Length Mean
7	Avg Fwd Segment Size
8	Fwd Packet Length Max
9	Avg Bwd Segment Size
10	Bwd Packet Length Max
11	Bwd Packet Length Min
12	Flow Duration
13	Flow IAT Max
14	Max Packet Length
15	PSH Flag Count
16	Fwd Packet Length Min
17	Min Packet Length

7.3.3 Data Compression

The data compression phase of this study is to enable us to extract the features of the data arising from the common features in section 3.2. At this point, fewer features are needed going forward and the features to be extracted must bear a true representation of the original features - in the form of a compressed representation. The Long-Shot-Term-Memory Autoencoder (LSTM-AE), as explained in Section 3.5, was used to compress the data and extract features. Before the compression of the features, three experimental values, 8, 6, and 5, were used as a time-step for the sequence. Similarly, a sliding window of 9, 5, 2 and 1 were also used to obtain a more effective sequence that will make learning the dependencies and prediction as granular as possible. Eventually, 5 was used for the time-step, and 1 was used for the sliding window. The time-step and sliding window are essential parameters as they play crucial roles in determining how the input data (common features data) is processed and represented by the model. While the time-step parameter refers to the number of observations to be considered in each input sequence provided to the LSTM autoencoder, it provides a sequence of data often encountered in IoT sensor readings for which cybersecurity data are essential in this case because of the need to understand the past and present event to predict the future. The sliding window, on the other hand, governs how input sequences are constructed and the extent of the overlap between them, thus allowing the model to capture finer-grained patterns. A smaller sliding window may result in more overlap between sequences, allowing the model to capture finer-grained patterns. Another justification for using the parameters is that it also helps to minimize the limitations highlighted in Section 8.1 of this thesis.

The sequenced data is used as the input of the LSTM-AE model. As the sequence unfolds, the LSTM-AE cells process the data and generate a compressed representation of the input, as shown in Equations 3.18 and 3.19. In the implementation, a four-layer LSTM-AE model was employed with *tanh* as the activation function for compression. For recurrent activation, the *sigmoid* function was used with a $glorot\_uniform\_kernel$ as initializer. These parameters, including the *regularizer* were achieved after using Bayesian optimizer to search for the best combination. The LSTM architecture used in this study consists of the input, encoder, latent, and decoder layers. While the input layer defines the input layer of the network. It expects input sequences of 3D shape where the first dimension is the number of time steps, the second dimension is the number of features, and the third dimension is the feature dimensionality; the encoder layers, on the other hand, consisted of four (4) layers with 64, 32, and 16 units (also referred to as neurons or cells) in each LSTM layer. The last unit (16) is connected to the bottleneck

layer with an output restriction of five (5) nodes. The encoder layers process the input sequences and extract increasingly abstract representations of the input data. More importantly, the *return_sequences* parameter was set to True for all the layers except in the encoder bottleneck layer, which returns only the output at the last time-step. Because the aim is to obtain the compressed representation of the encoder, there was no need for reconstruction of the output at the decoder phase. The encoder output, achieved through the encoder architecture, is a compressed representation of the input data into a lower-dimensional space at the bottleneck layer. In determining the size of the nodes at the bottleneck layer, careful consideration was given to avoid over-constricting the data, which could lead to information discrimination loss, reduced model generalization, decreased model robustness, and limited adaptability, as noted by Liang (2011); Zeng et al. (2020).

7.3.4 Data splitting and Semi-Supervised Learning for Model training

A shallow deep learning was used at this stage, starting with the splitting of the deparameterized data representing the output of the encoder layer. The data were split into training and testing subsets, marking a crucial step in the adoption of semi-supervised learning approaches. The division of data into training and testing subsets adhered to a ratio of 70:30, allocating 70% for training and the remaining 30% for testing. Following this, the training data were further partitioned into three distinct subsets: X_labeled, y_labeled, and X_unlabeled. Simultaneously, the testing data were also split into two subsets of test_data and test_label. Before fitting shallow learning, however, the approach explained in Section 3.4.2 was implemented to enhance model generalization and effectiveness.

A Shallow Deep Neural Network (SDL), as detailed in section 3.7.2, was employed for model training, using the output of the LSTM-AE encoder's bottleneck layer as input. In training the dense model, we used the bottleneck layer output from the LSTM-AE encoder. In the dense layer, each neuron underwent a linear transformation of its input, followed by a non-linear activation function. This linear transformation involved computing a weighted sum of inputs from the previous layer, where each input was multiplied by an adjustable weight parameter. The architecture implemented at this stage comprised a simple network with an input layer, one hidden layer, and an output layer. The input layer consisted of 12 nodes, whereas the hidden and output layers had 2 and 1 nodes, respectively. With an input shape of 5 corresponding to the feature size, the Rectified

Linear Unit ($ReLU$) activation function was selected for both the input and hidden layers, setting negative inputs to zero and leaving positive inputs unchanged. The output layer employed the sigmoid activation function, mapping inputs to a range of 0–1, aligning with the binary class of the input data. Subsequently, the model was compiled using binary crossentropy as the loss function, $Adam$ as the optimizer, and accuracy as the metric. The model underwent training using the labeled dataset and was subsequently evaluated. Bayesian optimization (see Section 3.7.3) was employed to search for optimal parameters, which is especially beneficial for tuning hyperparameters in machine learning models and optimizing functions that are computationally expensive to evaluate using other approaches, such as grid search. The parameters that underwent tuning included the number of hidden layers and neurons, batch size, and activation functions. To this end, the evaluation was performed using an epoch of 300 and a batch size of 64. The results for both the benchmark and compressed datasets are presented in Tables 7.3 and 7.4.

Training and model fitting were performed on both the benchmark dataset (having 68 features) and the compressed dataset (having 5 features). The output of the evaluation of the test data shows that even with a reduced computational time, the memory footprint was still large, which makes it unfit for use as a lightweight model in the IoT. Therefore, there is a need to further reduce the model size and computation time through pruning.

7.3.5 Pruning and Deparameterisation:

During the process of pruning, the approaches highlighted in Section 3.6 were used. The dataset used here was a compressed dataset obtained after training using the shallow deep learning model. The weight parameter was pruned and started by cloning the model so that the weights of the original model were the same as those of the cloned model. Then, using a $pruning - schedule$ function Prechelt (1997) in the Python library, a class such as $tfmot.sparsity.keras.ConstantSparsity$ Singh et al. (2022) was used to specify the target sparsity level during training. The schedule maintains a constant sparsity level throughout the training. By introducing sparsity, the goal is to reduce the number of parameters in the model, thus leading to benefits such as reduced model size, increased efficiency, and lower energy consumption. Because the data have already been normalized using the min-max approach to a range of [0 & 1], with a threshold of 0.5 being the sparsity level, certain weights are therefore set to zero based on their magnitudes. From the results in Tables 7.3 and 7.4, some level of efficiency through high accuracy was achieved, but the size of the model was still very high, which gives credence to the

assertion by Li et al. (2016) which was to the effect that the success of model training is often accompanied by a significant increase in computation cost occasioned by parameter storage. Obviously, ignoring the increase in computation cost occasioned by the introduction of sparsity (0) would certainly not aid in the efforts to achieve a LIDS for the IoT. To this end, it became imperative to strip the structured sparsity patterns introduced during the pruning process. The process of stripping the parameters introduced by pruning is called deparameterization (Depar). This was necessary to further reduce the memory footprint.

Here, $tfmot.sparsity.keras.strip_pruning$ of TensorFlow Keras was used to address the challenges associated with model size. The application of this function involved the use of the $strip_pruning$ method, which is an integral part of the pruning process. Essentially, $strip_pruning$ systematically eliminates the sparsity-related elements from the model, selectively retaining the essential core layers adorned with pruned weights. Consequently, this procedure results in the creation of a deparameterized model that is characterized by enhanced lightness and operational efficiency compared with its original counterpart. The deparameterized model not only mitigates memory requirements but also streamlines subsequent optimization steps, making it more manageable for further analyses. In essence, the combined implementation of pruning and deparameterization serves as a judicious strategy for effectively addressing the challenge of excessive model size. This approach not only reduces the memory footprint but also potentially enhances performance. To summarize, the success of this methodology lies in its ability to prune the model for size reduction, followed by deparameterization to eliminate superfluous operations introduced during pruning. The resultant model is optimized and ready for subsequent phases of processing. The results of model training using the deparameterized model for both datasets are shown in Tables 7.3 and 7.4.

Table 7.3: MQTT-IoT-IDS2020 dataset metrics

	Metrics					
	Acc	***Precis***	***Recall***	***F1***	***Time***	***Memory***
Model on all 68 Features	99.96	100	100	100	1.7s	31.5MB
Pruned_Model (5 nodes)	99	99	100	100	1.05s	25MB
Depar(Pruned_model)	**99**	**99**	**100**	**100**	**0.932**s	**85**Kb

Table 7.4: CIC-DS2017 dataset metrics

	Metrics					
	Acc	***Precis***	***Recall***	***F1***	***Time***	***Memory***
Model on all 68 Features	97.8	96	100	97	1.7s	33.5MB
Pruned_Model (5 nodes)	98	96	100	97	1.2s	25MB
Depar(Pruned_model)	**98**	**96**	**100**	**97**	1.03s	**85**Kb

7.3.6 Further training and model resilience:

After successfully completing the pruning and deparameterization phases, the model underwent retraining using the output of the deparameterized model. Specifically denoted as mod_L, this retraining was executed on the labeled data (X_labeled and y_labeled) for 200 epochs using a batch_size of 64. Subsequently, mod_L was employed to predict labels for the X_unlabeled data, generating pseudo-labels known as y_pseudo. To create a unified dataset for training, both X_labeled and X_unlabeled data were concatenated alongside y_labeled and y_pseudo. Following a shuffle process to ensure a balanced combination of labeled and pseudo-labeled samples, the model underwent another training phase using the same number of epochs and batch_size. This strategic approach not only expanded the training dataset but also facilitated the model in refining its predictions and adapting to the pseudo-labeled data.

Furthermore, to ensure that the model is resilient to adversarial attacks such as label poisoning attacks, the *EllipticEnvelope* function was used for outlier detection. The function creates an instance of the *EllipticEnvelope* class, which the model uses to detect outliers. In adversarial training, detecting outliers or anomalies is crucial for the identification of malicious or poisoned data. Consequently, the fit method was then called on the *EllipticEnvelope* instance to train the outlier detection model on the training data (X_labeled). The fit method learns the parameters of the elliptic envelope representing the normal behavior of the data. Then, the *predict* method was used to obtain predictions from the trained outlier detector for each data point in the training data (X_labeled). Furthermore, a variable, *inlier*, was created to contain the predicted labels, where 1 indicates an is_inlier (normal data) and -1 indicates an outlier (potentially malicious or poisoned data). Following on, a new X_labeled was generated to filter the X_labeled in order to retain the data identified as inliers, while the outlier identified by the *EllipticEnvelope* model was removed (see equation 7.2). This step is crucial for removing potentially poisoned examples from the training set.

$$X\_labeled = X\_labeled[is\_inlier == 1] \tag{7.1}$$

Similarly, the same approach is applied to the y_labeled in order to filter the labels and retain the *inlier* data and to ensure that the labels corresponds to the filtered training data (X_labeled), (see equation 7.3).

$$y\_labeled = y\_labeled[is\_inlier == 1] \tag{7.2}$$

In summary, for adversarial attacks such as label poisoning attacks, this approach employs an $ellipticenvelope-based$ outlier detector to identify and remove potential outliers (poisoned data) from the training data, thus allowing for a more robust and secure adversarial training process. The result of the poisoned attack and the resilience of the model in comparison with the other results are displayed in Tables 7.7 and 7.8.

7.3.7 Quantisation

Quantization is a pivotal technique employed in deep learning to reduce the memory and computation requirements of neural network models without compromising their accuracy. At this point, the resilient deparameterized model was quantized. This method entails mapping continuous value ranges to discrete values, effectively diminishing the number of bits essential to represent each parameter or activation in the network. The TensorFlow Lite model was used to preserve the weight of the model. The following are the steps we followed in implementing QAT:

1. The deparameterized model was trained and optimised using float DEFAULT Byte quantisation, resulting in a q_model. This approach explored different optimization options using float 8 bytes, float 16 bytes, and float 32 bytes to select the most effective one.

2. The output (q_model) from the previous step was fine-tuned and trained for 30 epochs, followed by validation, resulting in a fine-tuned-q_model.

3. Finally, a TensorFlow Lite model was applied to the fine-tuned model to obtain a TFLite_model.

Starting with the deparameterized model, it was saved, and subsequently, the TFLite converter was employed to transform the model into the TensorFlow Lite format. The

resultant model is stored in the variable $tflite\_model$. Further optimization ensued by leveraging the default optimizations provided by TensorFlow Lite for the converter. Subsequently, a new TFLite converter was employed for a second conversion, this time with the application of quantization. The resulting quantized model was placed in the variable $tflite\_quant\_model$. As part of the Quantization Aware Training (QAT) approach, the TensorFlow Model Optimization library was imported, enabling the training of the model with a blend of both full-precision and quantized parameter versions. A function from the TensorFlow Model Optimization library denoted as $quantize\_model$, was employed to quantize the original model, thereby creating a quantization-aware model (q_aware_model). The quantized version was used during forward and backward passes, whereas full precision was employed during weight updates to counteract information loss from quantization errors. The incorporation of quantization awareness during training has several advantages. After 30 epochs of training with the training data, the quantization-aware model is recompiled. A new TFLite converter is instantiated, and optimization is applied using the default TensorFlow Lite optimizations. Ultimately, the TFLite converter converts the quantization-aware model, producing the quantized TensorFlow Lite model saved under the variable $tflite\_qaware\_model$.

In summary, the process is initiated by saving the deparameterized model and converting it into the TensorFlow Lite format. Subsequently, quantization is applied to the original model using the TensorFlow Model Optimization library. The quantization-aware model undergoes training and is then converted into a quantized TensorFlow Lite format. This comprehensive process prepares the model for implementation and inference on test data using the TensorFlow Lite interpreter, facilitating an effective evaluation of the results.

7.3.8 Model Deployment and Inferencing

In section 3.7.6, the deployment was carried out on both a Windows system and a Raspberry Pi 4 to validate the model's effectiveness. After the deployment, inference was performed immediately following the quantization process. Subsequently, the quantized model that had been saved was deployed on the Pi. Similarly, the test data resulting from the original data's division into training and testing sets was also deployed on the device for validation.

7.4 Results and Discussion

7.4.1 Result

This section provides an overview of the results obtained by evaluating the Lightweight Intrusion Detection model on two datasets. The evaluation process involved using the test data on a Raspberry Pi. Starting with the original benchmark dataset, which initially comprised 68 features, semi-supervised learning was applied. The tables in this section showcase the outputs derived from this process. Subsequently, feature extraction was performed, compressing the data down to 5 features. The presented results are based on the fitting of the model with the five extracted features. The sequential progression of the results involved pruning, deparameterization, and inferencing. Each phase contributed to the overall evaluation of the model performance. This study highlights how the OCFSDA model addresses the research questions and provides a comprehensive understanding of its performance and efficacy. Throughout the evaluation, various metrics, such as overall accuracy, precision, recall, F1-score, computation time, and model size, were recorded. The model was evaluated at different stages, including the evaluation of the model on all 68 features (benchmark data), the pruned data, the deparameterized model, and the inferencing. To further assess the model's performance, an ROC curve showing the Area Under the Curve (AUC) was plotted. In addition, the plots of the training and validation loss and accuracy were analyzed to check for overfitting or underfitting.

Table 7.5: MQTT-IoT-IDS2020 dataset Classification report

	Metrics					
	Acc	***Precis***	***Recall***	***F1***	***Time***	***Memory***
Model on all 68 Features	99.96	100	100	100	1.7s	315Kb
Pruned_Model (5 nodes)	99	99	100	100	1.05	255Kb
Depar(Pruned_model)	99	99	100	100	0.932s	85Kb
OCFSDA (propose model)	**99**	**100**	**98**	**99**	**0.30s**	**2Kb**

Table 7.6: CIC-IDS2017 dataset Classification report

	Metrics					
Model	***Acc***	***Precis***	***Recall***	***F1***	***Time***	***Memory***
Model on all 68 Features	97.8	96	100	97	1.7s	265Kb
Pruned_Model (5 nodes)	98	96	100	97	1.2s	223Kb
Depar(Pruned_model)	98	96	100	97	1.03s	85Kb
OCFSDA (propose model)	**97**	**95**	**100**	**97**	**0.12s**	**2Kb**

Following the methodologies that encompass feature selection and data compression, the subsequent stages of this experiment were evaluated to determine the reduction in computation cost, which is the mainstay of this study. The results of the evaluation of the two datasets (MQTT-IoT-IDS202 and CIC-IDS2017 datasets) are displayed in Tables 7.5 & 7.6. As previously provided, to ensure consistency and unbiasedness, the experiments were limited to semi-supervised learning because the use of unlabeled data can lead to better generalization of the model and can help mitigate the impact of label noise often found in real-world scenarios by using additional information from unlabeled data. The classification reports arising from the models are displayed in the tables. From the tables, the comparative assessment of overall accuracy between the LSTM model on the benchmark dataset and the proposed model demonstrates parity at 99% and 97.8%, respectively. This equivalence is corroborated by the convergence patterns depicted in Figure 7.3b, which affirms the absence of overfitting. The observed convergence indicates a minimal disparity between the training and loss metrics. Similarly, a critical comparison of the other matrices (precision, recall and F1-score) shows parity of the recall (sensitivity values) and F1-score for the OCFSDA model on the CIC-IDS2017 dataset and the 2% and 1% decline for the MQTT-IoT-IDS2020 dataset. Similarly, for precision, the OCFSDA model achieved 100% which is at par with the benchmark dataset model for the MQTT-IoT-IDS2020 dataset and achieved a 1% decline in the CIC-IDS2017 dataset. In terms of computation time and model size, considering the importance of computation cost in resource-constrained IoT devices, the model size, which indicates the amount of memory used, was drastically reduced from 315Kb and 265Kb to just 2Kb. Furthermore, the computation time significantly decreased to 0.3s and 0.12s, respectively. This is in contrast to the benchmark model, which required 1.7s for classification with memory sizes of 315 and 256Kb. The OCFSDA model achieves classification in considerably less time and effectively reduces the model size to 2Kb. As a result, the computation cost is significantly reduced compared with the approaches before the implementation of the OCFSDA model.

In addition, Figures 7.5 and 7.6 present ROC curves that visually demonstrate the OCFSDA model's ability to distinguish between the two classes. These curves showcase the model's discriminative prowess. To further evaluate the model's performance, Figures 7.3a and 7.4 display plots of the training and validation losses. These plots are crucial for assessing the potential concerns of overfitting or underfitting. This analysis holds paramount importance in understanding the behavior and efficacy of the model throughout the training and validation phases. The trajectory of the training loss plot

provides insights into the model's learning progression on the training data. Simultaneously, the validation loss plot indicates the model's ability to generalize to novel and unobserved data. A consistent downward trend in both the training and validation loss plots, as depicted in Figures 7.3a and 7.4, signifies the model's proficiency in refining predictions on unseen data. Conversely, an upward trend may suggest overfitting.

Figure 7.3: Plot showing training and validation loss (left) and the training and validation accuracies (right) for MQTT-IoT-IDS2020 dataset

7.4.2 Receiver Operating Characteristic (ROC) Curve

The Receiver Operating Characteristic (ROC) curve is a crucial visual representation of the performance of a binary classification model. It provides key insights into the model's classification accuracy, particularly through the Area Under the Curve (AUC). In Figure 7.5 & 7.6, for instance, when the AUC attains a value of 1 for the OCFSDA model on the MQTT-IoT-IDS2020 dataset, it indicates a flawless classification. Similarly, for the CIC-IDS2017 dataset, the OCFSDA model's AUC attains a value of 0.97 (97%), which indicates a near perfect classification of the values. Overall, this signifies the model's ability to perfectly differentiate between the benign and malicious classes. In other words, the model successfully identified positive and negative instances in the dataset, achieving 100% sensitivity (true positive rate) and 100% specificity (true negative rate).

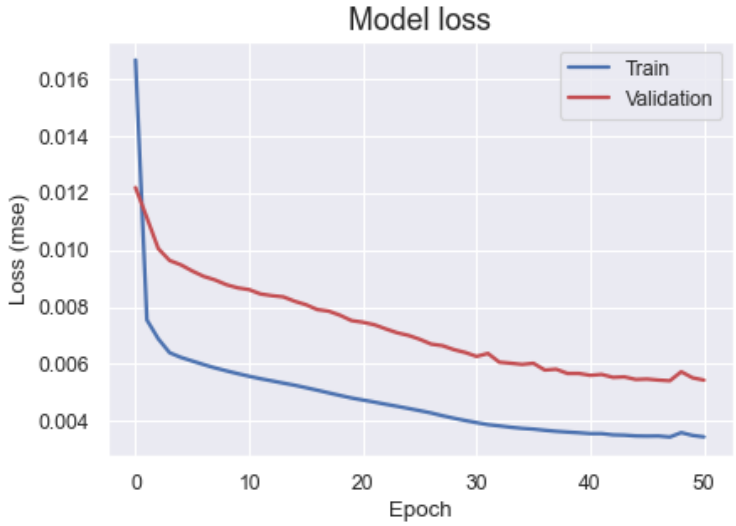


Figure 7.4: Plot showing training and validation loss for CIC-IDS2017 dataset

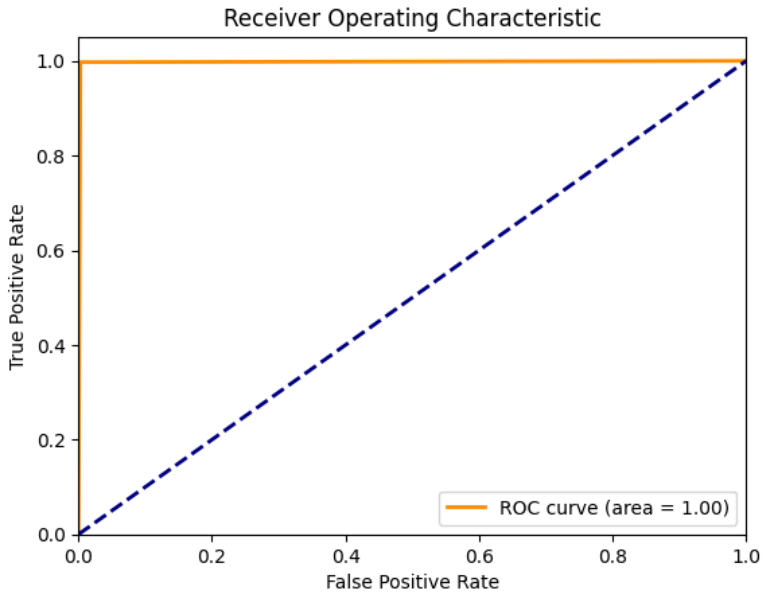


Figure 7.5: ROC curve showing the Area Under the Curve (AUC) for MQTT dataset.

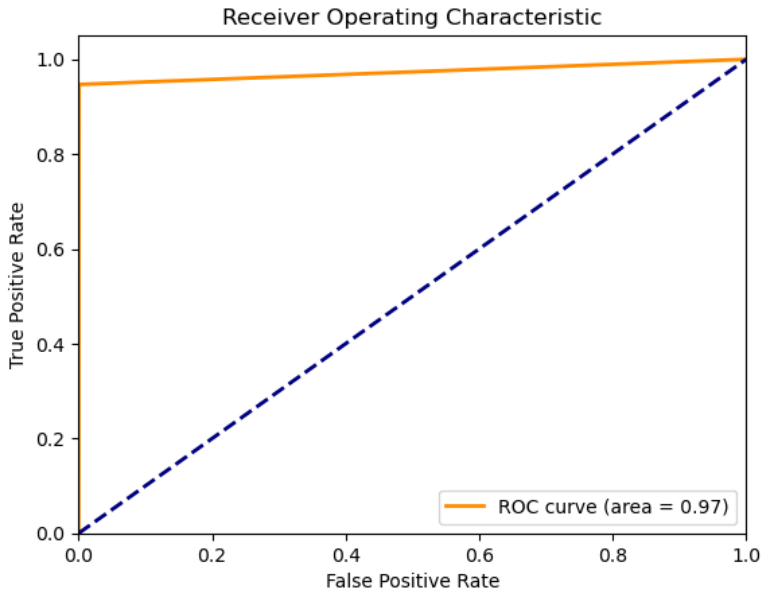


Figure 7.6: ROC curve showing the Area Under the Curve (AUC) for CIC-IDS2017 dataset

7.4.3 Model resilience against label Poison Attack

To assess the OCFSDA model's resilience against label poisoning attacks, an experiment was conducted by intentionally flipping three (3) label samples from their original label to the other Taheri et al. (2020) in the training dataset. The purpose of using this approach was to test the model's ability to withstand such attacks and to evaluate its performance on new and unseen data. The resilience of the OCFSDA model proved evident, particularly when its performance was compared with the performance of the model on the benchmark dataset with the same kind of attack. This was a targeted attack, and the impact of the measures included in the technique helped to reduce the impact. The reduction in the impact could be attributed to the integration of an outlier detection method within the robust learning technique (see section 7.3.7). This approach effectively identified and removed poisoned examples from the training data, thus contributing to the model's resilience. Additionally, the data augmentation method using pseudo-label data creation (as described in section 3.5.2) played a crucial role in enabling the OCFSDA model to generalize effectively and counter the impact of outliers. These combined techniques provide a clean and resilient training dataset that enhances the model's accuracy when evaluated on the testing dataset. The results showcasing the model's resilience against label poisoning attacks are presented in Tables 7.7 and 7.8.

Table 7.7: Table showing results before and after label poisoning attack on MQTT-IoT-IDS2020 dataset.

Model	Acc	Prec	Recall	F1	Time	Mem
Benchmark before poison Attack	99.96	100	100	97	1.7	315Kb
Benchmark after poison Attack	**79**	**43**	**94**	**64**	**1.73s**	**308Kb**
OCFSDA before Poison Attack	99	100	98	99	0.30s	2Kb
OCFSDA after poison Attack	**83**	**68**	**100**	**74**	**0.14s**	**2Kb**

Table 7.8: Table showing results before and after label poisoning attack on CIC-IDS2017 dataset.

Model	Acc	Prec	Recall	F1	Time	Mem
Benchmark before poison Attack	97.8	96	100	97	1.7	265Kb
Benchmark after poison Attack	**75.4**	**53**	**84**	**64.8**	**1.63s**	**255Kb**
OCFSDA before Poison Attack	97	95	100	97	0.12s	2Kb
OCFSDA after poison Attack	**78.2**	**63**	**100**	**76**	**0.13s**	**2Kb**

Tables 7.7 and 7.8 present a comparison of the results before and after a label poison attack as part of this thesis study. This comparison demonstrates the resilience of the LSTM model on the benchmark dataset and the OCFSDA model before and after the attack. The tables display the model results both before and after the poisoning attack. Table 7.7 shows that the overall accuracy of the benchmark dataset experienced a degradation of 20.9%, while the OCFSDA model showed a degradation of 16.16%. Similarly, for the CIC-IDS2017 dataset presented in Table 7.8, the classification rate of the LSTM model on the benchmark data degraded by 22.9%, while the OCFSDA model exhibited a degradation of 19.38% in the classification rate. A classical analysis of the tables provides that the OCFSDA model exhibited remarkable resilience compared to the benchmark model, with the degradation in classification rates translating to an improvement of approximately 4.74% and 3.52%, respectively, over the benchmark. Moreover, the OCFSDA model demonstrated notably higher recall rates, with accuracies of 100% in both datasets compared to 94% and 84% achieved by the benchmark model. This represents an improvement of 6% and 16% over the benchmark model's approach on the respective datasets. These findings underscore the robustness of the proposed OCFSDA model despite label poisoning attacks.

7.4.4 Comparison with other works

After successfully achieving resilience and developing a computationally efficient model, it became crucial to conduct a comparative analysis between the performance of the OCFSDA model and other relevant studies focusing on Lightweight Intrusion Detection in the context of the Internet of Things. To ensure objectivity, the comparison was specifically limited to results obtained from studies that used the same two datasets employed in this study. A detailed comparison can be found in Table 7.9.

Table 7.9: Model result comparison with other related approaches of other authors

	Metrics						
LID Model	***Ref***	***Acc***	***Precision***	***Recall***	***F1-score***	***Cmp_Time***	***Mem_size***
OCSVM	Ciklabakkal et al. (2019)	99	X	X	X	X	X
Isolation F.	Ciklabakkal et al. (2019)	84	X	X	X	X	X
SENMQTT	Siddharthan et al. (2022)	100	X	X	100	0.04s	X
NL_SVM-IoT	Jaafar et al. (2022b)	99.34	X	X	X	17.57s	X
DL-HIDS	Idrissi et al. (2022)	96.69	X	X	X	2e-6	2.704Kb
GAN-AE	Boppana and Bagade (2023b)	97.3	97.4	97.3	97.3	X	X
1D-DCNN	Rizvi et al. (2022)	99.7	X	X	X	X	X
SS-DEEP-ID	Abdel-Basset et al. (2021b)	99.6	99.48	99.23	99.35	1.1s	X
ELETL-IDS	Okey et al. (2023)	100	X	X	X	X	X
Self-Attention	Li (2022)	98.9	98	98.6	98.3	X	X
KD-TCNN	Wang et al. (2022b)	99.44	99.48	99.47	99.46	X	18.1Kb
HFS-KODE	Jaw and Wang (2021)	99.99	99.2	99.75	99.3	208s	X
IBGJO	Hanafi et al. (2023)	98.21	98.48	98.92	97.25	X	X
HDFEF	Li et al. (2022)	99.7	99.73	99.96	99.84	138.098s	X
OCFSDA	**Our model(MQTT-IoT-IDS20)**	**99**	**100**	**98**	**99**	**0.30s**	**2Kb**
OCFSDA	**Our model (CIC-IDS2017)**	**97**	**95**	**100**	**97**	**0.12s**	**2Kb**

Table 7.9 shows the results of some impressive studies in lightweight intrusion detection for resource-constrained devices, specifically in the context of IoT security. Improved overall accuracy has been achieved, but challenges persist in assessing important factors such as computational time and model size, which significantly influence computation cost. Among the notable approaches mentioned, Idrissi et al. (2022) attained an accuracy of 96% in a remarkably short computational time of 2μs, using a compact model size of 2.7KB. Furthermore, the deployment by the authors was on Arduino using a dual-core processor other than raspberry pi that was used in this study. In addition, the author's optimized model, which used five layers with 7 features in their work, tends to have a degrading overall accuracy from their benchmark model, which had 7 layers, 16 Features with an overall accuracy of 99.74% and a model size of 343Kb. Interestingly, this study did not disclose precision, recall, F1-score values, and the necessary ROC information, which could help make a more incisive comparison. Similarly, Siddharthan et al. (2022) achieved an overall accuracy and F1-score of 100%, with a computation time of 0.04s. However, this study also lacked essential scores for other critical metrics and omitted an overview of the model size. In a comprehensive evaluation, our OCFSDA approach to the two datasets delivered an overall accuracy of 99% and 97%. Furthermore, our proposed

model successfully generated precision, recall, and F1-score values within 0.30 and 0.12s, respectively, while maintaining a compact model size of 2Kb.

7.5 Conclusion

In conclusion, the proliferation of Internet of Things (IoT) devices has made them susceptible to various attacks due to inherent vulnerabilities. The constraints imposed by their protocols, power consumption, and memory footprint make conventional intrusion detection methods less effective. This study aimed to address the challenge of intrusion detection in resource-constrained IoT devices by developing a lightweight OCFSAD model through efficient dimension reduction techniques that encompass feature selection and extraction. Through the application of three feature selection techniques, relevant features correlated with the target feature were identified and selected. These features were extracted using the LSTM-autoencoder, compressing the output to five nodes. Further optimization, including pruning and deparameterization, eliminated unnecessary weights and sparsity. The model demonstrated resilience against attacks such as poison attacks. Quantization was leveraged to enhance inference efficiency, resulting in a TFLite model with reduced memory usage and faster inference times. Before inferencing, the TFLite interpreter, a component of TFLite, loaded and preprocessed the input data in line with the OCFSDA model's format and requirements. Subsequently, the interpreter facilitated running inferences, processing input and output data, and generating predictions. Deployed on a Raspberry Pi4 using semi-supervised learning, the OCFSDA model achieved remarkable overall accuracies of 99% and 97%, coupled with high performance across other evaluation metrics. Crucially, the OCFSDA Lightweight Intrusion Detection model successfully classified instances within 0.30 and 0.12s, using a mere 2KB of memory.

Chapter 8

Conclusion

In critical infrastructure, IoT devices are deployed in various locations, including remote and hard-to-reach areas. To ensure effective and computationally efficient intrusion detection of attacks on IoT devices in such remote locations, the integration of AI technologies is crucial. The integration of AI-based intrusion detection approaches is driven by the increasing incidence of cyber-attacks on IoT devices. However, the resource-constrained nature of IoT devices poses challenges for directly deploying AI techniques, necessitating the development of a computationally cost-effective and resilient model for IoT attack detection with high precision, minimal time, and minimal memory usage. To address these challenges, this thesis develops a model that meets the aforementioned requirements. The research questions at the core of this study have been effectively addressed through a series of approaches, as presented in the chapters.

One of the research questions addressed in this study is: How can effective generalization be achieved to enhance the intrusion detection of attacks on IoT devices? Given that class imbalance and low data regime frequently undermine cybersecurity and intrusion detection methodologies, a novel data augmentation approach called Sort-Augment and Combine (SAC) was introduced. SAC preserves the original data structure while oversampling the minority class(es) or the entire dataset to facilitate effective learning and prevent overfitting. By training multiple machine learning algorithms, the model demonstrated near-perfect classification accuracy, thereby providing a comprehensive response to research question one (RQ1).

Subsequently, addressing research question two (RQ2), which focuses on reducing data dimensionality to develop a lightweight intrusion detection model for resource-constrained IoT devices, an ensemble of feature selection techniques was utilized. This ensemble effectively ranked and selected a subset of non-redundant features based on their significance. Termed Common Feature Techniques (CFT), these selected features were integrated with additional learning algorithms, resulting in improved classification performance while reducing computational time and memory usage. The model's output on the CFT data effectively addresses RQ2.

Moreover, in order to refine dimensionality and enhance the CFT dataset's resilience and efficiency for IoT attack detection, compression and feature extraction were executed utilizing the encoder component of the Long-Short-Term-Memory (LSTM) Autoencoder. The resultant bottleneck layer, comprised of five nodes, ensured optimal representation, while a semi-supervised Shallow Deep Learning approach was adopted for model fitting. Additionally, the model underwent pruning and deparameterization, fortifying it against adversarial attacks. Bayesian optimization and quantization techniques were then applied for further model refinement before employing the TFLite Interpreter for inference. The deployment of the Optimized Lightweight Intrusion Detection model (OCFSDA) on both a Windows system and Raspberry Pi 4 yielded impressive outcomes, which form the foundation for research question three (RQ3). This question delves into the realm of IoT security, exploring how the strategies outlined in Question 2 (RQ2) can be optimized to bolster the resilience, efficiency, and overall performance of the intrusion detection model.

The OCFSDA model achieved remarkable overall accuracies of 99% and 97% on different datasets, along with high performance in other evaluation metrics. Crucially, the OCFSDA model demonstrated its efficiency by classifying instances within 0.30s and 0.12s, using only 2KB of memory. Furthermore, the model exhibited robustness against adversarial attacks, with overall accuracy and recall of 83% and 100% in 0.14s using 2KB of memory. In comparison, the benchmark dataset achieved an overall accuracy and recall of 79% and 94% in 0.83s and required 68KB of memory. Similarly, for the second dataset, the OCFSDA model achieved a resilient overall accuracy and recall of 78.2% and 100% in 0.13s using 2KB of memory against adversarial attacks, while the benchmark model achieved 75.4% and 84% overall accuracy and recall in 0.73s and required 65KB of memory.

8.1 Limitations

A few of the limitations encountered in the course of this study are highlighted below.

Feature selection and Data compression: Three feature selection techniques and an LSTM autoencoder were used to choose a subset of common features and extract features. While these methods were effective, the LSTM is known for longer computation time. Additionally, a significant constraint encountered when using the LSTM architecture is that the last 5 rows are often omitted in the output of the bottleneck layer when providing a sequence of step-by-step inputs based on the sample size, time-step, and number of features. To address this issue and ensure equal length between the output and the labels, the last 5 rows are omitted from the labels. This approach leads to unintended loss of data.

Dataset and Testbed: This study was limited to available datasets as against the initial plan of using a testbed for the simulation of attacks and demonstration of the model's robustness and resilience.

Adversarial attack In this study, the adversarial attack was limited to label poisoning by flipping the class labels. Incremental attacks were not performed to demonstrate the model's resilience against multiple or more rigorous attacks.

8.2 Future work

This study aimed to develop a computationally cost-effective model for efficient detection of IoT attacks, prioritizing minimal run-time and memory usage. The thesis chapters provide a clear outline of hypotheses, with Chapter 4 revealing the poor performance of the LDA, SVM, and KNN models in detecting Natural and Noevent classes. Although the attack-to-natural ratio of 1:3 indicates a relatively balanced dataset, it was crucial to investigate the cause and explore ways to improve classification in future work.

In Chapter 5, the SAC model approach was introduced and implemented through library-generated synthetic data and feature perturbations. These data augmentation approaches demonstrated greater effectiveness, with the models fitted on the approaches showing improved classification compared to other familiar techniques like ROSE and SMOTE. However, the SAC-library generated synthetic data struggled to match performance with models utilizing the ROSE and SMOTE oversampling techniques, except when combined with the ROSE augmented data, suggesting concatenation as a potential

avenue for future work across multiple datasets.

Chapter 6 focused on three feature selection techniques used to generate a subset of common features. Future work could broaden the scope by including additional approaches for feature selection, enabling the selection of optimal feature sets compatible with a wider range of learning algorithms.

In Chapter 7, the resilience of the proposed model against adversarial attacks, specifically data poisoning attacks, was tested. Future research should encompass various forms of adversarial attacks to provide a comprehensive evaluation of the model's robustness in different attack scenarios. Moreover, during the application of the LSTM-autoencoder for data compression and feature extraction, it was observed that the input sequence consistently fell short of some rows compared to the original dataset, with the last five rows consistently excluded. Thorough investigation and resolution of this issue are essential in future work to ensure accurate and complete feature extraction.

Bibliography

Abdel-Basset, M., Ding, W., and El-Shahat, D. (2021a). A hybrid harris hawks optimization algorithm with simulated annealing for feature selection. *Artificial Intelligence Review*, 54(1):593–637.

Abdel-Basset, M., Hawash, H., Chakrabortty, R. K., and Ryan, M. J. (2021b). Semi-supervised spatiotemporal deep learning for intrusions detection in iot networks. *IEEE Internet of Things Journal*, 8(15):12251–12265.

Abdulla, H., Al-Raweshidy, H. S., and Awad, W. S. (2023). Denial of service detection for iot networks using machine learning. In *ICAART (3)*, pages 996–1003.

Aburomman, A. A. and Reaz, M. B. I. (2016). Ensemble of binary svm classifiers based on pca and lda feature extraction for intrusion detection. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 636–640. IEEE.

Acharya, N. and Singh, S. (2018). An iwd-based feature selection method for intrusion detection system. *Soft Computing*, 22:4407–4416.

Act, P. (2001). Uniting and strengthening america by providing appropriate tools required to intercept and obstruct terrorism (usa patriot act) act of 2001. *Public Law*, 107:56.

Adamo, N., Al-Ansari, N., Sissakian, V., Laue, J., and Knutsson, S. (2021). Dam safety: hazards created by human failings and actions. *Journal of Earth Sciences and Geotechnical Engineering*, 11(1):65–107.

Adepu, S., Palleti, V. R., Mishra, G., and Mathur, A. (2020). Investigation of cyber attacks on a water distribution system. In *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18*, pages 274–291. Springer.

Aggarwal, C. C. and Aggarwal, C. C. (2017). *An introduction to outlier analysis*. Springer.

Aghakhani, H., Meng, D., Wang, Y.-X., Kruegel, C., and Vigna, G. (2021). Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. In *2021 IEEE European symposium on security and privacy (EuroS&P)*, pages 159–178. IEEE.

Ahmad, I., Wan, Z., and Ahmad, A. (2023). A big data analytics for ddos attack detection using optimized ensemble framework in internet of things. *Internet of Things*, 23:100825.

Alazzam, H., Sharieh, A., and Sabri, K. E. (2020). A feature selection algorithm for intrusion detection system based on pigeon inspired optimizer. *Expert systems with applications*, 148:113249.

Alghanam, O. A., Almobaideen, W., Saadeh, M., and Adwan, O. (2023). An improved pio feature selection algorithm for iot network intrusion detection system based on ensemble learning. *Expert Systems with Applications*, 213:118745.

Alladi, T., Chamola, V., Sikdar, B., and Choo, K.-K. R. (2020). Consumer iot: Security vulnerability case studies and solutions. *IEEE Consumer Electronics Magazine*, 9(2):17–25.

Almiani, M., AbuGhazleh, A., Al-Rahayfeh, A., Atiewi, S., and Razaque, A. (2020). Deep recurrent neural network for iot intrusion detection system. *Simulation Modelling Practice and Theory*, 101:102031.

Alrawais, A., Alhothaily, A., Hu, C., and Cheng, X. (2017). Fog computing for the internet of things: Security and privacy issues. *IEEE Internet Computing*, 21(2):34–42.

Anajemba, J. H., Tang, Y., Iwendi, C., Ohwoekevwo, A., Srivastava, G., and Jo, O. (2020). Realizing efficient security and privacy in iot networks. *Sensors*, 20(9):2609.

Anderson, J. P. (1980). Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company.*

Andrew, L. et al. (2020). The vulnerability of vital systems: how'critical infrastructure'became a security problem. In *Securing'the Homeland'*, pages 17–39. Routledge.

Anthi, E., Williams, L., and Burnap, P. (2018). Pulse: an adaptive intrusion detection for the internet of things.

Appavu, S., Rajaram, R., Nagammai, M., Priyanga, N., and Priyanka, S. (2011). Bayes theorem and information gain based feature selection for maximizing the performance of classifiers. In *Advances in Computer Science and Information Technology: First International Conference on Computer Science and Information Technology, CCSIT 2011, Bangalore, India, January 2-4, 2011. Proceedings, Part I 1*, pages 501–511. Springer.

Azhagusundari, B., Thanamani, A. S., et al. (2013). Feature selection based on information gain. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 2(2):18–21.

Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings.

Bedi, P., Mewada, S., Vatti, R. A., Singh, C., Dhindsa, K. S., Ponnusamy, M., and Sikarwar, R. (2021). Detection of attacks in iot sensors networks using machine learning algorithm. *Microprocessors and Microsystems*, 82:103814.

Beerman, J., Berent, D., Falter, Z., and Bhunia, S. (2023). A review of colonial pipeline ransomware attack. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, pages 8–15. IEEE.

Ben Brahim, A. and Limam, M. (2018). Ensemble feature selection for high dimensional data: a new method and a comparative study. *Advances in Data Analysis and Classification*, 12:937–952.

Bertino, E. and Islam, N. (2017). Botnets and internet of things security. *Computer*, 50(2):76–79.

Bicaku, A., Tauber, M., and Delsing, J. (2020). Security standard compliance and continuous verification for industrial internet of things. *International Journal of Distributed Sensor Networks*, 16(6):1550147720922731.

Boppana, T. K. and Bagade, P. (2023a). Gan-ae: An unsupervised intrusion detection system for mqtt networks. *Engineering Applications of Artificial Intelligence*, 119:105805.

Boppana, T. K. and Bagade, P. (2023b). GAN-AE: An unsupervised intrusion detection system for MQTT networks. *Engineering Applications of Artificial Intelligence*, 119:105805.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Burns, M. G. (2019). *Managing energy security: an all hazards approach to critical infrastructure*. Routledge.

Butun, I. and Gidlund, M. (2019). Location privacy assured internet of things. *ICISSP*, 19:1–8.

Butun, I., Österberg, P., and Gidlund, M. (2019a). Preserving location privacy in cyber-physical systems. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 1–6. IEEE.

Butun, I., Österberg, P., and Song, H. (2019b). Security of the internet of things: Vulnerabilities, attacks, and countermeasures. *IEEE Communications Surveys & Tutorials*, 22(1):616–644.

Chawla, S. and Thamilarasu, G. (2018). Security as a service: real-time intrusion detection in internet of things. In *Proceedings of the Fifth Cybersecurity Symposium*, pages 1–4.

Choi, S.-K., Yang, C.-H., and Kwak, J. (2018). System hardening and security monitoring for iot devices to mitigate iot security vulnerabilities and threats. *KSII Transactions on Internet & Information Systems*, 12(2).

Ciklabakkal, E., Donmez, A., Erdemir, M., Suren, E., Yilmaz, M. K., and Angin, P. (2019). ARTEMIS: An intrusion detection system for MQTT attacks in internet of things. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. IEEE.

Cil, A. Y., Abdurahman, D., and Cil, I. (2022). Internet of things enabled real time cold chain monitoring in a container port. *Journal of Shipping and Trade*, 7(1):9.

Çolak, M. and Irmak, E. (2023). A state-of-the-art review on electric power systems and digital transformation. *Electric Power Components and Systems*, 51(11):1089–1112.

Conteh, N. Y. and Royer, M. D. (2016). The rise in cybercrime and the dynamics of exploiting the human vulnerability factor. *International Journal of Computer (IJC)*, 20(1):1–12.

Coventry, L. and Branley, D. (2018). Cybersecurity in healthcare: A narrative review of trends, threats and ways forward. *Maturitas*, 113:48–52.

Cox, E., Bell, K., and Brush, S. (2022). Joint committee on the national security strategy inquiry: Critical national infrastructure and climate adaptation. *Paliamentary inquiry*.

Cvitić, I., Perakovic, D., Gupta, B. B., and Choo, K.-K. R. (2021). Boosting-based ddos detection in internet of things systems. *IEEE Internet of Things Journal*, 9(3):2109–2123.

Dag, A. Z., Johnson, M., Kibis, E., Simsek, S., Cankaya, B., and Delen, D. (2023). A machine learning decision support system for determining the primary factors impacting cancer survival and their temporal effect. *Healthcare Analytics*, 4:100263.

Dall'Ora, N., Centomo, S., and Fummi, F. (2019). Industrial-iot data analysis exploiting electronic design automation techniques. In *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 103–109. IEEE.

Damghani, H., Damghani, L., Hosseinian, H., and Sharifi, R. (2019). Classification of attacks on iot. In *4th international conference on combinatorics, cryptography, computer science and computation*.

Das, S., Saha, S., Priyoti, A. T., Roy, E. K., Sheldon, F. T., Haque, A., and Shiva, S. (2021). Network intrusion detection and comparative analysis using ensemble machine learning and feature selection. *IEEE transactions on network and service management*, 19(4):4821–4833.

Davahli, A., Shamsi, M., and Abaei, G. (2020). A lightweight anomaly detection model using svm for wsns in iot through a hybrid feature selection algorithm based on ga and gwo. *Journal of Computing and Security*, 7(1):63–79.

Demir, S. and Şahin, E. K. (2022). Evaluation of oversampling methods (over, smote, and rose) in classifying soil liquefaction dataset based on svm, rf, and naïve bayes. *Avrupa Bilim ve Teknoloji Dergisi*, (34):142–147.

Deng, L., Li, D., Yao, X., and Wang, H. (2019). Retracted article: mobile network intrusion detection for iot system based on transfer learning algorithm. *Cluster Computing*, 22:9889–9904.

Derhab, A., Guerroumi, M., Gumaei, A., Maglaras, L., Ferrag, M. A., Mukherjee, M., and Khan, F. A. (2019). Blockchain and random subspace learning-based ids for sdn-enabled industrial iot security. *Sensors*, 19(14):3119.

Derksen, S. and Keselman, H. J. (1992). Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables. *British Journal of Mathematical and Statistical Psychology*, 45(2):265–282.

Di Pinto, A., Dragoni, Y., and Carcano, A. (2018). Triton: The first ics cyber attack on safety instrument systems. *Proc. Black Hat USA*, 2018:1–26.

Dina, A. S., Siddique, A., and Manivannan, D. (2022). Effect of balancing data using synthetic data on the performance of machine learning classifiers for intrusion detection in computer networks. *IEEE Access*, 10:96731–96747.

Dini, P. and Saponara, S. (2021). Analysis, design, and comparison of machine-learning techniques for networking intrusion detection. *Designs*, 5(1):9.

Diro, A. A. and Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761–768.

Djenna, A., Harous, S., and Saidouni, D. E. (2021). Internet of things meet internet of threats: New concern cyber security issues of critical cyber infrastructure. *Applied Sciences*, 11(10):4580.

Djenna, A. and Saïdouni, D. E. (2018). Cyber attacks classification in iot-based-healthcare infrastructure. In *2018 2nd Cyber Security in Networking Conference (CSNet)*, pages 1–4. IEEE.

Dondossola, G., Szanto, J., Masera, M., and Nai Fovino, I. (2008). Effects of intentional threats to power substation control systems. *International journal of critical infrastructures*, 4(1-2):129–143.

Duda, R. O., Hart, P. E., et al. (2006). *Pattern classification*. John Wiley & Sons.

Elmubark, M., Karrar, A., and Hassan, N. (2019). Survey in anomaly and misuse intrusion detection system. *IOSR Journal of Engineering*, 9:65.

Escudero, C., Sicard, F., and Zamai, E. (2018). Process-aware model based idss for industrial control systems cybersecurity: approaches, limits and further research. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 605–612. IEEE.

Fadaee, M., Bisazza, A., and Monz, C. (2017). Data augmentation for low-resource neural machine translation. *arXiv preprint arXiv:1705.00440*.

Fatima, M., Rehman, O., and Rehman, I. M. (2023). Li-ids: An approach towards a lightweight ids for resource-constrained iot. In *2023 International Conference on Smart Applications, Communications and Networking (SmartNets)*, pages 1–6. IEEE.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Fenanir, S., Semchedine, F., and Baadache, A. (2019). A machine learning-based lightweight intrusion detection system for the internet of things. *Revue d'Intelligence Artificielle*, 33(3).

Foley, J., Moradpoor, N., and Ochen, H. (2020). Employing a machine learning approach to detect combined internet of things attacks against two objective functions using a novel dataset. *Security and Communication Networks*, 2020:1–17.

Friedman, V. (2018). On the edge: Solving the challenges of edge computing in the era of iot. *URL: https://data-economy. com/on-the-edge-solving-the-challenges-of-edge-computing-in-the-eraof-iot*.

Garmaroodi, M. S. S., Farivar, F., Haghighi, M. S., Shoorehdeli, M. A., and Jolfaei, A. (2020). Detection of anomalies in industrial iot systems by data mining: Study of christ osmotron water purification system. *IEEE Internet of Things Journal*, 8(13):10280–10287.

Garnett, R. (2023). *Bayesian optimization*. Cambridge University Press.

Ge, M., Syed, N. F., Fu, X., Baig, Z., and Robles-Kelly, A. (2021). Towards a deep learning-driven intrusion detection approach for internet of things. *Computer Networks*, 186:107784.

Genç, Z. A., Lenzini, G., and Ryan, P. (2017). The cipher, the random and the ransom: a survey on current and future ransomware. *Advances in Cybersecurity 2017*.

Gilchrist, A. (2017). *IoT Security Issues*. De Gruyter, Incorporated.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE.

Gorman, S. (2009). Electricity grid in us penetrated by spies. *The wall street journal*, 8(8).

Gowtham, M. and Pramod, H. (2021). Semantic query-featured ensemble learning model for sql-injection attack detection in iot-ecosystems. *IEEE Transactions on Reliability*, 71(2):1057–1074.

Gu, Q., Li, Z., and Han, J. (2012). Generalized fisher score for feature selection. *arXiv preprint arXiv:1202.3725*.

Guenther, J. and Sawodny, O. (2019). Feature selection for thermal comfort modeling based on constrained lasso regression. *IFAC-PapersOnLine*, 52(15):400–405.

Gunduz, M. Z. and Das, R. (2020). Cyber-security on smart grid: Threats and potential solutions. *Computer networks*, 169:107094.

Gupta, A., Pandey, O. J., Shukla, M., Dadhich, A., Mathur, S., and Ingle, A. (2013). Computational intelligence based intrusion detection systems for wireless communication and pervasive computing networks. In *2013 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–7. IEEE.

Gustineli, M. (2022). A survey on recently proposed activation functions for deep learning. *arXiv preprint arXiv:2204.02921*.

Haber, M. J. and Haber, M. J. (2020). *Privileged attack vectors*. Springer.

Halim, Z., Yousaf, M. N., Waqas, M., Sulaiman, M., Abbas, G., Hussain, M., Ahmad, I., and Hanif, M. (2021). An effective genetic algorithm-based feature selection method for intrusion detection systems. *Computers & Security*, 110:102448.

Han, J., Kamber, M., and Pei, J. (2012). 3 - data preprocessing. In Han, J., Kamber, M., and Pei, J., editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 83–124. Morgan Kaufmann, Boston, third edition edition.

Hanafi, A. V., Ghaffari, A., Rezaei, H., Valipour, A., and arasteh, B. (2023). Intrusion detection in internet of things using improved binary golden jackal optimization algorithm and lstm. *Cluster Computing*, pages 1–18.

Hasan, M., Islam, M. M., Zarif, M. I. I., and Hashem, M. (2019). Attack and anomaly detection in iot sensors in iot sites using machine learning approaches. *Internet of Things*, 7:100059.

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., and Sikdar, B. (2019). A survey on iot security: application areas, security threats, and solution architectures. *IEEE Access*, 7:82721–82743.

Heady, R., Luger, G., Maccabe, A., and Servilla, M. (1990). The architecture of a network level intrusion detection system. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States); New Mexico ....

Hindy, H., Tachtatzis, C., Atkinson, R., Bayne, E., and Bellekens, X. (2020). Mqtt-iot-ids2020: Mqtt internet of things intrusion detection dataset. *IEEE Dataport*.

Hoque, N., Singh, M., and Bhattacharyya, D. K. (2018). Efs-mi: an ensemble feature selection method for classification. *Complex & Intelligent Systems*, 4(2):105–118.

Hossain, E., Khan, I., Un-Noor, F., Sikander, S. S., and Sunny, M. S. H. (2019). Application of big data and machine learning in smart grid, and associated security concerns: A review. *Ieee Access*, 7:13960–13988.

Huang, H., Li, T., Ding, Y., Li, B., and Liu, A. (2023). An artificial immunity based intrusion detection system for unknown cyberattacks. *Applied Soft Computing*, 148:110875.

Huang, T., You, S., Zhang, B., Du, Y., Wang, F., Qian, C., and Xu, C. (2022). Dyrep: bootstrapping training with dynamic re-parameterization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 588–597.

Iaiani, M., Tugnoli, A., Bonvicini, S., and Cozzani, V. (2021). Analysis of cybersecurity-related incidents in the process industry. *Reliability Engineering & System Safety*, 209:107485.

Idrissi, I., Mostafa Azizi, M., and Moussaoui, O. (2021). A lightweight optimized deep learning-based host-intrusion detection system deployed on the edge for iot. *International Journal of Computing and Digital System*.

Idrissi, I., Moussaoui, O., and Azizi, M. (2022). A lightweight optimized deep learning-based host-intrusion detection system deployed on the edge for IoT. *International Journal of Computing and Digital Systems*, 11(1):209–216.

Jaafar, F., Malik, Y., Serre, J., Wang, H., and Wang, T. (2022a). Lightweight intrusion detection in mqtt based sensor network. In *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–9. IEEE.

Jaafar, F., Malik, Y., Serre, J., Wang, H., and Wang, T. (2022b). Lightweight intrusion detection in MQTT based sensor network. In *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE.

Jan, S. U., Ahmed, S., Shakhov, V., and Koo, I. (2019). Toward a lightweight intrusion detection system for the internet of things. *IEEE access*, 7:42450–42471.

Janiesch, C., Zschech, P., and Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, 31(3):685–695.

Janjua, Z. H., Vecchio, M., Antonini, M., and Antonelli, F. (2019). Irese: An intelligent rare-event detection system using unsupervised learning on the iot edge. *Engineering Applications of Artificial Intelligence*, 84:41–50.

Jaw, E. and Wang, X. (2021). Feature selection and ensemble-based intrusion detection system: an efficient and comprehensive approach. *Symmetry*, 13(10):1764.

JPT staff, . (2019). E&p notes (november 2019). *Journal of Petroleum Technology*, 71(11):18–22.

Jun, C. and Chi, C. (2014). Design of complex event-processing ids in internet of things. In *2014 sixth international conference on measuring technology and mechatronics automation*, pages 226–229. IEEE.

Jung, W., Zhao, H., Sun, M., and Zhou, G. (2020). Iot botnet detection via power consumption modeling. *Smart Health*, 15:100103.

Kasongo, S. M. and Sun, Y. (2020). Performance analysis of intrusion detection systems using a feature selection method on the unsw-nb15 dataset. *Journal of Big Data*, 7:1–20.

Khoei, T. T., Slimane, H. O., and Kaabouch, N. (2022). A comprehensive survey on the cyber-security of smart grids: Cyber-attacks, detection, countermeasure techniques, and future directions. *arXiv preprint arXiv:2207.07738*.

Koroniotis, N., Moustafa, N., and Sitnikova, E. (2020). A new network forensic framework based on deep learning for internet of things networks: A particle deep framework. *Future Generation Computer Systems*, 110:91–106.

Koroniotis, N., Moustafa, N., Sitnikova, E., and Turnbull, B. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796.

Kramer, O. and Kramer, O. (2016). Scikit-learn. *Machine learning for evolution strategies*, pages 45–53.

Kumar, A. and Lim, T. J. (2019). Edima: Early detection of iot malware network activity using machine learning techniques. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 289–294. IEEE.

Kumar, K., Zindani, D., and Davim, J. P. (2019). *Industry 4.0: developments towards the fourth industrial revolution*. Springer.

Laiton-Bonadiez, C., Branch-Bedoya, J. W., Zapata-Cortes, J., Paipa-Sanabria, E., and Arango-Serna, M. (2022). Industry 4.0 technologies applied to the rail transportation industry: A systematic review. *Sensors*, 22(7):2491.

LATA, K. (2019). *Data Augmentation and Synthetic Data Generation using Generative Adversarial Networks*. PhD thesis, NATIONAL INSTITUTE OF TECHNOLOGY, KURUKSHETRA KURUKSHETRA-136119.

Le, A., Loo, J., Chai, K. K., and Aiash, M. (2016). A specification-based ids for detecting attacks on rpl-based network topology. *Information*, 7(2):25.

Le, Q. V. et al. (2015). A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks. *Google Brain*, 20:1–20.

Lee, R. M., Assante, M. J., and Conway, T. (2014a). German steel mill cyber attack. *Industrial Control Systems*, 30(62):1–15.

Lee, T.-H., Wen, C.-H., Chang, L.-H., Chiang, H.-S., and Hsieh, M.-C. (2014b). A lightweight intrusion detection scheme based on energy consumption analysis in 6lowpan. In *Advanced Technologies, Embedded and Multimedia for Human-centric Computing: HumanCom and EMC 2013*, pages 1205–1213. Springer.

Lemley, J., Bazrafkan, S., and Corcoran, P. (2017). Smart augmentation learning an optimal data augmentation strategy. *Ieee Access*, 5:5858–5869.

Lewis, T. G. (2019). *Critical infrastructure protection in homeland security: defending a networked nation*. John Wiley & Sons.

Li, D. and Majd, N. E. (2023). Intrusion detection in iot leveraged by multi-access edge computing using machine learning. In *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 441–446. IEEE.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Li, J. (2022). Research on intrusion detect system of internet of things based on deep learning. In *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*, pages 55–58. IEEE.

Li, Y., Qin, T., Huang, Y., Lan, J., Liang, Z., and Geng, T. (2022). Hdfef: A hierarchical and dynamic feature extraction framework for intrusion detection systems. *Computers & Security*, 121:102842.

Li, Y., Xu, Y., Liu, Z., Hou, H., Zheng, Y., Xin, Y., Zhao, Y., and Cui, L. (2020). Robust detection for network intrusion of industrial iot based on multi-cnn fusion. *Measurement*, 154:107450.

Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403.

Liang, Y. (2011). Efficient temporal compression in wireless sensor networks. In *2011 IEEE 36th Conference on Local Computer Networks*, pages 466–474. IEEE.

Liao, L., Li, H., Shang, W., and Ma, L. (2022). An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(3):1–40.

Liu, H., Zhou, M., Lu, X. S., and Yao, C. (2018). Weighted gini index feature selection method for imbalanced data. In *2018 IEEE 15th international conference on networking, sensing and control (ICNSC)*, pages 1–6. IEEE.

Lobo, F. (2018). Upstream oil & gas cyber risk: Insurance technical review. *Lloyd's Market Assoc.: London, UK*.

Ma, Z., Liu, L., Meng, W., Luo, X., Wang, L., and Li, W. (2023). Adcl: Towards an adaptive network intrusion detection system using collaborative learning in iot networks. *IEEE Internet of Things Journal*.

Mafarja, M., Heidari, A. A., Habib, M., Faris, H., Thaher, T., and Aljarah, I. (2020). Augmented whale feature selection for iot attacks: Structure, analysis and applications. *Future Generation Computer Systems*, 112:18–40.

Maglaras, L. A., Kim, K.-H., Janicke, H., Ferrag, M. A., Rallis, S., Fragkou, P., Maglaras, A., and Cruz, T. J. (2018). Cyber security of critical infrastructures. *Ict Express*, 4(1):42–45.

Mahoney, M. V. and Chan, P. K. (2003). An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 220–237. Springer.

Manek, A. S., Shenoy, P. D., and Mohan, M. C. (2017). Aspect term extraction for sentiment analysis in large movie reviews using gini index feature selection method and svm classifier. *World wide web*, 20:135–154.

Manhas, J. and Kotwal, S. (2021). Implementation of intrusion detection system for internet of things using machine learning techniques. *Multimedia Security: Algorithm Development, Analysis and Applications*, pages 217–237.

Manokaran, J. and Vairavel, G. (2022). Smart anomaly detection using data-driven techniques in iot edge: a survey. In *Proceedings of Third International Conference on Communication, Computing and Electronics Systems: ICCCES 2021*, pages 685–702. Springer.

Markopoulou, D. and Papakonstantinou, V. (2021). The regulatory framework for the protection of critical infrastructures against cyberthreats: Identifying shortcomings and addressing future challenges: The case of the health sector in particular. *Computer law & security review*, 41:105502.

McHugh, J. (2000). Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294.

Meir, Y., Tevet, O., Tzach, Y., Hodassman, S., Gross, R. D., and Kanter, I. (2023). Efficient shallow learning as an alternative to deep learning. *Scientific Reports*, 13(1):5423.

Meurisch, C., Bayrak, B., Giger, F., and Mühlhäuser, M. (2020). Pdsproxy: Trusted iot proxies for confidential ad-hoc personalization of ai services. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–2. IEEE.

Miller, T., Staves, A., Maesschalck, S., Sturdee, M., and Green, B. (2021). Looking back to look forward: Lessons learnt from cyber-attacks on industrial control systems. *International Journal of Critical Infrastructure Protection*, 35:100464.

Mo, Y., Kim, T. H.-J., Brancik, K., Dickinson, D., Lee, H., Perrig, A., and Sinopoli, B. (2011). Cyber–physical security of a smart grid infrastructure. *Proceedings of the IEEE*, 100(1):195–209.

Mohammadi, S., Mirvaziri, H., Ghazizadeh-Ahsaee, M., and Karimipour, H. (2019). Cyber intrusion detection by combined feature selection algorithm. *Journal of information security and applications*, 44:80–88.

Mohee, A. (2022). A realistic analysis of the stuxnet cyber-attack.

Mokhtari, S., Abbaspour, A., Yen, K. K., and Sargolzaei, A. (2021). A machine learning approach for anomaly detection in industrial control systems based on measurement data. *Electronics*, 10(4):407.

Mollus, K., Westhoff, D., and Markmann, T. (2014). Curtailing privilege escalation attacks over asynchronous channels on android. In *2014 14th International Conference on Innovations for Community Services (I4CS)*, pages 87–94. IEEE.

Morris, T. and Gao, W. (2014a). Industrial control system traffic data sets for intrusion detection research. In Butts, J. and Shenoi, S., editors, *Critical Infrastructure Protection VIII*, pages 65–78, Berlin, Heidelberg. Springer Berlin Heidelberg.

Morris, T. and Gao, W. (2014b). Industrial control system traffic data sets for intrusion detection research. In *Critical Infrastructure Protection VIII: 8th IFIP WG 11.10 International Conference, ICCIP 2014, Arlington, VA, USA, March 17-19, 2014, Revised Selected Papers 8*, pages 65–78. Springer.

Morris, T., Pan, S., Lewis, J., Moorhead, J., Younan, N., King, R., Freund, M., and Madani, V. (2011a). Cybersecurity risk testing of substation phasor measurement units and phasor data concentrators. In *Proceedings of the seventh annual workshop on cyber security and information intelligence research*, pages 1–1.

Morris, T., Srivastava, A., Reaves, B., Gao, W., Pavurapu, K., and Reddi, R. (2011b). A control system testbed to validate critical infrastructure protection concepts. *International Journal of Critical Infrastructure Protection*, 4(2):88–103.

Moustafa, N. (2019). New generations of internet of things datasets for cybersecurity applications based machine learning: Ton_iot datasets. In *Proceedings of the eResearch Australasia Conference, Brisbane, Australia*, pages 21–25.

Murali, S. and Jamalipour, A. (2019). A lightweight intrusion detection for sybil attack under mobile rpl in the internet of things. *IEEE Internet of Things Journal*, 7(1):379–388.

Mushtaq, E., Zameer, A., Umer, M., and Abbasi, A. A. (2022). A two-stage intrusion detection system with auto-encoder and lstms. *Applied Soft Computing*, 121:108768.

Nagpal, D., Garg, G., and Babbar, H. (2023). Case studies and use case scenarios of cps-iot. In *2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, pages 1196–1202. IEEE.

Nam, S., Jeon, S., Kim, H., and Moon, J. (2020). Recurrent gans password cracker for iot password security enhancement. *Sensors*, 20(11):3106.

NCSC (2020). Denial of service (dos) guidance.

Neisse, R., Baldini, G., Steri, G., Ahmad, A., Fourneret, E., and Legeard, B. (2017). Improving internet of things device certification with policy-based management. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE.

NG, B. A. and Selvakumar, S. (2020). Anomaly detection framework for internet of things traffic using vector convolutional deep learning approach in fog environment. *Future Generation Computer Systems*, 113:255–265.

Ngo, Q.-D., Nguyen, H.-T., Le, V.-H., and Nguyen, D.-H. (2020). A survey of iot malware and detection methods based on static features. *ICT Express*, 6(4):280–286.

Nguyen, B. H., Xue, B., and Zhang, M. (2020). A survey on swarm intelligence approaches to feature selection in data mining. *Swarm and Evolutionary Computation*, 54:100663.

Nguyen, X.-H., Nguyen, X.-D., Huynh, H.-H., and Le, K.-H. (2022). Realguard: A lightweight network intrusion detection system for iot gateways. *Sensors*, 22(2):432.

Nõmm, S. and Bahşi, H. (2018). Unsupervised anomaly based botnet detection in iot networks. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1048–1053. IEEE.

Nowok, B., Raab, G. M., and Dibben, C. (2016). synthpop: Bespoke creation of synthetic data in r. *Journal of statistical software*, 74:1–26.

OConnor, T., Enck, W., and Reaves, B. (2019). Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 140–150.

Oh, D., Kim, D., and Ro, W. W. (2014). A malicious pattern detection engine for embedded security systems in the internet of things. *Sensors*, 14(12):24188–24211.

O'Hara, P. M. (2019). *Internet of things risks in the energy and healthcare and public health sectors of US critical infrastructure*. PhD thesis, Utica College.

Okey, O. D., Melgarejo, D. C., Saadi, M., Rosa, R. L., Kleinschmidt, J. H., and Rodríguez, D. Z. (2023). Transfer learning approach to ids on cloud iot devices using optimized cnn. *IEEE Access*, 11:1023–1038.

O'Ree, A. J. and Obaidat, M. S. (2011). Security enhancements for uddi. *Security and Communication Networks*, 4(8):871–887.

Otokwala, U., Petrovski, A., and Kalutarage, H. (2021). Improving intrusion detection through training data augmentation. In *2021 14th International Conference on Security of Information and Networks (SIN)*, volume 1, pages 1–8. IEEE.

Pajouh, H. H., Javidan, R., Khayami, R., Dehghantanha, A., and Choo, K.-K. R. (2016). A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks. *IEEE Transactions on Emerging Topics in Computing*, 7(2):314–323.

Pan, S., Morris, T., and Adhikari, U. (2015a). Developing a hybrid intrusion detection system using data mining for power systems. *IEEE Transactions on Smart Grid*, 6(6):3104–3113.

Pan, S., Morris, T. H., and Adhikari, U. (2015b). A specification-based intrusion detection framework for cyber-physical environment in electric power system. *Int. J. Netw. Secur.*, 17(2):174–188.

Panthong, R. and Srivihok, A. (2015). Wrapper feature subset selection for dimension reduction based on ensemble learning algorithm. *Procedia Computer Science*, 72:162–169.

Parra, G. D. L. T., Rad, P., Choo, K.-K. R., and Beebe, N. (2020). Detecting internet of things attacks using distributed deep learning. *Journal of Network and Computer Applications*, 163:102662.

Pasupa, K. and Sunhem, W. (2016). A comparison between shallow and deep architecture classifiers on small dataset. In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–6. IEEE.

Paudice, A., Muñoz-González, L., and Lupu, E. C. (2019). Label sanitization against label flipping poisoning attacks. In *ECML PKDD 2018 Workshops: Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings 18*, pages 5–15. Springer.

Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

Peri, N., Gupta, N., Huang, W. R., Fowl, L., Zhu, C., Feizi, S., Goldstein, T., and Dickerson, J. P. (2020). Deep k-nn defense against clean-label data poisoning attacks. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 55–70. Springer.

Pishva, D. (2017). Iot: their conveniences, security challenges and possible solutions. *Adv. Sci. Technol. Eng. Syst. J*, 2(3):1211–1217.

Pisner, D. A. and Schnyer, D. M. (2020). Support vector machine. In *Machine learning*, pages 101–121. Elsevier.

Pliatsios, D., Sarigiannidis, P., Lagkas, T., and Sarigiannidis, A. G. (2020). A survey on scada systems: secure protocols, incidents, threats and tactics. *IEEE Communications Surveys & Tutorials*, 22(3):1942–1976.

Pradhan, M., Nayak, C. K., and Pradhan, S. K. (2020). Intrusion detection system (ids) and their types. In *Securing the internet of things: Concepts, methodologies, tools, and applications*, pages 481–497. IGI Global.

Prechelt, L. (1997). Connection pruning with static and adaptive pruning schedules. *Neurocomputing*, 16(1):49–61.

Priya, D. D., Kiran, A., and Purushotham, P. (2022). Lightweight intrusion detection system (l-ids) for the internet of things. In *2022 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC)*, pages 1–4. IEEE.

Protogerou, A., Papadopoulos, S., Drosou, A., Tzovaras, D., and Refanidis, I. (2021). A graph neural network method for distributed anomaly detection in iot. *Evolving Systems*, 12(1):19–36.

Quincozes, S. E., Kazienko, J. F., and Quincozes, V. E. (2023). An extended evaluation on machine learning techniques for denial-of-service detection in wireless sensor networks. *Internet of Things*, 22:100684.

Rachburee, N. and Punlumjeak, W. (2015). A comparison of feature selection approach between greedy, ig-ratio, chi-square, and mrmr in educational mining. In *2015 7th international conference on information technology and electrical engineering (ICITEE)*, pages 420–424. IEEE.

Ramyachitra, D. and Manikandan, P. (2014). Imbalanced dataset classification and solutions: a review. *International Journal of Computing and Business Research (IJCBR)*, 5(4):1–29.

Rebuffi, S.-A., Gowal, S., Calian, D. A., Stimberg, F., Wiles, O., and Mann, T. A. (2021). Data augmentation can improve robustness. *Advances in Neural Information Processing Systems*, 34:29935–29948.

Resul, D. and Gündüz, M. Z. (2020). Analysis of cyber-attacks in iot-based critical infrastructures. *International Journal of Information Security Science*, 8(4):122–133.

Riggs, H., Tufail, S., Parvez, I., Tariq, M., Khan, M. A., Amir, A., Vuda, K. V., and Sarwat, A. I. (2023). Impact, vulnerabilities, and mitigation strategies for cyber-secure critical infrastructure. *Sensors*, 23(8):4060.

Rizvi, S., Scanlon, M., McGibney, J., and Sheppard, J. (2022). Deep learning based network intrusion detection system for resource-constrained environments. In *International Conference on Digital Forensics and Cyber Crime*, pages 355–367. Springer.

Rodofile, N. R., Radke, K., and Foo, E. (2017). Framework for scada cyber-attack dataset creation. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–10.

Rodríguez, D., Ruiz, R., Cuadrado-Gallego, J., and Aguilar-Ruiz, J. (2007). Detecting fault modules applying feature selection to classifiers. In *2007 IEEE International Conference on Information Reuse and Integration*, pages 667–672. IEEE.

Roy, S., Li, J., Choi, B.-J., and Bai, Y. (2022). A lightweight supervised intrusion detection mechanism for iot networks. *Future Generation Computer Systems*, 127:276–285.

Saba, T., Rehman, A., Sadad, T., Kolivand, H., and Bahaj, S. A. (2022). Anomaly-based intrusion detection system for iot networks through deep learning model. *Computers and Electrical Engineering*, 99:107810.

Saheed, Y. K., Abiodun, A. I., Misra, S., Holone, M. K., and Colomo-Palacios, R. (2022). A machine learning-based intrusion detection for detecting internet of things network attacks. *Alexandria Engineering Journal*, 61(12):9395–9409.

Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality

reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11.

Sanchez, O. R., Repetto, M., Carrega, A., Bolla, R., and Pajo, J. F. (2021). Feature selection evaluation towards a lightweight deep learning ddos detector. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE.

Sandri, M. and Zuccolotto, P. (2008). A bias correction algorithm for the gini variable importance measure in classification trees. *Journal of Computational and Graphical Statistics*, 17(3):611–628.

Sarhan, M., Layeghy, S., Moustafa, N., and Portmann, M. (2021). Netflow datasets for machine learning-based network intrusion detection systems. In *Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings 10*, pages 117–135. Springer.

Sayood, K. (2017). *Introduction to data compression*. Morgan Kaufmann.

Schneible, J. and Lu, A. (2017). Anomaly detection on the edge. In *MILCOM 2017-2017 IEEE military communications conference (MILCOM)*, pages 678–682. IEEE.

Schneier, B. (2017). Iot security: What's plan b? *IEEE Security & Privacy*, 15(05):96–96.

Scholkopf, B. and Smola, A. J. (2018). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Selim, G. E. I., Hemdan, E. E.-D., Shehata, A. M., and El-Fishawy, N. A. (2021). Anomaly events classification and detection system in critical industrial internet of things infrastructure using machine learning algorithms. *Multimedia Tools and Applications*, 80:12619–12640.

Sen, J. and Mehtab, S. (2020a). Introductory chapter: Machine learning in misuse and anomaly detection. *Computer and Network Security*, page 3.

Sen, J. and Mehtab, S. (2020b). Machine learning applications in misuse and anomaly detection. *Security and privacy from a legal, ethical, and technical perspective*, page 155.

Sha, K., Wei, W., Yang, T. A., Wang, Z., and Shi, W. (2018). On security challenges and open issues in internet of things. *Future generation computer systems*, 83:326–337.

Shafiq, M., Tian, Z., Sun, Y., Du, X., and Guizani, M. (2020). Selection of effective machine learning algorithm and bot-iot attacks traffic identification for internet of things in smart city. *Future Generation Computer Systems*, 107:433–442.

Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116.

Sharma, B., Sharma, L., and Lal, C. (2019). Anomaly detection techniques using deep learning in iot: a survey. In *2019 International conference on computational intelligence and knowledge economy (ICCIKE)*, pages 146–149. IEEE.

Sharma, B., Sharma, L., Lal, C., and Roy, S. (2024). Explainable artificial intelligence for intrusion detection in iot networks: A deep learning based approach. *Expert Systems with Applications*, 238:121751.

Siddharthan, H., Deepa, T., and Chandhar, P. (2022). Senmqtt-set: An intelligent intrusion detection in iot-mqtt networks using ensemble multi cascade features. *IEEE Access*, 10:33095–33110.

Siedlecki, W. and Sklansky, J. (1989). A note on genetic algorithms for large-scale feature selection. *Pattern recognition letters*, 10(5):335–347.

Sikri, A., Singh, N., and Dalal, S. (2023). Chi-square method of feature selection: Impact of preprocessing of data. *International Journal of Intelligent Systems and Applications in Engineering*, 11(3s):241–248.

Simon, T. (2017). Chapter seven: Critical infrastructure and the internet of things. *Cyber security in a volatile world*, 93.

Singh, S., Sharma, K., Karna, B. K., and Raj, P. (2022). Pruning and quantization for deeper artificial intelligence (ai) model optimization. In *International Conference on Robotics, Control, Automation and Artificial Intelligence*, pages 933–945. Springer.

Smith, D. C. (2018). Enhancing cybersecurity in the energy sector: a critical priority.

Stellios, I., Kotzanikolaou, P., Psarakis, M., Alcaraz, C., and Lopez, J. (2018). A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. *IEEE Communications Surveys & Tutorials*, 20(4):3453–3495.

Stergiopoulos, G., Gritzalis, D. A., and Limnaios, E. (2020). Cyber-attacks on the oil & gas sector: A survey on incident assessment and attack patterns. *IEEE Access*, 8:128440–128475.

Stoddart, K. (2022). Cyberwar: Attacking critical infrastructure. In *Cyberwarfare: Threats to Critical Infrastructure*, pages 147–225. Springer.

Sudqi Khater, B., Abdul Wahab, A. W. B., Idris, M. Y. I. B., Abdulla Hussain, M., and Ahmed Ibrahim, A. (2019). A lightweight perceptron-based intrusion detection system for fog computing. *applied sciences*, 9(1):178.

Summerville, D. H., Zach, K. M., and Chen, Y. (2015). Ultra-lightweight deep packet anomaly detection for internet of things devices. In *2015 IEEE 34th international performance computing and communications conference (IPCCC)*, pages 1–8. IEEE.

Sun, A., Lim, E.-P., and Liu, Y. (2009). On strategies for imbalanced text classification using svm: A comparative study. *Decision Support Systems*, 48(1):191–201.

Taheri, R., Javidan, R., Shojafar, M., Pooranian, Z., Miri, A., and Conti, M. (2020). On defending against label flipping attacks on malware detection systems. *Neural Computing and Applications*, 32:14781–14800.

Tang, B. and He, H. (2015). Kerneladasyn: Kernel based adaptive synthetic data generation for imbalanced learning. In *2015 IEEE congress on evolutionary computation (CEC)*, pages 664–671. IEEE.

Tanner, M. A. and Wong, W. H. (1987). The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee.

Tewari, A. and Gupta, B. B. (2020). Security, privacy and trust of different layers in internet-of-things (iots) framework. *Future generation computer systems*, 108:909–920.

Thanh Noi, P. and Kappas, M. (2017). Comparison of random forest, k-nearest neighbor, and support vector machine classifiers for land cover classification using sentinel-2 imagery. *Sensors*, 18(1):18.

Tharwat, A., Gaber, T., Ibrahim, A., and Hassanien, A. E. (2017). Linear discriminant analysis: A detailed tutorial. *AI communications*, 30(2):169–190.

Tian, Q., Li, J., and Liu, H. (2019). A method for guaranteeing wireless communication based on a combination of deep and shallow learning. *IEEE Access*, 7:38688–38695.

Torsæter, A. (2019). Innovation accounting-an empirical study of performance measurement in industrial research and development in the norwegian oil and gas sector. Master's thesis, NTNU.

Trinh, H. D., Zeydan, E., Giupponi, L., and Dini, P. (2019). Detecting mobile traffic anomalies through physical control channel fingerprinting: A deep semi-supervised approach. *IEEE Access*, PP:1–1.

Tyagi, A. K., Agarwal, K., Goyal, D., and Sreenath, N. (2020). A review on security and privacy issues in internet of things. *Advances in Computing and Intelligent Systems: Proceedings of ICACM 2019*, pages 489–502.

UK, G. (2017). Public summary of sector security and resilience plans. *Cabinet Office, London*.

Ullah, I. and Mahmoud, Q. H. (2021). Design and development of a deep learning-based model for anomaly detection in iot networks. *IEEE Access*, 9:103906–103926.

Utomo, D. and Hsiung, P.-A. (2019). Anomaly detection at the iot edge using deep learning. In *2019 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pages 1–2. IEEE.

Van Essen, B., Macaraeg, C., Gokhale, M., and Prenger, R. (2012). Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 232–239. IEEE.

Vargas, P. and Tien, I. (2023). Impacts of 5g on cyber-physical risks for interdependent connected smart critical infrastructure systems. *International Journal of Critical Infrastructure Protection*, 42:100617.

Vaz, R. (2019). Venezuela's power grid disabled by cyber attack. *Green Left Weekly*, 1213:15.

Viganò, E., Loi, M., and Yaghmaei, E. (2020). Cybersecurity of critical infrastructure. *The Ethics of Cybersecurity*, pages 157–177.

Villegas-Ch, W., Govea, J., and Jaramillo-Alcazar, A. (2023). Iot anomaly detection to strengthen cybersecurity in the critical infrastructure of smart cities. *Applied Sciences*, 13(19):10977.

Wang, J., Wang, M., Liu, Q., Yin, G., and Zhang, Y. (2022a). Deep anomaly detection in expressway based on edge computing and deep learning. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13.

Wang, J., Xu, J., Zhao, C., Peng, Y., and Wang, H. (2019). An ensemble feature selection method for high-dimensional data based on sort aggregation. *Systems Science & Control Engineering*, 7(2):32–39.

Wang, Z., Li, Z., He, D., and Chan, S. (2022b). A lightweight approach for network intrusion detection in industrial cyber-physical systems based on knowledge distillation and deep metric learning. *Expert Systems with Applications*, 206:117671.

Wankhade, K., Patka, S., and Thool, R. (2013). An efficient approach for intrusion detection using data mining methods. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1615–1618. IEEE.

Win, T. Z. and Kham, N. S. M. (2019). *Information gain measured feature selection to reduce high dimensional data*. PhD thesis, MERAL Portal.

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. (2020). Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*.

Xanthopoulos, P., Pardalos, P. M., Trafalis, T. B., Xanthopoulos, P., Pardalos, P. M., and Trafalis, T. B. (2013). Linear discriminant analysis. *Robust data mining*, pages 27–33.

Xu, Y., Tang, Y., and Yang, Q. (2020). Deep learning for iot intrusion detection based on lstms-ae. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Advanced Manufacture*, pages 64–68.

Yan, K. and Zhang, D. (2015). Feature selection and analysis on correlated gas sensor data with recursive feature elimination. *Sensors and Actuators B: Chemical*, 212:353–363.

Yu, K., Tan, L., Mumtaz, S., Al-Rubaye, S., Al-Dulaimi, A., Bashir, A. K., and Khan, F. A. (2021). Securing critical infrastructures: deep-learning-based threat detection in iiot. *IEEE Communications Magazine*, 59(10):76–82.

Yu, K., Yang, Y., and Ding, W. (2022). Causal feature selection with missing data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(4):1–24.

Zakariyya, I., Kalutarage, H., and Al-Kadri, M. O. (2023). Towards a robust, effective and resource efficient machine learning technique for iot security monitoring. *Computers & Security*, 133:103388.

Zebari, R., Abdulazeez, A., Zeebaree, D., Zebari, D., and Saeed, J. (2020). A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. *Journal of Applied Science and Technology Trends*, 1(2):56–70.

Zeng, D., Wu, Z., Ding, C., Ren, Z., Yang, Q., and Xie, S. (2020). Labeled-robust regression: simultaneous data recovery and classification. *IEEE Transactions on Cybernetics*, 52(6):5026–5039.

Zhang, H., Li, J.-L., Liu, X.-M., and Dong, C. (2021). Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection. *Future Generation Computer Systems*, 122:130–143.

Zhang, L., Liu, K., Xie, X., Bai, W., Wu, B., and Dong, P. (2023). A data-driven network intrusion detection system using feature selection and deep learning. *Journal of Information Security and Applications*, 78:103606.

Zhang, Y., Zhang, H., and Zhang, B. (2022). An effective ensemble automatic feature selection method for network intrusion detection. *Information*, 13(7):314.

Zhao, R., Gui, G., Xue, Z., Yin, J., Ohtsuki, T., Adebisi, B., and Gacanin, H. (2021). A novel intrusion detection method based on lightweight neural network for internet of things. *IEEE Internet of Things Journal*, 9(12):9960–9972.

Zheng, L., Zhang, H., Han, W., Zhou, X., He, J., Zhang, Z., Gu, Y., and Wang, J. (2022). Technologies, applications, and governance in the internet of things. In *Internet of Things-Global Technological and Societal Trends from Smart Environments and Spaces to Green ICT*, pages 143–177. River Publishers.

Zhou, Y., Cheng, G., Jiang, S., and Dai, M. (2020). Building an efficient intrusion detection system based on feature selection and ensemble classifier. *Computer networks*, 174:107247.

Zhu, M. and Gupta, S. (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

Zhu, T., Lin, Y., and Liu, Y. (2017). Synthetic minority oversampling technique for multiclass imbalance problems. *Pattern Recognition*, 72:327–340.

Table 1: **Appendix 1** - MQTT-IoT-IDS2020 Dataset Feature Ranking

	Feature Ranking			
	Dataset Features	***Info - Gain Ranking***	***Chi - Squared Ranking***	***Gini - Index Ranking***
1	Destination.Port	Total.Length.of.Fwd.Packets	URG.Flag.Count	Total.Length.of.Fwd.Packets
2	Flow.Duration	Subflow.Fwd.Bytes	min_seg_size_forward	Subflow.Fwd.Bytes
3	Total.Fwd.Packets	Average.Packet.Size	Destination.Port	Average.Packet.Size
4	Total.Backward.Packets	Bwd.Packet.Length.Mean	Min.Packet.Length	Destination.Port
5	Total.Length.of.Fwd.Packets	Destination.Port	Bwd.IAT.Total	Bwd.Packet.Length.Mean
6	Total.Length.of.Bwd.Packets	Avg.Bwd.Segment.Size	Fwd.Packet.Length.Std	Avg.Bwd.Segment.Size
7	Fwd.Packet.Length.Max	Total.Length.of.Bwd.Packets	Fwd.Packet.Length.Mean	Subflow.Bwd.Bytes
8	Fwd.Packet.Length.Min	Subflow.Bwd.Bytes	Avg.Fwd.Segment.Size	Total.Length.of.Bwd.Packets
9	Fwd.Packet.Length.Mean	Fwd.Header.Length.1	Bwd.IAT.Std	Fwd.Header.Length.1
10	Fwd.Packet.Length.Std	Fwd.Header.Length	Fwd.Packet.Length.Max	Fwd.Header.Length
11	Bwd.Packet.Length.Max	Bwd.Packet.Length.Max	Bwd.IAT.Max	Bwd.Packet.Length.Max
12	Bwd.Packet.Length.Min	Init_Win_bytes_forward	Total.Length.of.Fwd.Packets	Init_Win_bytes_forward
13	Bwd.Packet.Length.Mean	Fwd.Packet.Length.Mean	Subflow.Fwd.Bytes	Avg.Fwd.Segment.Size
14	Bwd.Packet.Length.Std	Avg.Fwd.Segment.Size	Fwd.PSH.Flags	Fwd.Packet.Length.Mean
15	Flow.Bytes.s	Fwd.Packet.Length.Max	SYN.Flag.Count	Fwd.Packet.Length.Max
16	Flow.Packets.s	Bwd.Header.Length	Bwd.Packet.Length.Min	Bwd.Header.Length
17	Flow.IAT.Mean	Fwd.IAT.Max	Bwd.IAT.Mean	Fwd.IAT.Max
18	Flow.IAT.Std	Fwd.IAT.Total	Init_Win_bytes_backward	Fwd.IAT.Total
19	Flow.IAT.Max	Fwd.IAT.Mean	Total.Fwd.Packets	Fwd.IAT.Mean
20	Flow.IAT.Min	Total.Fwd.Packets	Subflow.Fwd.Packets	Subflow.Fwd.Packets
21	Fwd.IAT.Total	Fwd.IAT.Std	Fwd.Header.Length	Total.Fwd.Packets
22	Fwd.IAT.Mean	Subflow.Fwd.Packets	Fwd.Header.Length.1	Init_Win_bytes_backward
23	Fwd.IAT.Std	Init_Win_bytes_backward	Fwd.Packet.Length.Min	Fwd.IAT.Std
24	Fwd.IAT.Max	Packet.Length.Mean	act_data_pkt_fwd	Packet.Length.Mean
25	Fwd.IAT.Min	act_data_pkt_fwd	Flow.Packets.s	act_data_pkt_fwd
26	Bwd.IAT.Total	Packet.Length.Std	Fwd.Packets.s	Packet.Length.Variance
27	Bwd.IAT.Mean	Packet.Length.Variance	Init_Win_bytes_forward	Packet.Length.Std
28	Bwd.IAT.Std	Bwd.IAT.Max	Bwd.Packet.Length.Mean	Bwd.IAT.Total
29	Bwd.IAT.Max	Bwd.IAT.Total	Avg.Bwd.Segment.Size	Fwd.Packet.Length.Std
30	Bwd.IAT.Min	Fwd.Packet.Length.Std	Bwd.Packet.Length.Max	Bwd.Packet.Length.Std
31	Fwd.PSH.Flags	Bwd.Packet.Length.Std	Active.Std	Bwd.Packets.s
32	Fwd.Header.Length	Bwd.Packets.s	Bwd.Header.Length	Bwd.IAT.Max
33	Bwd.Header.Length	Bwd.IAT.Mean	Bwd.Packet.Length.Std	Total.Backward.Packets
34	Fwd.Packets.s	Total.Backward.Packets	Bwd.Packets.s	Subflow.Bwd.Packets
35	Bwd.Packets.s	Subflow.Bwd.Packets	Bwd.IAT.Min	Fwd.Packets.s
36	Min.Packet.Length	Fwd.Packets.s	Fwd.IAT.Min	Max.Packet.Length
37	Max.Packet.Length	Max.Packet.Length	Total.Backward.Packets	Bwd.IAT.Mean
38	Packet.Length.Mean	Flow.Duration	Subflow.Bwd.Packets	Flow.Duration
39	Packet.Length.Std	Bwd.Packet.Length.Min	FIN.Flag.Count	Bwd.Packet.Length.Min
40	Packet.Length.Variance	Flow.Bytes.s	Down.Up.Ratio	Flow.Bytes.s
41	FIN.Flag.Count	Flow.IAT.Max	Average.Packet.Size	Flow.IAT.Max
42	SYN.Flag.Count	Flow.IAT.Std	Packet.Length.Mean	Flow.IAT.Std
43	RST.Flag.Count	Flow.IAT.Mean	Flow.IAT.Min	Flow.IAT.Mean
44	PSH.Flag.Count	Bwd.IAT.Std	Flow.Bytes.s	Flow.Packets.s
45	ACK.Flag.Count	Flow.Packets.s	Packet.Length.Std	Bwd.IAT.Std
46	URG.Flag.Count	Active.Min	Packet.Length.Variance	Active.Min
47	ECE.Flag.Count	Active.Mean	Max.Packet.Length	Bwd.IAT.Min
48	Down.Up.Ratio	Bwd.IAT.Min	Idle.Std	Active.Mean
49	Average.Packet.Size	Active.Max	Total.Length.of.Bwd.Packets	Active.Max
50	Avg.Fwd.Segment.Size	Fwd.IAT.Min	Subflow.Bwd.Bytes	Fwd.IAT.Min
51	Avg.Bwd.Segment.Size	Fwd.Packet.Length.Min	Fwd.IAT.Mean	Fwd.Packet.Length.Min
52	Fwd.Header.Length.1	Min.Packet.Length	RST.Flag.Count	URG.Flag.Count
53	Subflow.Fwd.Packets	URG.Flag.Count	ECE.Flag.Count	Min.Packet.Length
54	Subflow.Fwd.Bytes	Down.Up.Ratio	PSH.Flag.Count	Down.Up.Ratio
55	Subflow.Bwd.Packets	min_seg_size_forward	Idle.Max	min_seg_size_forward
56	Subflow.Bwd.Bytes	Flow.IAT.Min	Active.Min	Flow.IAT.Min
57	Init_Win_bytes_forward	Idle.Min	Flow.IAT.Max	Idle.Mean
58	Init_Win_bytes_backward	Idle.Max	Idle.Mean	Idle.Max
59	ac_data_pkt_fwd	Idle.Mean	Flow.IAT.Std	Idle.Min
60	min_seg_size_forward	PSH.Flag.Count	Active.Mean	PSH.Flag.Count
61	Active.Mean	Idle.Std	Flow.IAT.Mean	Idle.Std
62	Active.Std	Fwd.PSH.Flags	Fwd.IAT.Max	Fwd.PSH.Flags
63	Active.Max	SYN.Flag.Count	Idle.Min	SYN.Flag.Count
64	Active.Min	Active.Std	Fwd.IAT.Total	Active.Std
65	Idle.Mean	ACK.Flag.Count	Flow.Duration	ACK.Flag.Count
66	Idle.Std	FIN.Flag.Count	Active.Max	FIN.Flag.Count
67	Idle.Max	RST.Flag.Count	Fwd.IAT.Std	ECE.Flag.Count
68	Idle.Min	ECE.Flag.Count	ACK.Flag.Count	RST.Flag.Count

Table 2: **Appendix 2** - CIC-IDS2017 Dataset Feature Ranking

	Feature Ranking			
	Dataset Features	***Info - Gain Ranking***	***Chi - Squared Ranking***	***Gini - Index Ranking***
1	Destination Port	Packet Length Mean	PSH Flag Count	Avg Fwd Segment Size
2	Flow Duration	Packet Length Std	ACK Flag Count	Packet Length Std
3	Total Fwd Packets	Packet Length Variance	URG Flag Count	Flow Bytes/s
4	Total Backward Packets	Destination Port	Flow Duration	Destination Port
5	Total Length of Fwd Packets	min_seg_size_forward	Fwd IAT Total	min_seg_size_forward
6	Total Length of Bwd Packets	Average Packet Size	Bwd IAT Total	Average Packet Size
7	Fwd Packet Length Max	Init_Win_bytes_backward	Min Packet Length	Init_Win_bytes_backward
8	Fwd Packet Length Min	Init_Win_bytes_forward	Packet Length Mean	Init_Win_bytes_forward
9	Fwd Packet Length Mean	Flow Bytes/s	Avg Bwd Segment Size	Packet Length Variance
10	Fwd Packet Length Std	Subflow Fwd Bytes	Bwd Packet Length Mean	Subflow Fwd Bytes
11	Bwd Packet Length Max	Total Length of Fwd Packets	Init_Win_bytes_forward	Total Length of Fwd Packets
12	Bwd Packet Length Min	Fwd Packet Length Mean	Average Packet Size	Packet Length Mean
13	Bwd Packet Length Mean	Avg Fwd Segment Size	Packet Length Std	Fwd Packet Length Mean
14	Bwd Packet Length Std	Fwd Packet Length Max	Fwd PSH Flags	Fwd Packet Length Max
15	Flow Bytes/s	Subflow Bwd Bytes	SYN Flag Count	Subflow Bwd Bytes
16	Flow Packets/s	Total Length of Bwd Packets	Bwd Packet Length Std	Total Length of Bwd Packets
17	Flow IAT Mean	Bwd Packet Length Mean	Bwd Packet Length Max	Avg Bwd Segment Size
18	Flow IAT Std	Avg Bwd Segment Size	Flow IAT Max	Bwd Packet Length Mean
19	Flow IAT Max	Bwd Packet Length Max	Max Packet Length	Bwd Packet Length Max
20	Flow IAT Min	Bwd Packet Length Min	Fwd IAT Max	URG Flag Count
21	Fwd IAT Total	Bwd Packets/s	Init_Win_bytes_backward	Bwd Packets/s
22	Fwd IAT Mean	Flow Duration	Idle Max	Flow Duration
23	Fwd IAT Std	Flow IAT Max	Idle Mean	Flow IAT Max
24	Fwd IAT Max	Fwd Packets/s	Bwd IAT Max	Fwd Packets/s
25	Fwd IAT Min	Flow Packets/s	Bwd Packet Length Min	Flow Packets/s
26	Bwd IAT Total	Bwd Header Length	Idle Min	Bwd Header Length
27	Bwd IAT Mean	Flow IAT Mean	FIN Flag Count	Flow IAT Mean
28	Bwd IAT Std	Fwd Header Length	Fwd Packet Length Std	Total Fwd Packets
29	Bwd IAT Max	Fwd Header Length.1	Flow IAT Std	Fwd Header Length
30	Bwd IAT Min	Max Packet Length	Fwd Packet Length Mean	Max Packet Length
31	Fwd PSH Flags	PSH Flag Count	Avg Fwd Segment Size	PSH Flag Count
32	Fwd Header Length	Fwd Packet Length Min	Fwd IAT Std	Fwd IAT Mean
33	Bwd Header Length	Flow IAT Min	Fwd Packet Length Min	Flow IAT Min
34	Fwd Packets/s	Min Packet Length	Fwd Packet Length Max	Min Packet Length
35	Bwd Packets/s	Fwd IAT Max	Bwd IAT Mean	Fwd IAT Max
36	Min Packet Length	Fwd IAT Mean	Fwd IAT Mean	Subflow Bwd Packets
37	Max Packet Length	Fwd IAT Total	Packet Length Variance	Fwd IAT Total
38	Packet Length Mean	Fwd IAT Min	min_seg_size_forward	Fwd IAT Min
39	Packet Length Std	Total Fwd Packets	Bwd IAT Std	Fwd Header Length.1
40	Packet Length Variance	Subflow Fwd Packets	Bwd IAT Min	Subflow Fwd Packets
41	FIN Flag Count	Subflow Bwd Packets	Flow IAT Mean	Fwd Packet Length Min
42	SYN Flag Count	Total Backward Packets	Fwd IAT Min	Total Backward Packets
43	RST Flag Count	act_data_pkt_fwd	Bwd Packets/s	act_data_pkt_fwd
44	PSH Flag Count	Flow IAT Std	Total Length of Fwd Packets	Flow IAT Std
45	ACK Flag Count	Bwd IAT Max	Subflow Fwd Bytes	Bwd IAT Total
46	URG Flag Count	Bwd IAT Mean	Idle Std	Bwd IAT Max
47	ECE Flag Count	Bwd IAT Total	act_data_pkt_fwd	Bwd IAT Mean
48	Down/Up Ratio	Bwd IAT Min	Fwd Packets/s	Bwd IAT Min
49	Average Packet Size	Down/Up Ratio	Total Fwd Packets	Down/Up Ratio
50	Avg Fwd Segment Size	Fwd IAT Std	Subflow Fwd Packets	Fwd IAT Std
51	Avg Bwd Segment Size	Fwd Packet Length Std	Active Max	Fwd Packet Length Std
52	Fwd Header Length.1	ACK Flag Count	Destination Port	ACK Flag Count
53	Fwd Avg Bytes/Bulk	Bwd IAT Std	Bwd Header Length	Bwd IAT Std
54	Fwd Avg Packets/Bulk	Bwd Packet Length Std	Total Backward Packets	Bwd Packet Length Std
55	Fwd Avg Bulk Rate	Active Mean	Subflow Bwd Packets	Active Mean
56	Bwd Avg Bytes/Bulk	Idle Max	Fwd Header Length	Idle Mean
57	Bwd Avg Packets/Bulk	Active Max	Fwd Header Length.1	Active Max
58	Bwd Avg Bulk Rate	Idle Min	Total Length of Bwd Packets	Active Min
59	Subflow Fwd Packets	Idle Mean	Subflow Bwd Bytes	Idle Min
60	Subflow Fwd Bytes	Active Min	Active Std	Idle Max
61	Subflow Bwd Packets	Idle Std	Active Mean	Idle Std
62	Subflow Bwd Bytes	Active Std	Active Min	Active Std
63	Init_Win_bytes_forward	URG Flag Count	Down/Up Ratio	Bwd Packet Length Min
64	Init_Win_bytes_backward	SYN Flag Count	Flow IAT Min	Fwd PSH Flags
65	act_data_pkt_fwd	Fwd PSH Flags	RST Flag Count	SYN Flag Count
66	min_seg_size_forward	FIN Flag Count	ECE Flag Count	FIN Flag Count
67	Active Mean	RST Flag Count	Flow Bytes/s	ECE Flag Count
68	Active Std	ECE Flag Count	Flow Packets/s	RST Flag Count

Table 3: **Appendix 3** - Gas pipeline Dataset Feature Ranking

	Feature Ranking			
	Info - Gain	***Chi - Squared***	***Gini - Index***	***Common Features***
1	resp$_{r}ead_{f}un$	response$_{a}ddress$	resp$_{r}ead_{f}un$	setpoint
2	setpoint	response$_{m}emory$	setpoint	response$_{a}ddress$
3	response$_{a}ddress$	response$_{m}emory_{c}ount$	control$_{m}ode$	resp$_{r}ead_{f}un$
4	response$_{m}emory$	resp$_{w}rite_{f}un$	resp$_{w}rite_{f}un$	resp$_{w}rite_{f}un$
5	response$_{m}emory_{c}ount$	resp$_{l}ength$	response$_{a}ddress$	response$_{m}emory$
6	resp$_{w}rite_{f}un$	setpoint	response$_{m}emory_{c}ount$	response$_{m}emory_{c}ount$
7	resp$_{l}ength$	sub$_{f}unction$	resp$_{l}ength$	resp$_{l}ength$
8	control$_{m}ode$	control$_{s}cheme$	response$_{m}emory$	control$_{m}ode$
9	time	time	measurement	time
10	measurement	control$_{m}ode$	time	measurement
11	command$_{a}ddress$	resp$_{r}ead_{f}un$	command$_{a}ddress$	command$_{a}ddress$
12	control$_{s}cheme$	command$_{a}ddress$	control$_{s}cheme$	
13	pump	measurement	comm$_{r}ead_{f}unction$	
14	sub$_{f}unction$	comm$_{r}ead_{f}unction$	sub$_{f}unction$	
15	comm$_{r}ead_{f}unction$	command$_{m}emory_{c}ount$	pump	

Table 4: **Appendix 4** - BoT-IoTID20 Dataset Feature Ranking

	Feature Ranking			
	Info - Gain	***Chi - Squared***	***Gini - Index***	***Common Features***
1	Dst_Port	Fwd_Pkt_Len_Std	Dst_Port	Dst_Port
2	Src_Port	Dst_Port	Flow_Duration	Src_Port
3	Flow_Duration	Pkt_Len_Std	Bwd_Pkts.s	Flow_Duration
4	Init_Bwd_Win_Byts	Pkt_Len_Var	Flow_IAT_Max	TotLen_Bwd_Pkts
5	TotLen_Bwd_Pkts	Init_Bwd_Win_Byts	Src_Port	Fwd_Pkt_Len_Max
6	Subflow_Bwd_Byts	Fwd_Header_Len	Bwd_IAT_Tot	Fwd_Pkt_Len_Min
7	Pkt_Size_Avg	Subflow_Fwd_Byts	Flow_IAT_Mean	Bwd_Pkt_Len_Mean
8	TotLen_Fwd_Pkts	TotLen_Fwd_Pkts	Flow_IAT_Min	Bwd_Pkt_Len_Min
9	Subflow_Fwd_Byts	Fwd_Pkts.s	Bwd_Header_Len	Pkt_Len_Max
10	Fwd_Pkt_Len_Mean	ECE_Flag_Cnt	Flow_IAT_Std	Bwd_Pkts.s
11	Fwd_Seg_Size_Avg	Bwd_PSH_Flags	Bwd_IAT_Max	Flow_IAT_Mean
12	Fwd_Pkt_Len_Max	PSH_Flag_Cnt	Fwd_IAT_Tot	
13	Pkt_Len_Mean	Protocol	Pkt_Len_Max	
14	Fwd_Pkt_Len_Min	Bwd_Pkts.s	TotLen_Bwd_Pkts	
15	Bwd_Pkt_Len_Mean	Fwd_Act_Data_Pkts	Bwd_Pkt_Len_Max	
16	Bwd_Seg_Size_Avg	Subflow_Bwd_Pkts	Fwd_Pkt_Len_Std	
17	Flow_IAT_Max	Tot_Bwd_Pkts	Bwd_Pkt_Len_Min	
18	Idle_Max	Subflow_Fwd_Pkts	Bwd_IAT_Min	
19	Bwd_Pkt_Len_Min	Tot_Fwd_Pkts	Fwd_IAT_Max	
20	Pkt_Len_Max	Pkt_Size_Avg	Fwd_Header_Len	
21	Bwd_Pkt_Len_Max	Fwd_Pkt_Len_Max	Fwd_Pkt_Len_Min	
22	Pkt_Len_Min	Flow_Duration	Fwd_Pkt_Len_Max	
23	Bwd_Pkts.s	Bwd_Pkt_Len_Std	Pkt_Len_Min	
24	Flow_IAT_Mean	Flow_IAT_Mean	Bwd_Pkt_Len_Mean	