ADDIS ABABA UNIVERSITY
COLLEGE OF NATURAL SCIENCES

A Framework for Detecting Multiple Cyberattacks in IoT Environment

Yonas Mekonnen

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

25th February 2025

Addis Ababa University

College of Natural Sciences

Yonas Mekonnen

Advisor: Mesfin Kifle (PhD)

This is to certify that the thesis prepared by Yonas Mekonnen, titled: *A Framework for Detecting Multiple cyberattacks in IoT Environment* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name Signature Date

Advisor: Mesfin Kifle (PhD)

Examiner: Fekede Getahun (PhD)

Examiner: Minale Ashagrie (PhD)

Abstract

The Internet of Things refers to the growing trend of embedding ubiquitous and pervasive computing capabilities through sensor networks and internet connectivity. The growth and expansion of newly evolved cyberattacks, network patterns and heterogeneous nature of cyberattacks trend has become the warfare across the globe and challenges to apply single layer cyberattacks detection techniques to the Internet of Things. This research work identified the lack of cyberattacks detection framework as the major gap for detection of multiple cyberattacks such as denial of services, distributed denial of services, and Mairi attacks while it includes multiple parameters at the same time.

The proposed framework contains three modules; data acquisition and preprocessing module that is responsible for capturing and pre-processing the captured data and ready for the construction of the model, then the attack detection module which is the core engine that orchestrates the detection of cyberattacks, the third module notifies and displays the results in a dashboard. This research study used multiple parameters including multiple attack classes, network packet patterns, and three scaler types namely no scaler, MinMax, and Standard, and regardless of the defined parameters used, minmax scaler followed by standard scaler gives better detection performance than models trained with no scaler.

The proposed framework is trained and evaluated with different models including CNN, Hybrid, FFNN, and LSTM provides a result of 91.42%, 82.75%, and 78.38% ,74.83% detection accuracy respectively where it is observed that CNN model outperforms the optimal results among followed by hybrid and FFNN.

Keywords: IoT Environments, Cyberattacks, Multiple Attack Detection, Framework

Acknowledgments

I am deeply grateful to my advisor, Dr. Mesfin Kifle, for his invaluable help in resolving the technical complexities of the research, offering insightful feedback and constructive criticism, advocating for the use of precise grammar and consistent notation in my writing, and thoroughly reviewing and commenting on multiple versions of this document. His tenacity and support empowered me to overcome numerous arduous situations and successfully complete my thesis.

I am extremely grateful to the people, including my family and friends, who have been instrumental in facilitating the completion of my thesis. Sisay, your unwavering advice, assistance, and support are memorable.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

ANN	Artificial Neural Network
API	Application Program Interface
CNN	Convolutional Neural Network
CSV	Comma Separated Version
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
DnRaNN	Dense Random Neural Network
DoS	Denial of Service
DSR	Design Science Research
FFNN	Feeded-Forward Neural Network
IDS	Intrusion Detection System
IIoT	Industrial IoT
IoV	Internet of Vehicles
LSTM	Long-Short Term Memory
MFA	Multi-Factor Authentication
MiTM	Man-in-the-Middle
ML	Machine Learning
MLib	Machine Learning Library
MLP	Multi-Layer Perceptron
PCAP	Packet Capture
RCE	Remote Code Execution
RFI	Remote File Inclusion
RFID	Radio Frequency Identification
RNNs	Recurrent Neural Networks
SDIoT	Software Defined IoT

SDN	Software Defined Network
SQL	Structural Query Language
SSL	Secure Socket Layer
TPR	True Positive Rate
WSNs	Wireless Sensor Networks
XEE	XML External Entity
XSS	Cross-Site Scripting

Chapter 1 Introduction

1.1 Overview

Nowadays our daily activities are highly dependent on enterprise technologies and applications, and more importantly, internet access has a multifaceted social impact across the globe. IoT [1] is the phenomenon of pervasive computing, the growing trend of embedding computational capability, data-collecting sensors, and internet connectivity into everyday objects. The author [2] stated by the year 2020, the number of interconnected objects will exceed thirty billion. In addition, [3] stated that there would be a broader internet that encompasses people, objects, and information, and that this "Internet of Everything" will replace the current "Internet of Things."

IoT has a significant benefit to our lives [4] because it improves our ability to interconnect things and processes data in diverse fields like manufacturing, transportation, healthcare, agriculture, and aiding customers and retailers in making crucial decisions.

IoT technology maturity has becoming increasingly advanced [5] however, the privacy and security challenges are still immature which introduces new difficulties [6, 7] including constrained devices, dynamic and heterogeneous networks, battery autonomy, and insecure communication, which consequently make it harder to secure data.

IoT cyberattacks, such as DoS, Recon, SQL injection, DDoS, MiTM, and XSS, modify packets as they are being transmitted over a network, changing the content of the messages and using up resources. These attacks have a direct impact on the flow of packets and can carry out packet queuing attacks [8].
Previous research, such as [9], proposed a cross-layer attack detection framework. However, it lacks detecting multiple cyberattack class categories while applying multiple parameters and the detection pattern is based on network-flow which have a drawback for example, to generate features in a flow system, the flow must end with a termination flag or timeout.

In this research study, the notion of multiple attack detection refers to identifying different IoT cyberattacks by applying two or more attack type categories, two or more window-sizes, and two or more scaler types by applying two or more machine learning models.

1.2 Motivation

For real-time applications, it is essential to deal with concerns including privacy, trust, identification, and security [10, 11]. Security and privacy worries [12, 13] also slow the expansion of the IoT. Intruders often use commonplace goods to gain access to sensitive information. Instances of data breaches and unauthorized access can result in significant financial losses due to the misuse of individuals' personal information. IoT devices also give rise to concerns regarding security and privacy [14, 15]. Users should possess the capacity to limit access to their personal information and grant permission solely to specific individuals for its utilization.

The increase of venerability and attacks exposure on IoT environments requires an innovative cyberattacks detection framework. Situations such as [16, 17, 18] IoT applications such as smart city traffic management, smart traffic management, and smart hospital a single attack detection framework (for example only DDoS detection, or Botnet detection) would be insufficient to safeguard against the diverse range of IoT cyberattacks. This is because IoT has become an integral part of society and existing research studies are based on a single framework which only handles specific attacks, but the real-world demand requires a holistic end-to-end multiple cyberattack detection. In addition, the heterogeneity of IoT setups, characterized by numerous devices, protocols, and communication channels, requires the utilization of varied multiple detection approaches.

Therefore, the aim of this thesis is to design a framework detecting multiple IoT cyberattacks by incorporating a multi-faceted approach including multiple window-sizes, multiple scalers, multiple attack class, and multiple machine learning models.

1.3 Statement of the Problem

IoT technology becoming increasingly advanced [10, 11] which is an integral part of societies and communities however, the privacy and security challenges are still immature which introduces new and emerging technologies Example insecure communication, which consequently make it harder to secure data and application services. Products owned by consumers are being attacked by cybercriminals, thereby facilitating the occurrence of unauthorized access and data breaches [14, 15].

Previous research, such as [9], proposed a cross-layer attack detection framework. However, it lacks detecting multiple cyberattack class categories while applying multiple

parameters. The study proposed by [19] focus on IoT cyberattacks however, the proposed framework did not address and ignore and lack to detect multiple IoT related attack patterns and data imbalance issue. Again, a scientific research studies such as [20, 21, 22] has focused on detection on classical intrusion detection systems for industrial IoT services however, these studies had an issue in detection of multiple emerging IoT attack types based on the IoT environment profiling and behaviors.

To the best of my knowledge, there is no research study that detects multi-faced IoT cyberattacks security detection framework in IoT arena. The proposed framework by existing researchers such as [10, 11] do dose not adhered to established (user, and packet behavioral attack patterns), feature extraction method used is based on flow-based (to generate features in a flow system, the flow must end with a termination flag or timeout. for a flow to be identified as an attack, it is necessary to wait for it to be terminated first. Meaning only after the flow ends can the statistics required for attack detection be calculated), attack class categories are limited to single attack class categories, the dataset used (older version and not accurately represent evolving cyberattacks).

Hence, these research studies aim to design a framework for detecting multiple cyberattacks in IoT environments by addressing the following research questions.

- Which parameter values optimize the detection of multiple cyberattacks in IoT enviroments?
- How do the accuracy and efficiency of trained models compare when detecting multiple types of cyberattacks?

1.4 Objectives

General Objective

The general objective of the study is to design a framework for detecting multiple cyberattacks in IoT environments.

Specific Objectives

The specific objective of the study is described as follows.

- Conduct literature review by analyzing and providing a detailed description of the various cyberattacks that take place in IoT environments

- Examine and assess current IoT frameworks, methods, and tools for detecting IoT cyberattacks.
- Design and propose a framework detecting of multiple IoT cyberattacks.
- Evaluation of the proposed framework.

1.5 Methods

Throughout the course of this research, DSR methods will be applied which consist of the following procedures [23] and process model shown in Figure 1-1. The reason behind choosing DSR method is because it focuses on solving real-world problems through creation of innovative artifacts (the framework in this case). The DSR methodology allows for a more structured approach to both designing and evaluating the artifact which is as important as experimental analysis that provides a solid framework for the iterative process of designing and refining the artifact through continuous testing and evaluation, which is essential to my research.

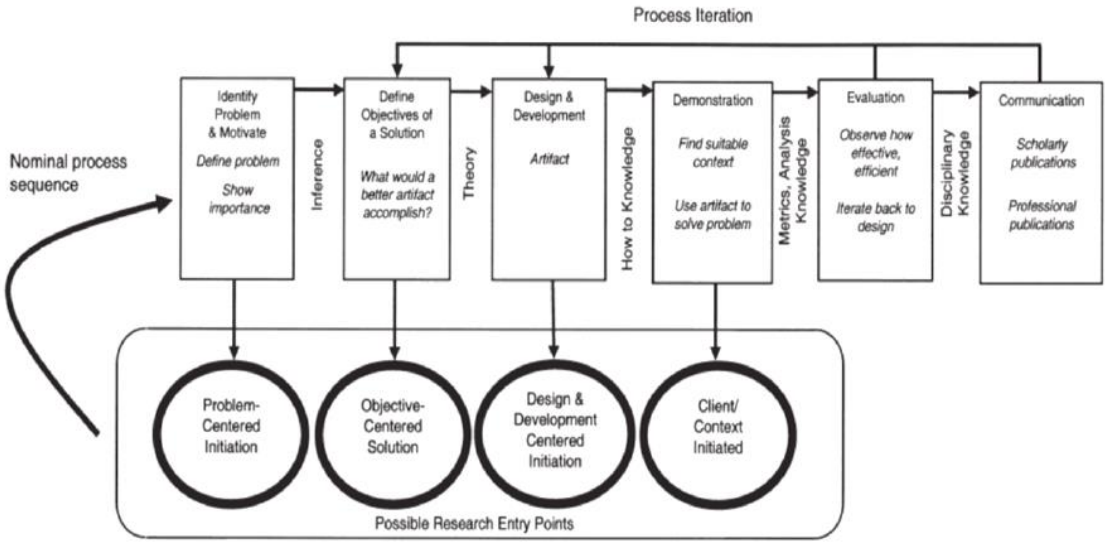

Figure 1-1: DSR Process Model [23]

A) Problem Identification and Motivations

To clearly identify the problem being treated and provide explanation elicitation and assessments of the gaps between existing research efforts will also take place.

B) Solution Goals

Studies will be conducted with the goal of designing and modeling a framework for detecting multiple cyberattacks in the IoT environment.

C) Design and Prototype Development

The framework must be planned and constructed by using a design science methodology. With the help of prototype production, feedback will be collected, as well as the revision of the framework to ensure the objective is met. To properly detect cyberattacks, we use deep learning algorithms because deep learning algorithms efficiently handle identifying and learning complex patterns in high-dimensional datasets, and provide state-of-the art performances, best for automatic feature extraction, and has proven success in various industries in deep detection of attacks. Additionally, Rabbit MQ, MQTT, as well as VScode development framework in combination with python and MLib will be used because these tools are useful for machine-learning projects and have a wide range of libraries that enable flexibility in data loading, preprocessing, and visualization.

D) Demonstration and Evaluation

The proposed framework feasibleness will be demonstrated and evaluated by emulating Docker Composer simulator. Also, the assessing of the model's execution and influence can be done using several proper ways such as testing, and simulation.

- Dataset: Key inquiries include the most recent IoT cybersecurity dataset, which is called CICIoT2023 used for our studies to identify the most prominent features, multiple deep learning algorithms that had been pretrained were applied to adapt to the ever-changing IoT threat landscape gadgets. The primary key reasons of selecting CICIoT2023 dataset is that compared to other IoT datasets such as (Edge-IIoTset, WUSTL-IIoT, X-IIOTID, MQTT-IoT-IDS, IoTIDS, N-BaIoT, Kitsune, BoT-IoT, and IoT-23) CICIoT2023 dataset is very novel, enrich and extensive for large scale IoT attacks security analytics which supports 100+ IoT related devices in real IoT operations [24].
- Testing: Proper test of the framework should be performed using the acquired dataset.
- Experiments: Prove out and execute the performance experiments that prove the effect of the system on the different elements of input and the environmental factors.

- Model findings from the datasets will be validated and graphically represented based on the model used in the analysis.

E) Communication

The outcomes of the study should be communicated to the appropriate parties. In addition, the research findings articulate their contributions to the various journals by creating a discussion area across the practical implications of the framework.

1.6 Scope and Limitations

The aim of this study is to develop and demonstrate detection of a multiple cyberattack in an IoT environment. The dataset used in this investigation is based on and only used CICIoT2023 dataset as the dataset is primary includes bunch of attack types as well as live traffic captured data will be analyzed in reference to CICIoT2023 dataset. Furthermore, the research study is limited to eighteen classes of attack class information mentioned in Annex A (i.e., not cover all attack class due to high performance computing resource and computational cost as well as limited time constraint).

1.7 Application of Results

It is anticipated that this research will be able to manifest a significant technological leap in the detection of numerous cyberattacks targeting IoT applications and services. The implied methodology and approach are most likely the ones that will be expanded to a wide range of applications. This will lead to far-reaching impacts on educators and researchers, consequently putting further research that examines the safety of IoT. Additionally, the research will lead to enhanced security and safety of IoT devices, software, and services, thus defending devices from being hacked and IoT services developing into a more reliable and secure form.

1.8 Organization of the Rest of the Thesis

The remainder of the dissertation comprises five main chapters. This section organizes the chapters in more detail therefore, the reader can know the main points of each.

Chapter Two covers the areas of IoT technologies, IoT components, benefits, and IoT models that have been implemented in several sectors. These are backed by examples such as IoT application scenarios and IoT cyberattack types that IoT covers. It also discusses

the most innovative approaches for cybersecurity that have been introduced in Chapter One.

In Chapter Three, related works that have significant relations with the thesis but other related to the thesis are taken into consideration. Also, besides that, the various IoT-based attack types occurring in the layers of IoT will be disclosed.

Chapter Four will discuss the proposed multiple cyberattacks detection mechanism. Apart from that, the parts discussing the relationships between components and their high-level descriptions will be presented.

Chapter Five introduces the themes which are related to demonstrating how the model is implemented. The tools used to undergo a prototype of development are also specified, in relation to their importance as per the prototype development. Furthermore, in this Chapter prototype results will be presented and discussed in detail.

Chapter Six will provide recommendations and future work as suggested improvements and better proposals which could be improved and used for future research.

Chapter 2 Literature Review

This Chapter contains explanations and discussion of digital and IoT technologies. Taxonomy of IoT, components, benefits, challenges, general IoT frameworks, IoT implementation model across multiple industries, IoT application scenarios, and IoT cyberattacks and detection types, and machine learning techniques will be discussed briefly.

2.1 Digitalization and IoT Technology

The primary initial steps towards a smarter system with higher process quality and more efficient energy consumption efficiency in this age of IoTization era in particular, Industry 4.0, a technological driven approach, and Industry 5.0, the human-centric approach and initiatives is to give services and industries with IoT competence means. This is important because it grants consumers to execute their duties at the same time being responsible for the harm they cause to the environment [25].

Industry-wide digital transformation has been largely driven by a combination of computing from the cloud, the production of data virtually connected with IoT, and the rollout of 5G wireless cellular communication anywhere worldwide. This new generation of networks places an emphasis on machine-type communication, allowing for communication with historically low latency and unprecedented reliability. It also offers extremely wide IoT connectivity, which paves the way for an enormous number of new uses across many different verticals, including transportation, healthcare, agriculture, energy, and manufacturing [26].

2.2 Taxonomy of IoT

Articulating the various and multidimensional dimensions and its concomitant results of IoT, and how they affect them in a broad spectrum of areas and technological landscapes, are the most suitable taxonomies and ecosystems around IoT. An IoT taxonomy ecosystem is also a kind of standardization which allows the necessary components and infrastructure to be employed in a harmonious direction across technology suppliers. Due to the heterogeneity and complexity of IoT ecosystem there are no agreed standards for IoT taxonomy at all [27].

A plethora of domain classifications have been generated by the diversity and complexities of the IoT environment. Classifications like these are commonly referred to as taxonomies. There are various approaches to categorizing IoT taxonomies. One approach [28] is to follow a descriptive framework that involves factors such as protocols, structure, power use, scalability, security, social network, and interoperability. Another approach [29] to use taxonomies that are specific to communication technologies, operating systems, gateway operating modes, architecture, middleware, platforms, storage techniques, capability and performance, application life cycle, and entity and service life cycle.

Furthermore, mentioning applications, platforms, middleware, devices on the Internet of Things (IoT) as well as operating systems, communication interfaces, and networks, were all a part of the innovative taxonomy that was developed in [30].

2.3 Elements, Components, and Characteristics of IoT

The elements of IoT [6] grouped into three categories: first one technology that enable "things" to acquire contextual information, secondly, technologies that enable "things" to process contextual information, and the third technologies to improve security and privacy. The first two categories can be jointly understood as functional building blocks required building "intelligence" into "things", which are indeed the features that differentiate the IoT from the usual Internet. The third category is not a function but rather a de facto requirement, without which the penetration of the IoT would be severely reduced.

The main components of IoT include identification of devices which basically give addressing and naming of those devices using different methods such as product number, and instructions, sensing which do actions via actuators, DFID tags etc., processing component performs the computation, communication used as a bridge on orchestration of device and sensors communications, and finally semantics where it decides on data what to do with it, and then responds to the devices. Below is a visual representation of IoT components [31] .

On the other hand, the research study [32] mentioned the four categories of Internet of Things system components includes things, communication links, gateways, and storage that symbolize several areas of technological advancement.

Some of the common essential characteristics of IoT are [3, 6] interconnectivity means everything's linked to network of networks, heterogeneity where IoT environments are diversified platforms , dynamic means IoT services are changes based on context such as speed, locations, changes in device states etc., other characteristics includes economic scale whereby the IoT technologies tremendously growing and consume starting from individuals to large organization, and finally IoT needs a central management because of massive data processing nature.

2.4 Internet of Things Advantages

Businesses and consumers alike stand to gain from the many efficiency and security enhancements promised by IoT. Enterprise services afforded by IoT were unbelievably valuable to sectors including education, manufacturing, agriculture, health, and industrial infrastructure systems.

Every industry is embracing linked things to keep up with the globe nowadays. Students with hearing loss or other disabilities can now access the curriculum using connected gloves and tablets, expanding the reach of education beyond the typical classroom setting. For other students with disabilities, the Internet of Things can also prove to be an immense help. In today's fast-paced world, smart homes and cities are becoming more commonplace to satisfy people's basic technological requirements including safety, waste management, better air quality, and entertainment [33].

Whether it's wearable, telemedicine, or remote patient monitoring, IoT has brought a plethora of benefits medical service industry, from fitness trackers and health monitors to network-enabled medical devices that will transform and revolutionize the healthcare systems [34].

The Internet of Things has introduced new approaches making water management and soil monitoring more effective in the farming sector [35].

Factories can increase production while decreasing unnecessary expenditure on energy and equipment repairs with the help of IoT. The advent of IoT has also improved energy management in electric grids to unprecedented heights [36]. The generation of actionable data

is yet another benefit of IoT. This data has the potential to improve management, quality control, efficiency, and productivity across all sectors.

Home automation systems and internet-enabled appliance management platforms are contributing to the growing popularity of "smart" services, which encompass a wide range of concepts [37], including "smart cities" and "smart buildings," among consumers. This has gone a long way and is now an essential part of people's daily lives, from internet connected home appliances to smart cities and beyond.

Connected vehicles [38], made possible by IoT, have the potential to revolutionize smart vehicles by making public services like emergency response, infrastructure improvement, road traffic control, reduced energy consumption, and a myriad of other cross-cutting issues easier to implement in a more cost-effective way.
Internet of Things (IoT) devices will boost situational awareness, which in turn will save lives, boost operational efficiency, which in turn will save costs, lessen the need for human interventions, and make predictive analytics possible, which will allow for the identification of future public safety crises.

2.5 IoT Security Trends

Implementing IoT is not without its difficulties, there are various factors to IoT security flaws both technical and non-technical, which can be broadly classified into four categories which are technology, system interoperability, dependability, and the market trend [39, 40] largely affects the IoT security trend due to the fact that IoT now adays become an integral part of the communities.

After the first major attack on IoT occurred in 2016 [41], attention is now turning to ensuring the gap on security of the IoT devices. Lack of a unified and non-standardized security protocol and the heterogeneity of devices that make up IoT network makes it impractical to implement a robust security mechanism.

Given the multi-faceted nature of IoT such as size, power consumption, and cost, resource limits pose serious obstacles for IoT devices. The heterogeneity of IoT devices, which is a

common occurrence in distributed environments where most IoT environments use sensors, actuators, and other devices, is another significant issue [42].

Several mission-critical Internet of Things devices take part in the data flow, making the connectivity problem even more worrying. There is a significant technological hurdle to ensuring consistent data flow for diverse Internet of Things devices. When it comes to IoT, customer confidence is paramount, making security a top priority. Due to factors such as size, power, and cost, security management is given less attention in the IoT. Security breaches can compromise the integrity of IoT, leading to significant monetary and public relations setbacks.

2.6 IoT Application Scenario's

IoT applications are quickly permeating every sector, where many sides of human existence are being revolutionized by the advent of IoT, an extensive collection of things that are interconnected to collect and share data. There is a vast variety of sectors and fields where the Internet of Things could be applied, each with its own set of problems and possibilities. Smart cities and urban infrastructure management, IIoT, smart agriculture and precision farming, environmental monitoring, retail and customer experience enhancement, transportation and logistics management, energy and smart grid management, wearable computing, connected healthcare, and personal health tracking are some of the key application scenario areas for IoT environments [43-45]. Figure 2-1 displays a few of the many IoT applications.

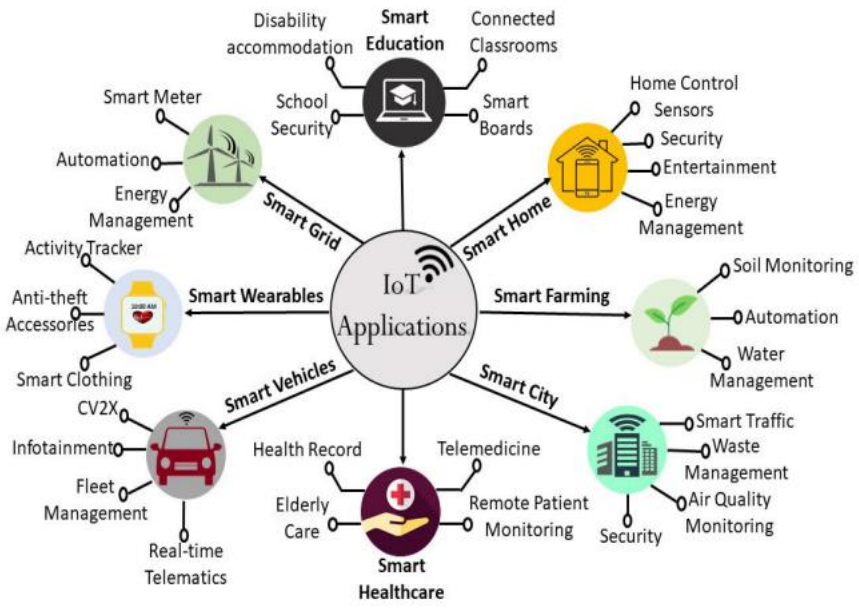

Figure 2-1: IoT application Representation.

Some of the IoT applications areas are described as follows:

- **Smart Home:** A "smart home" [43] is a home or building that can automatically react to events happening in its immediate environment. There are two main components to a smart home system. The first is the hardware and sensors, and the second part is the application and user processing hardware interface. Users can access and control their connected devices from anywhere, allowing them to track temperature, lighting, door opening, climate, and security systems. This allows them to set up real-time alarms and change settings as needed regardless of time and locations [44].
- **Smart Agriculture:** Smart agriculture technology [45, 46] built on IoT technologies, can benefit various real-time agricultural operations, including irrigation, plant protection, fertilization, disease prediction, and product quality. By combining precision agriculture with innovative technology, farmers can watch their plants remotely, boost harvest yields, and make more accurate decisions. By facilitating the integration of technological solutions IoT has transformed agriculture, bringing together productivity, pleasant production, and better quantity [47].

- **Smart Cities:** Smart cities consist of various components such as houses, healthcare, transportation, parking, environment, climate, waste management and people through the integration of AI [48]. The primary aim of a smart city is to collect data, manage information, and put decisions into action. Low running expenses and comfort are the results of smart city components that provide security, comfort, and energy savings and allow homeowners to control their assets remotely [49].
- **Smart Health:** IoT technologies [50] have the potential to enhance the quality of life, manage chronic diseases, and implement life-saving interventions through smart health applications made possible by IoT. IoT technology is revolutionizing healthcare by providing such as health monitoring, smart intelligent medical care and efficient management of hospital personnel and equipment, thereby improving health services and reducing risks [51]. Additionally, the study [52] presents how IoT could help with identifying COVID-19 epidemic, as well as about cloud-integrated systems, wearable tech, blockchain technology, cognitive radio (CR)-based IoT, and healthcare delivery.
- **Industrial IoT:** Industrial IoT allows businesses to make quick decisions through smart manufacturing and supply chain by integrating digital and physical technology into building flexible, networked operations. Industrial Internet of Things (IIoT) solutions boost production efficiency and product quality by using edge and networked sensors. Its primary goal is to increase profitability while simultaneously improving reliability, competitiveness, productivity, energy optimization via wireless sensors, another way IIoT that support sustainability efforts [53].

2.7 Cyberattack Types in IoT Environments

Firmware vulnerabilities, unauthorized access, insecure data transfer, and DoS attacks are just a few security dangers that IoT is creating. Security breaches, unauthorized changes to device settings, and interruptions to critical infrastructure are all outcomes of these threats. Breach of healthcare data, interruptions to transportation systems, attacks on industrial control systems, intrusions into private spaces, and misuse of personal information are all examples of real-life security hazards. To steal data or alter device behavior, hackers can use vulnerabilities in IoT apps to trick traditional approaches like behavioral analysis or user profiles. Attacks on

application logic and protocol-based attacks are examples of high-level risks to the Internet of Things [54].

Supply chain assaults insert harmful malware into IoT devices during manufacturing or updating, allowing for data theft or control. Machine-learning-based attacks exploit specific weaknesses in IoT networks or devices without detection. Attackers can launch targeted attacks by training machine learning models to detect exploitable vulnerabilities.

Some of the key attacks in IoT environments [55, 56, 57] includes.

- **Cross-Site Scripting:** XSS is an attack that inserts harmful code into legitimate websites or apps, allowing hackers to steal sensitive information, take over devices, and lead users to phishing websites.
- **Injection Attacks:** An adversary can launch an injection attack from a variety of attack spectrums by feeding an application code or data of distinct kinds, which can then produce unanticipated effects [54, 58]. The study presented by [59] proposed SQLIA analysis such as SQLI vulnerability testing and detection using a defensive secret writing in internet application code. Another attacks, SQL injection attacks scenarios including blind injection, union queries, piggy-backed queries, and logically flawed queries. By using conditional expressions, aggregate functions, or group by clauses, the attackers hope to circumvent authentication and extract data.
- **Denial-of-Service:** There are several security risks in the IoT [60] that attackers are taking advantage of, and they do this by exploiting flaws in devices. IoT devices are ideal targets for DDoS attacks because they can overwhelm targets with many devices. The detection and removal of compromised devices is becoming more critical since IoT connected devices continues growing.
- **Remote Code Execution:** Is an exploit [61] allows malicious actors to remotely execute malicious code regardless of the developer's choice of language. A "remote" attacker is one who can launch attacks from a location other than the machine running the program. This code could compromise data, launch ransomware, access servers or software, open backdoors, and much more besides [62]. Furthermore, there must be greater focus on preventing software-based attacks that exploit various vulnerabilities in IoT services [63, 64].

- **XML External Entity:** XXE injection is one form of cybersecurity threat that use XML files during data transit and storage is known as XML injection attacks. In these types of assaults, the perpetrator changes the sent or stored XML data by modifying the document's elements, attributes, or entities; the victim's application may then be executed [61]. Attackers can manipulate XML data processing by exploiting XEE [65], which allows attackers to load values from URLs or files.
- **Remote File Inclusion:** An example of a code injection attack that gives a chance to access files and directories beyond what a web application intended is RFI. Attackers in RFI can take advantage of a security hole to install and run remote files on a victim's device. The exploit allows the bad actors to gain control of the victim's device by executing malicious malware hosted on their remote server [66].

2.8 IoT Attack and Detection Techniques

IoT environment has three foundational layers namely physical, network, and application layer. On top of the foundational various authors [67] proposed multiple layers for protection of IoT environments including the data/computing layer, Business Layer, and cloud layer and a seven-layer based architecture is proposed by author shown in below Figure [68, 69].

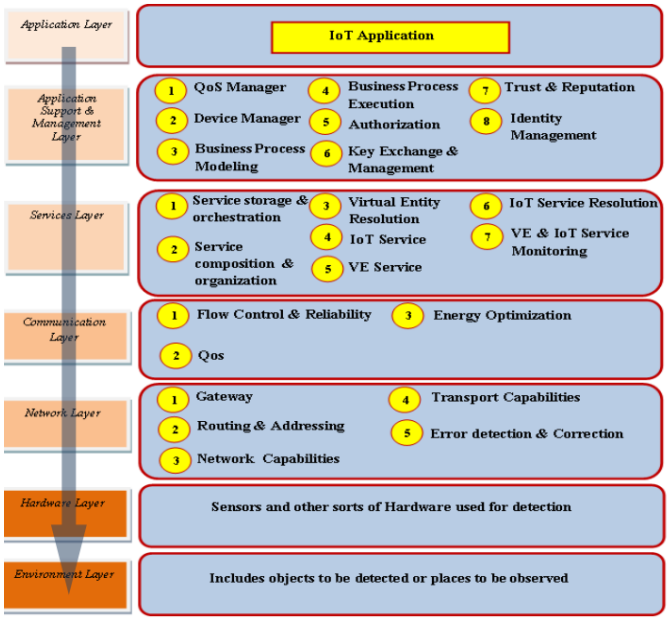

Figure 2-2: Seven-Layer Based Framework [84,85]

2.8.1 Three-Layer Based Attacks

It is a remarkably simple architecture that satisfies the basic three layers namely physical, network and application layers [32] discussed as below.

- **Physical Layer**

The study [32] discusses the link spoofing security issue in SDIoT controllers. It conducts vulnerability analysis and shows link-spoofing attacks. The authors propose a hybrid countermeasure, which combines initiative-taking and reactive methods to find and stop attacks. Even though the study's strength lies and provides remarkable detection rates, it only focusses on single spoofing attacks and ignores other attack types.

The authors [68] discuss the issue of jamming detection within the broader perspective of communication channel use. However, the data exchange challenges brought on by frequency jamming are the fundamental issues that need a technique to protect the communication channels, such as frequency hopping spread spectrum technology, a technique that spreads radio signals by varying switching among numerous frequency channels.

The study proposes [69] three protocols named, BASE, DeClone, and DeClone+, for clone detection in large anonymous RFID systems. Based on the result findings DeClone and DeClone+ are faster at detecting large systems than BASE and recognize cloning attacks more quickly with more cloned IDs. However, it does not compare the proposed protocols to existing ones and only considers anonymous RFID systems.

The research study explores alternative protocols and detection of sensor node capture attacks focusing on methods and approaches used to secure WSNs, especially with mobile nodes. It highlights that asymmetric key cryptosystems are insufficient for protecting WSNs due to their high computing power and the need for two key pre-distribution schemes for safe operation across sensor nodes [70].

The author [71] proposes a secure cross-device networking protocol to link IoT devices using Energy Distributed Control Centers (EDCCs) techniques. The paper also discusses potential attacks on IoT architectural layers and provides defenses against them. However, there is a

lack of practical evidence or case studies for the proposed solutions and security architecture, including the protocol's scalability and interoperability are not described.

The study [72] shows that timing attacks, particularly against OpenSSL, can compromise other software systems. The researchers measured the response time of an OpenSSL-based web server to decryption requests to prove how attackers can steal private keys. The study found that timing attacks can successfully extract the RSA private key from an OpenSSL-based web server running on a local network.

- **Network Layer Attack**

Attacks by passing network traffic such as MiTM attacks, Sybil attacks in wireless network [73] using K-means algorithms. The proposed method gives a detection accuracy of 95.13%. However, further improvements are needed to compare detection accuracy with other intrusion detection methods.

- **Application Layer Attack**

Attacks such as DDoS/DoS attack [10, 11], Botnet attacks [12, 14] , personal data theft [14, 15], Firmware exploitation (command and control , insecure code [19]), injection attacks [69], attacks XSS attack [74], and protocol-based attack [94] are some of the most common application-layer specific attack types in IoT environment. To secure IoT security, those articles can help to build on the concepts by detecting two or more multiple attacks.

2.8.2 Layer-Four Based Attacks

The researchers [75] have developed enhanced seven-layer based architecture finding or monitoring IoT devices items and environmental conditions. The study [76] explores the impact of IoT on insider threats in businesses and the challenges of detecting insider threats. The aim is to further investigate and evaluate the effectiveness of IoT insider threat detection approaches, develop detection systems and security mechanisms by expanding the taxonomy of attack vectors, and explore insider attack detection and privacy protection. We used this article as a reference for IoT insider threat detection systems by expanding attack vector taxonomy and exploring insider attack detection.

2.9 Machine Learning Based Detection

A significant technological hurdle for the IoT sector is the detection of such malevolent activities. This is why we need and use smart detection approaches from the field of machine learning (ML) to find potentially malicious attack patterns. Within ML, you can find anomaly detection methods that can automatically sort suspicious data from IoT networks into various categories [77].

2.9.1 Machine Learning Approaches and Methods

Below is the common machine learning approach used in attack detection.

- **Supervised learning:** In this method, data is labelled in a certain way, such as when a model is fed many samples of files to decide whether they contain malware. The model might choose more data based on this data labeling [78].
- **Unsupervised learning:** This approach makes use of unlabeled data, and the model marks them on its own depending on the characteristics of the data. It is often responsible for finding abnormalities in the data collection, and it is regarded to be the more powerful machine learning [79] and in addition a semi-supervised learning that combines both supervised and unsupervised method [80].
- **Reinforcement Learning:** This is used in an environment that is constantly changing [81] where it follows the environment-driven approach as well. A sub class of reinforcement learning called active learning functions similarly to that of a teacher, in that it may assist in the correction of errors and behavior in response to changes in the environment [82].
- **Deep learning:** Machine learning may be defined as a method involving many layers to be goes through to achieve high-level study features from a training set. That is, at every layer, we have neurons as well as an activation function for non-linear output. Deep learning is a specific type of feedforward neural network that can convert the features extracted to models with the utmost accuracy, with the output of the final layer becoming the input of the subsequent layer [83].

For IoT detections existing literature utilizes many different methods including Support-vector, which is most extensively used and recognized by several researchers. Data items sketched by the point in an n-dimensional region in support vector machines where n denotes the characteristics that are to be examined [84]. It does this by generating a hyper plane and dividing the data into several types [85].

Decision Tree is based on a gradient a group of decision trees is what makes up GBDT [86]. Random Forest: RF [87] follows the principles of bagging and random subspace, and it employs CART DTs as its foundational method. It excels in both classification and regression at the same time [88]. Another ANN deep learning method such as FFNN model uses a hidden layer to decide the number of neurons in the input and output layers. The hidden layer uses trial and error to figure out the number of neurons, while the input and output layers use the number of variables to determine the number of neurons [89]. Furthermore, RNN models such as LSTM collect data from multiple layers, unlike FFNNs. LSTMs use hidden unit connections to generate output and weight models using the approach called reinforcement neural networks (RNNs) due to their ability to handle gradient vanishing and exploding issues [89].

2.10 Chapter Summary

In this chapter we have discussed literature on the IoT environment, elements, characteristics, IoT taxonomy, and all advantages and IoT security environment are presented.

Several types of IoT applications, their real-world scenarios and their aims are also discussed. Furthermore, concepts related to IoT cyberattack types, machine learning approaches, and methods with detailed examples are discussed.

Chapter 3 Related Work

This Chapter presents the illustrations and discussion of existing research studies of IoT cyberattack detection techniques. Additionally, the research methods, addressed problems, limitations, and the gaps seen on the related work are discussed.

3.1 Framework Based on Machine Learning Detection

3.1.1 Support-Vector Machine

The study [90] provides security architecture for IoT devices using an ensemble method, distributed JRip and Support-Vector Machine algorithms. The evaluation was conducted using the NSL-KDD dataset with an accurate value of 99.71%. The proposed framework used Open Network Operating SDN controller, and Open-Source MANO Orchestrator. The main gap of the study of the proposed framework doesn't address detecting and response a novel cyberattacks for unknown signature patterns, the no seamless interaction between proposed IoT security modules. We use this research study as a reference by incorporating seamless interconnected modules and unknown attack patterns.

3.1.2 Decision Tree

A research study conducted by [91] presents a machine learning and stateful SDN architecture for identifying and preventing DDoS attacks in IoT networks using a decision tree technique. The FMDADM framework consists of four main components and a five-tiered structure.

To evaluate the effectiveness of FMDADM, various metrics were used, including accuracy, precision, recall, specificity, negative predictive value, false positive rate, false detection rate, false negative rate, average detection time, and F-measure. The suggested framework had a 99.79% accuracy rate, 99.09% precision, an F-measure of 99.43%, and a recall of 99.77% in the results. This was because it was better at detecting attacks and put more emphasis on choosing the right features which gives the solution is scalable and adaptable to different IoT network environments, and it achieves high performance values.

Despite promising performance results the proposed study remains limitation primary; it is limited cyberattack types and it does not include a diverse attack pattern, attack vectors, and

network conditions in different IoT environments. In addition, the proposed framework does not address a multi-controller environment in diverse IoT deployment scenarios.

3.1.3 Deep Deterministic Policy Gradient

The authors [92] proposed an intrusion detection technique that is based on Deep Deterministic Policy Gradient based algorithm and is designed to strengthen the security landscape of green Internet Things by using the traffic flow characteristics that are inherent to a green Internet of Things environment, the anomaly-based intrusion detection system homes in on distributed denial of service threats. The assessment framework for this technique included the CICDODoS2019 dataset, which was divided into two sub-datasets. These sub-datasets shed light on unique DDoS attack vectors that target TCP/UDP-based such as portmap, NetBIOS, and LDAP. An accurate rate of 99.14% was shown by the system in its ability to forecast network traffic patterns and find intrusions, which is evidenced by preliminary assessments that support the system's capabilities.

The research study method is not fully capture and manage complex traffic patterns, existing model struggling with high accuracy, and there is no testing based on varieties of unpredictable factors such as network congestion, diverse user behaviors, and different IoT devices. By addressing the gaps, we use the framework in distributed IoT environments for detecting multiple IoT cyberattacks.

3.1.4 Hybrid Machine Learning

An attack detection in IoT using machine learning proposed by the authors [93] in the application different domains such as object detections, text, image, and pattern recognition to employ and notify an early detection of cyberattacks. The paper proposed a classical machine learning techniques used Gradient Boosted decision tree, support vector, and random forest classifications to detect the heavy DDoS attacks with an enormous Botnet such as Mirai and the NSL-KDD data set. Based on the research the result showed that compared to the three algorithms used the RF algorithm provided the best accuracy of 85.34% on the fog layer.

The study has several primary gaps; the dataset used is outdated in detection multiple IoT cyberattack detections, also the proposed framework is limited only to fog layers and doesn’t include distributed components additionally, the proposed framework lacks identifying the

detection methods or techniques for identifying novel or emerging IoT attack types. Hence, we use this study to improve the framework's detection accuracy.

3.2 Framework Based on Deep Learning Detection

A growing number of recent studies are making use of deep learning models and methodologies [5] for the purpose of performing threat detection. Using the model and method that was used, we investigate the most recent research that has been done in this area.

- ### 3.2.1 Framework Based on Deep Neural Network

The study [19] suggests a distributed deep neural network framework (as shown on Figure 3-1) to find malware attacks on Internet of Things (IoT) devices. A multi-agent network of artificial intelligence models is employed in the proposed study to detect cyberattacks and harmful software in real-time. The study used both IoT-23 & Edge-IIoTset datasets to train and test models such as SVMs, DTs, GBs, NBs, and DNNs. Once the training data has been passed through several pre-processing steps, the models are evaluated using metrics including accuracy, and recall of 93% and 92% f1-score detection results respectively whereby the DNN model performs well among the trained model. The proposed framework, however, several gaps that required further analysis primary; the study primarily focuses on the detection of single cyberattacks and does not explore the detection of multiple cyberattacks simultaneously. Additionally, the dataset used in the research exhibits an imbalanced sample distribution and is limited in its feature set, which may lead to overfitting. Therefore, we can make use of the methods, and the approaches used as a reference to design a multiple cyberattack detection farmwork.

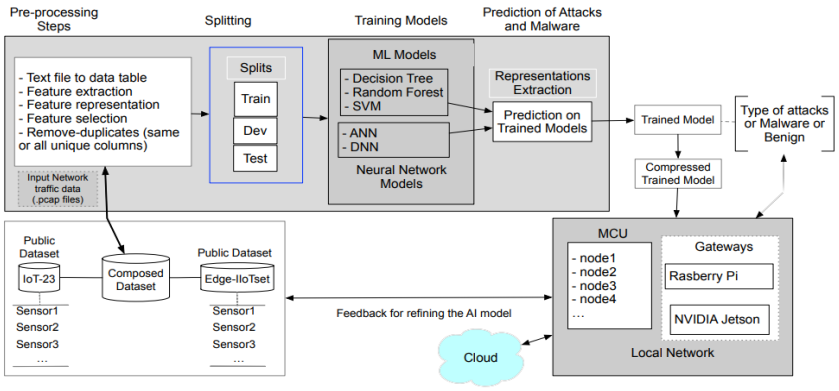

Figure 3-1: Proposed Framework by Author's **[19]**

An IoT intrusion detection framework based on a Dense Random Neural Network is proposed in study [22] in IoT networks. Researchers used the ToN-IoT security dataset and the Dense Random Neural Network techniques to evaluate the proposed framework accuracy value of 99.14%. Lightweight and adaptable security solutions for low-resource IoT networks are a strength of the proposed framework, which makes use of artificial intelligence and big data analytics. In addition, several performance evaluations were conducted, in which binary and multiclass scenarios were considered. As a limitation the study lacks on complex data handling on real-time attack scenarios, the dataset used not clear whether balanced or diverse enough to represent various attacks. Hence, we use it as a reference to improve diverse cyberattack scenarios by using a balanced dataset.

- ### 3.2.2 Framework Based on Convolutional Neural Network

This study [94] aims to make IoT services safer and stop data threats by suggesting a framework called Intelligent Intrusion Detection Framework for Multi-Clouds and IoT Environments. This framework uses swarm architecture and deep learning classifiers. Feature selection, categorization, and optimization are the three pillars of the proposed system. The primary methods used are 2D array-based convolutional neural network (2D-ACNN), RF, deep learning, and SGBoost whereby the framework evaluated using multiple datasets including NF-BoTIoT, NF-ToNIoT, F-UNSWNB15, and NF-CSE CI-CIDS2018 and gives a result of 95.20% accuracy. However, the framework doesn't detect attacks with different characteristics such as multiple attack types, and in case of feature selection techniques there are no detail comparisons of performance. As a result, we may use this article to investigate how the suggested architecture affects the accuracy of attack detections.

- ### 3.2.3 Framework Based on Hybrid Deep Learning

- **CNN and LSTM Hybrid Framework**

The authors [95] offer a method that utilizes multiple deep learning models with the objective of predicting damaging threats on IoT backbone networks using the open-source UNSW-NB15 and NSL-KDD99 datasets. The models used are autoencoder (CAE), MLP, DNN, and CNN whereby DNN provides 99.24% accurate value. The proposed framework's strength lies

in its use of many deep learning models, each of which may record unique aspects of network traffic and be compared to others to boost detection accuracy. The proposed framework, however, has a limitation including it does not provide a detail analysis for different multiple cyberattack scenarios across varied IoT networks, the dataset representation is not fully represented for emerging IoT cyberattacks, and finally the models does not address a real-time detection.

- **FFNN and LSTM Hybrid Framework**

The authors [89] proposed a distributed framework (which has four stage data treatment and processing, DL model training and testing, distributed framework development, and attack detection stage) based on deep learning to prevent attacks using a model LSTM shown in Figure 3-2 and FFNN shown in Figure 3-3 (the structure is identical to that shown in Figure 3-2 but distributed using an LSTM) using a combined dataset NSL-KDD and BoT-IoT which give a result of 99.95% detection accuracy. The research study limitations include inability to generalize diverse datasets, no specific techniques outlined for data augmentation, the study proposes investigating the causal relationship between IP addresses and attack types, particularly in DoS attacks but no detail explanation and exploration of attack patterns.

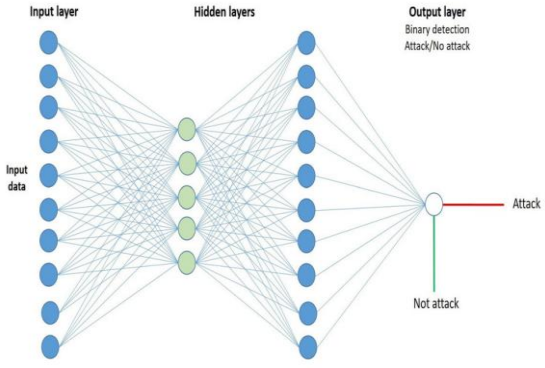

Figure 3-2: FFNN Attack Detection [89]

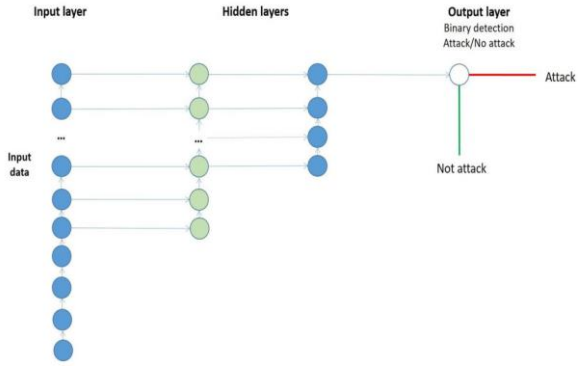

Figure 3-3: LSTM Attack Detection [89]

- **RNN and LSTM Hybrid Framework**

A deep blockchain framework was presented in a scientific publication [96] that combines a distributed intrusion detection system within a blockchain technology. To interpret sequential network data a bidirectional long short-term memory technique is used and evaluated based on UNSW-NB15 and BoT-IoT datasets whereby a DoS and DDoS cyberattacks using TCP

and UDP specific ports. According to the findings, the best size of concealed nodes is ten, which, when applied to the UNSW-NB15 dataset, results in an accuracy of 97.26%, while the BoT-IoT dataset yields accuracy of 96.71%. The proposed framework have is not scalable to detect multiple, especially when applied in large-scale IoT environments with numerous data points

In conclusion, the proposed framework that was carried out by many of the researchers, such as [91, 94, 97] adheres to and makes use of a similar kind of framework IoT attack and detection workflow, as depicted in Figure 3-4. This research study also uses the approaches as a reference.

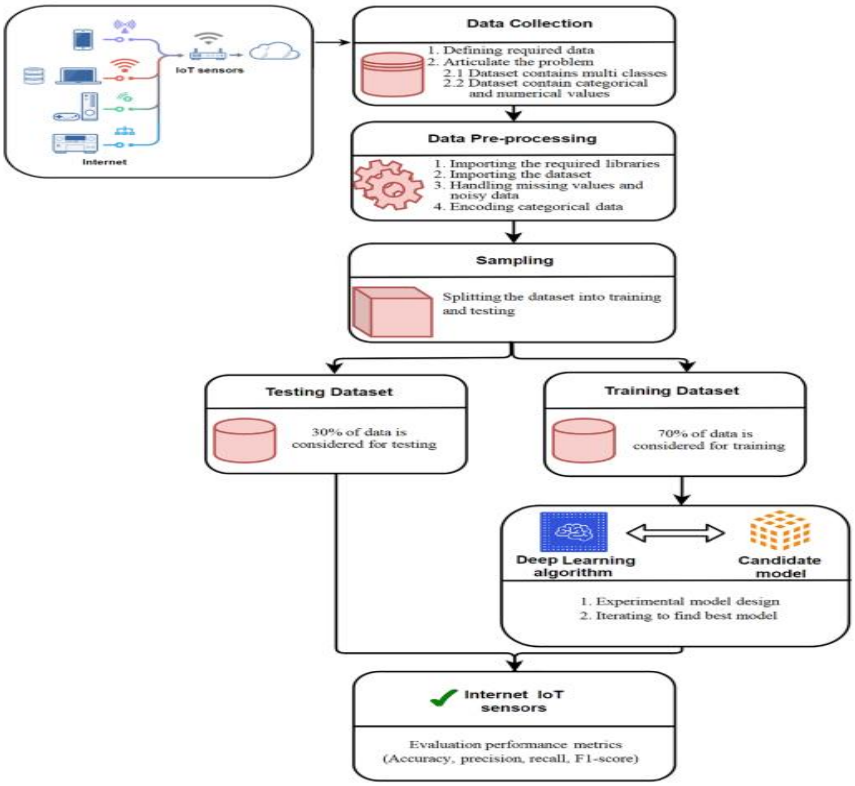

Figure 3-4: Framework for Network Anomaly detection in IoT Traffic **[97]**

Table 3-1: Related Work Summary

Papers	Proposed Solutions	Model & dataset	Metrics	Limitations
[19]	Presents a framework for an AI-powered system to detect malware threats.	DT, RF, NB, SVM, and DNN. IoT-23 dataset & Edge-IIoTset	93% Accuracy	- It does not explore the detection of multiple cyberattacks simultaneously, the dataset used is imbalanced sample distribution and is limited in its feature set, which may lead to overfitting.
[22]	Introduce a Dense Random Neural Network–Based Intrusion Detection System detection framework.	Dense Random Neural Network (DnRaNN) ToN IoT dataset	99.14% Accuracy	- The study lacks on complex data handling on real-time attack scenarios, the dataset used not clear whether balanced or diverse enough to represent various attacks.
[89]	Proposed a detection framework IoT Cyberattack in distributed network	LSTM & FFNN NSL-KDD and BoT-IoT	99.95% Accuracy	- Inability to generalize diverse datasets, no specific techniques outlined for data augmentation, the study proposes investigating the causal relationship between IP addresses and attack types, but no detail explanation and exploration of attack patterns.
[90]	Proposed SDN and NVF based frameworks to detect anomaly detection in IoT systems.	Support-Vector Machine NSL KDD dataset	99.71% Accuracy	- The proposed framework doesn't address detecting and response a novel cyberattacks for unknown signature patterns, no seamless interaction between proposed IoT security modules.
[91]	Bring out a framework for detecting and mitigating DDoS attacks on IoT networks that use many layers.	DT, BLR, SVM, and KNN algorithms Dataset Not Mentioned	99.79% Accuracy	- It used limited cyberattack types and it does not include a diverse attack pattern, attack vectors, and network conditions in different IoT environments. In addition, the proposed framework does not

				address a multi-controller environment in diverse IoT deployment scenarios.
[92]	Proposed an intrusion detection landscape for green Internet Things	Deterministic Policy Gradient CICDODoS2019 dataset	99.14% Accuracy	- The study method is not fully capture and manage complex traffic patterns, existing model struggling with high accuracy, and there is no testing based on varieties of unpredictable factors such as network congestion, diverse user behaviors, and different IoT devices
[93]	Proposed early detection of IoT Cyberattacks	NSL-KDD dataset.	85.34% Accuracy	- The dataset used is outdated in detection multiple IoT cyberattack detections, the proposed framework lacks identifying the detection methods or techniques for identifying novel or emerging IoT attack
[94]	Proposes IDF for Multi-Cloud and Internet of Things (IoT) Defense.	XGBoost, RF, CNN, and DL. NF-BoTIoT, NF-ToNIoT, F-UNSWNB15, and NF-CSE CICIDS2018	95.20% Accuracy	- The framework doesn't detect attacks with different characteristics such as multiple attack types, and in case of feature selection techniques there are no detail comparisons of performance.
[95]	Presents a system for finding harmful actions on the Internet of Things data center.	Autoencoder, CNNs, MLPs, and DNNs UNSW-NB15 and NSL-KDD99	98.96% Accuracy	- It does not provide detailed analysis for different multiple cyberattack scenarios across varied IoT networks, the dataset representation is not fully represented for emerging IoT cyberattacks, and the models do not address real-time detection.
[96]	Proposed a deep blockchain framework for Internet of Things (IoT) networks.	BLSTM UNSW-NB15 and BoT-IoT	97.26% & 96.71% Accuracy	- The proposed framework has is not scalable to detect multiple, especially when applied in large-scale IoT environments with numerous data points

3.3 Research Gaps

Below is the summary of identified gaps mentioned in existing research studies

- The framework proposed by [19, 91] does not explore the detection of multiple cyberattacks simultaneously, the dataset used is imbalanced sample distribution and is limited in its feature set, which may lead to overfitting. It does not include a diverse attack pattern, attack vectors, and network conditions in different IoT environments.
- The framework proposed by authors [90, 95] lack of detail analysis for different multiple cyberattack scenarios across varied IoT networks. The dataset representation is not fully represented for emerging IoT cyberattacks. It doesn't address detecting and response to novel cyberattacks for unknown signature patterns, no seamless interaction between proposed IoT security modules, and finally the models do not address a real-time detection.
- A study conducted by [22, 94], lacks on complex data handling on real-time attack scenarios, the dataset used not clear whether balanced or diverse enough to represent various attacks, doesn't detect attacks with different characteristics such as multiple attack types, and in case of feature selection techniques there are no detail comparisons of performance.
- In the research study [92], the method used is not fully capturing and managing complex traffic patterns, existing models struggling with high accuracy, and there is no testing based on varieties of unpredictable factors such as network congestion, diverse user behaviors, and different IoT devices.
- In the proposed framework [93], the dataset used is outdated in detection multiple IoT cyberattack detections, also the proposed framework is limited only to fog layers and doesn't include distributed components.
- Many of the existing proposed frameworks used are earlier generation of dataset (such as WUSTL-IIoT, Kitsune, CICIDS2017, BoT-IoT, NB-IoT, NS-KDD, UNSW-NB15, KDD99 and IoT-23) do not demonstrate IoT cyberattacks detection in the modern dynamic IoT environment setups.
- To the best of my knowledge, there is no research study that detects multi-faced IoT cyberattacks security detection framework in IoT arena [10, 11, 13].

Chapter 4 Detecting Multiple IoT Cyberattacks Framework (DMICAF)

In this chapter, we go over the methodology for detecting multiple cyberattacks in an IoT context. The two sections of the chapter are key design considerations and the suggested framework. In the second section, the proposed framework components and detailed descriptions of each component are discussed.

4.1 Design Considerations

The following are the design considerations that are incorporated into the proposed multiple cyberattacks detection framework.

- Multi-Layer Security: The attack detection framework integrates and monitor data, packet traffic, and behavioral patterns from multiple sources for early detection of emerging cyberattacks.
- Distributed Components: The proposed framework is designed based on distributed components to handle large volume of data generated by multiple IoT devices.
- Adaptive Responses: The framework supports adaptive, nearly real-time detection response mechanisms by triggering automatic detection alerts.

4.2 Proposed Framework

Numerous critical processes are engaged in designing an IoT cyberattacks detection framework. The proposed framework is presented in Figure 4-1.

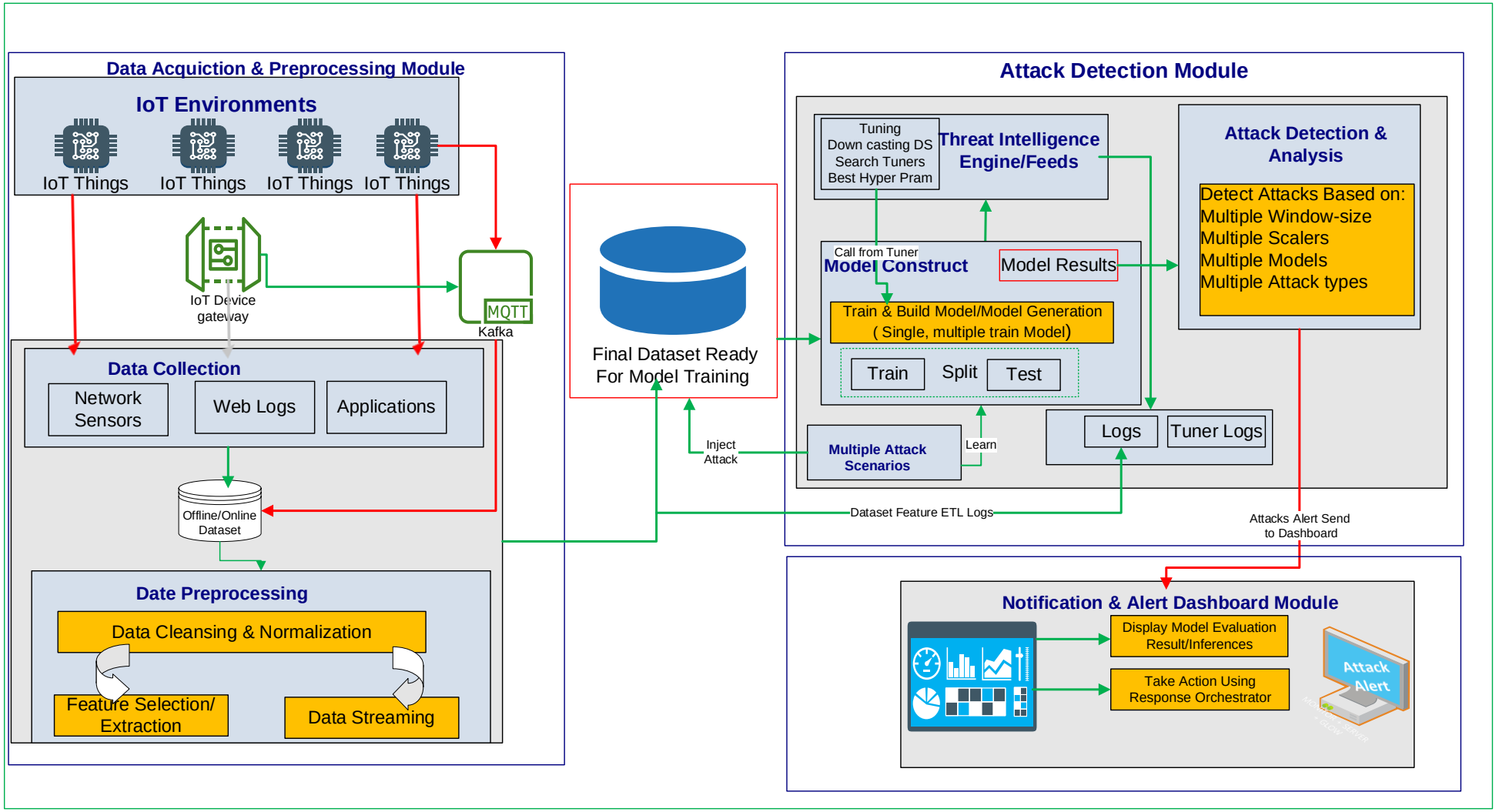

Figure 4-1: Multiple IoT Cyberattacks Detection Framework

4.2.1 Framework Components

Components of the proposed framework are discussed below.

4.2.1.1 Data Acquisition & Preprocessing Module

The responsibility of data acquisition and preprocessing module is to gather all necessary data from the IoT application environment via network sensors, logs, applications and even through API. Below is the detailed description of the sub-components.

- **IoT Environment:** This sub-component is used to indicate internet of things environments that feed and /or share data to IoT service in offline and/or real-time via IoT gateway.
- **Data Collection:** This sub-component collects data from various sources across the IoT environments such as from sensors, web logs and applications log to be stored into a database system for further detail action. Some of the data collected from IoT devices such as temperature sensors, motion detectors, environmental sensors, and security cameras included. Data is also collected from network packets or traffic logs that capture the flow of data between IoT devices, IoT gateways, and backend systems that help identify unusual network activity. To facilitate and collect the datasets from multiple IoT devices RabbitMQ and Paho MQTT python libraries are being used for real-time data collection and streaming, allowing for the ingestion of massive volumes of IoT data. Once the data is collected, we use the Cassandra database (since it is best fit for time-series data) for further analysis
- **Data Preprocessing:** Once the data capture and collect from IoT devices (packet data converted into raw dataset and the raw dataset converted into final versioned dataset) before the final version dataset the pre-processing stage performed various activities this includes cleaning removing irrelevant, nosy, or erroneous data points, handling of missing values and outliers. Then data transformation is normalizing, standardizing, and filling gaps; the primary focus is ensuring consistency data values including is used across different IoT environments. The tools used for data pre-processing are pandas handing missing data and data transformation, and Scikit-learn from imputation missing values using various imputations strategies such as (mean, media, most frequent), and machine learning pipelines for labeling and indexing. In addition, RabbitMQ is used as a message

broker for real-time data ingestion from IoT devices which helps to handle data streams efficiently by acting as an intermediary between data producers (IoT devices) and consumers (backend systems or storage solutions). The detailed data preprocessing diagram describes in below section:

- **Data Processing Model Design Diagram**

Different unbalanced datasets will be captured and produced via RabbitMQ messaging and MQTT protocol as shown in Figure 4-2 then the dataset for sure is unbalanced since number of records is recorded from different sources again dataset split into 80:10:10 percent rule for training, testing and validation respectively and data balancing can be done through the string indexer and encoding method will be done during the data normalization process.

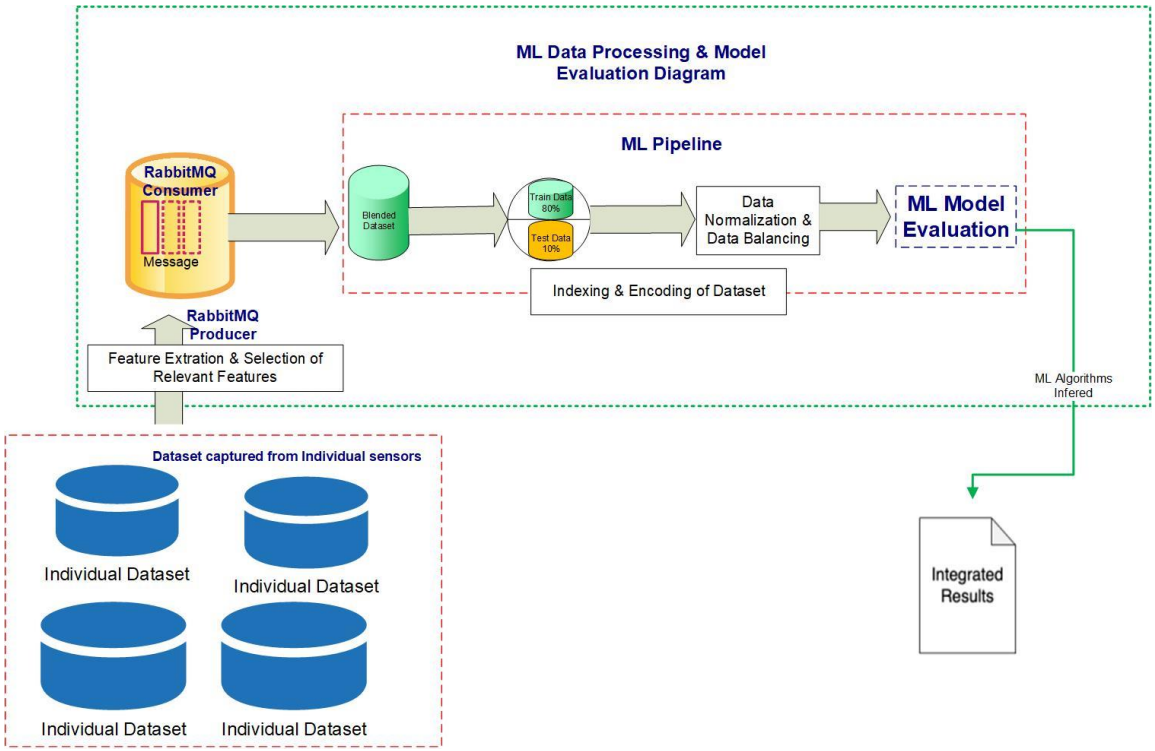

Figure 4-2: Data Pre-processing Pipeline Diagram

- **Feature Selection and Extraction:** an appropriate feature from the collected dataset will be selected for training characteristics and conducting framework evaluations to choose the right elements is crucial to the validity of the framework.
- **Data Streaming:** under the data preprocessing for live data streaming data cleaning and normalization as well as feature selection and/or extraction activities also performed accordingly.

4.2.1.2 Attack Detection Module

This module is the main engine of the framework that orchestrates the detection of cyberattacks threat in IoT environments. The below section presents detailed descriptions of the sub-components of the module.

- **Model Construct:** Training the framework with the features selected values is the core of the solution to construct the model. The proposed model construction uses a different model including CNN, FNN, LSTM, and Hybrid for the detection of multiple attack detection as per the defined algorithm 1-1. The primary reason for selection of the model are primary to compare the model performance comparison with the existing research studies mentioned in Literature Review Section, and secondly the selected models are efficiently handle and process large volume of data from IoT devices where it is well-suited for detecting cyberattacks in dynamic, time-sensitive environments, enabling real-time, robust, and adaptive detection of malicious activities across IoT environments.

Below is the framework pseudocode algorithm for multiple IoT cyberattack detection.

Pseudocode Algorithm-1: Multiple IoT Cyberattacks Detection

Input:

- DSnormal = {DS1, DS2, ..., DSn) // Set of IoT Normal Datasets
- Datai // Training data for each dataset Di
- Nadv // Adversarial/attack Count
- A // Attack Method
- D // Detection Mechanism

Output:

- M // Trained Detection Model

Procedures:

1. Sample Generation: // Generate adversarial samples
   For i = 1 to Nadv:
   x = Select a Normal IoT sample from DSnormal
   xadv = Apply adversarial attack A on x // xadv = A(x)
   Add (xadv, label(x)) to D_adv // Add to adversarial dataset
2. Model Training
   TrainedModel ← EmptyTrainedModel // Initialize the model
   For each IoT dataset Di in DS:

D_combined = DSnormal ∪ D_adv // Combine normal and adversarial datasets
FFNNModeli ← TrainFFNN (Datai) // Train FFNN for dataset Di
CNNModeli ← TrainCNN (Datai) // Train CNN for dataset Di
LSTMModeli ← TrainLSTM (Datai) // Train DLSTM for dataset Di
M ← TrainHybrid (CNN+LSTM) // Train Hybrid Model

3. Detection
   For each IoT dataset Di in DS:
   M_detection = ApplyDetection (M, Di) // Apply detection mechanism to the dataset
4. Model Evaluation: //Evaluate the performance of the detection model
   Evaluate the hybrid model M_detection using the evaluation dataset D_eval
   Compute metrics (Accuracy, Precision, Recall, F1-score)
5. Model Tunning:
   - Add the new adversarial samples to D_adv
     D_combined ← DSnormal ∪ D_adv // Retrain the model with the updated dataset
   - M_detection ← TrainModel(D_combined) // Retrain model
   - M_detection = ApplyDetection (M, D) // Apply detection to the updated model

- **Threat Intelligence Engine:** This module is the engine that performs tuning of the model construct and using different hyperparameters. During the tuning process the tuning logs are recorded for maintenance and troubleshooting purposes.
- **Detection and Analysis:** This module's primary responsibility detection is detection of various cyberattacks based on the different parameters used such as multiple window size, multiple scale type, multiple attack class. It is an engine that identifies abnormalities and classified as known and unknown attack that share and send the results to notification and alert dashboard.
- **Multiple Attack Scenarios:** The proposed framework used distributed IoT devices and components whereby any external attacker triggers an attack to the internal IoT devices and/or applications and/or services. In this scenario the attack module is any kind of IoT cyberattack that can be triggered from anywhere outside of the internal network and attacks happened based on actual attacks in simulated environments.

4.2.1.3 Notification & Alert Dashboard Module

This module shows the results that come from the detection module where it notifies alert based on the inputs. Below are the main components that provide further notification in real time.

- **Response Orchestrator:** A response orchestrator oversees reactions of those attacks and takes measures to provide insights into attacks and system health in a real-time notification.
- **Model Evaluation Results:** In the dashboard the values of the trained and test results with the details values are displayed against various cyberattacks based on the below key performance metrics.
    - Precision, recall, accuracy, and F1-score value will be measured and will identify malicious and benign traffic via the trained model.
    - An additional performance indicator including cross-validation, and confusion matrix are considered.

4.1.3. Attack Detection Framework Tunning and Optimization

The proposed IoT attack detection framework follows a step-by-step procedure starting with identifying IoT environments and attack types. Then a diverse IoT datasets, including normal and malicious traffic, are collected, cleaned, and standardized and ready for training. Selected machine learning algorithms, such as FFNN, LSTM, CNN, and hybrid models, are selected based on the needs of IoT attack detection. Traffic simulation generates both benign and attack-related data using MQTT and RabbitMQ. The models are then trained and optimized for performance through hyperparameter tuning and cross-validation. After training, the models classify traffic detecting attacks as benign or malicious. Performance metrics like accuracy, precision, recall, and F1-score are used for evaluation, comparing algorithms to identify the best model. The framework includes continuous model improvement, retraining with new attack patterns, and alerting mechanisms to notify administrators upon attack detection based on model performance tuning.

4.3 Model Generation

The model generation is a part of model constructing sub-components that triggers the four selected algorithms. Below is the detailed description of model generations.

4.3.1 CNN Model Generation

CNN detects IoT cyberattacks by treating IoT traffic as a 2D image-like structure, allowing CNN to learn spatial features from traffic patterns. The CNN framework, the input layer, reshapes the IoT traffic data into a 2D format, where features such as packet size are treated as image-like attributes. The CNN then uses multiple convolutional layers to extract these spatial features from the reshaped data. A max pooling layer follows, reducing the dimensionality of the feature maps. Afterward, a fully connected dense layer is used to perform the classification task. Finally, the output layer consists of a single neuron with a sigmoid activation function, designed for binary classification to determine whether an attack is present or not. Below is the CNN model generation.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, Flatten, Dense
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.metrics import AUC, Precision, Recall
from utils.logger import get_logger
from utils.config import LOG_DIR
from dl.preprocessor_utils import measure_duration
from dl.model_builder.custom_metrics import F1Score, WeightedF1Score
def build_cnn_model(hp, input_shape, num_classes, tuned_params=None):
  input_shape = (input_shape[1], 1)
  logger.info(f'Running CNN model builder with input_shape: {input_shape} and
num_class: {num_classes}')
  model = Sequential()
  # Define the input layer
  model.add(Input(shape=input_shape))
  # Map optimizer strings to their respective classes
    optimizer = {
      'adam': Adam(learning_rate=hp.Choice('adam_lr', [0.001, 0.0001, 0.01]),
clipnorm=1.0),
      }[optimizer_choice]
  # Add Conv1D layers with tunable hyperparameters
  model.add(Conv1D(
    filters=filters,
    kernel_size=kernel_size,
    padding='same',
    activation=conv1_activation
  ))
  # Add MaxPooling layer with tunable pool size
  model.add(MaxPooling1D(pool_size=pool_size))
  # Add more Conv1D layers for deeper networks
```

```
model.add(Conv1D(
   filters=filters_2,
   kernel_size=kernel_size_2,
   activation=conv2_activation,
   padding='same'
))
model.add(MaxPooling1D(pool_size=pool_size_2))
model.add(Flatten())
# Add Dense layers with tunable number of units
model.add(Dense(
   units=units,
   activation=dense_activation

))
# Output layer
model.add(Dense(num_classes, activation='softmax'))
# Compile the model with tunable optimizer and learning rate
model.compile(
   optimizer=optimizer,
   loss='sparse_categorical_crossentropy',
   metrics=[
      'accuracy',
      Precision(name='precision'),
      Recall(name='recall'), ]
)
# model.summary()
return model
```

4.3.2 Hybrid Model Generation

A hybrid model that combines CNN and LSTM networks is particularly effective for detecting IoT cyberattacks, as it can capture both spatial and temporal patterns in the data. The CNN layers are responsible for learning spatial features from the IoT traffic, while the LSTM layers focus on capturing the temporal dependencies within the sequence data. In this framework, the input layer consists of sequence data representing the time-series features of the IoT traffic. The CNN layers then extract spatial features from this data. Following the convolutional layers, the LSTM layer captures the temporal relationships, allowing the model to understand the progression of traffic patterns over time. A fully connected dense layer is used for the final classification, and the output layer contains a single neuron with a sigmoid activation function to perform binary classification, determining the presence of an attack. Detailed model generation is described in Annex-F section C.

4.3.3 FFNN Model Generation

The FFNN model for IoT attack detection is structured with several fully connected layers, designed to learn complex patterns from IoT traffic data. The model typically includes an input layer, multiple hidden layers, and an output layer. The input layer receives the features of the IoT traffic data, such as packet size, frequency, and source/destination IP addresses. The hidden layers apply nonlinear transformations to the data, enabling the model to learn intricate patterns. In this framework, the input layer receives the IoT traffic features, and the first hidden layer consists of a dense layer with 128 neurons and a ReLU activation function. The second hidden layer follows with 64 neurons, also using the ReLU activation function, and the third hidden layer consists of 32 neurons, again utilizing ReLU. The output layer contains a single neuron with a sigmoid activation function, performing binary classification to determine whether the traffic is normal or contains an attack. Detailed model generation is described in Annex-F section B.

4.3.4 LSTM Model Generation

The LSTM model is particularly well-suited for time-series data, where past information, such as previous traffic patterns, plays a crucial role in predicting future behavior, like detecting attacks. For IoT traffic, LSTMs are effective in capturing the temporal dependencies within the data, helping to identify patterns over time. In this model, the input layer consists of sequence data representing the time-series features of the IoT traffic. The LSTM layer, with 128 units, captures the temporal patterns and dependencies in the traffic data. A fully connected dense layer follows, used for the final classification task. The output layer contains a single neuron with a sigmoid activation function, performing binary classification to determine the presence of an attack. Detailed model generation is described in Annex-F section D.

4.4 Summary

In summary in this Chapter, we covered the detailed descriptions and flow of of proposed architecture, design consideration, architecture component and sub-component explanations, how the model algorithm operate, and the general generation procedures are explained.

Chapter 5 Implementation and Discussions

In this chapter, implementation of the proposed framework is presented. Methods and tools for programming and the modeling and manipulation of IoT attack detection environment elements are covered in this chapter. Finally, this chapter delves into the topic of experimentation and evaluation, which showcases the performance of DMICAF through a series of tests and comparisons.

5.1 Scope of Prototype

The primary scope of the prototype development is to fulfill the prerequisites of the attack detection framework mentioned in Figure 4-1. The cyberattacks detection are simulated and implemented within the scope of four attack categories: DDoS, DOS, Recon, and Mirai whereby and underneath with a sub attack class types with a total of eighteen attack class mentioned in Annex-C.

5.2 Tools for Development

In this section, the primary tools utilized are described in more detail.

5.2.1 Tools

In this research work for large-scale data preparation and model creation we used tools shown in Table 5-1.

Table 5-1: Lists of Tools

Tools	**Purpose**	**Tools**	**Purpose**
Pandas	For tabular, time series etc. data manipulation and analysis	Joblib	Provide lightweight pipelining
Scikit Learn	Provide supervised and unsupervised learning algorithms model evaluation and data preprocessing.	Multiprocessing	For the creation of processes that can run concurrently, leveraging multiple CPU cores.

Numpy	For large, multi-dimensional arrays and matrices	Tqdm	Monitor the progress of computationally intensive tasks.
TensorFlow	Development and deployment of machine learning models	Zarr	For working with large, compressed, N-dimensional arrays
Pytorch	For developing and deploying deep learning models.	Custom Logging	Allowing for the recording of events, errors, and other relevant information.
Conda	For managing Python packages and their dependencies	H5py	For efficient storage and manipulation of large data.

5.2.2 Rabit MQ & MQTT Protocol

To fulfill the main objectives of the study, it is necessary to utilize a specialized environment that can be able to store and process large volumes of IoT data that maintain high speed, user-friendliness, and reliable performance prediction. RabbitMQ & MQTT messaging protocols since both fulfill all the requirements of this research study, which is processing real-time streaming and key reason to use those platforms is because of an open-source platform that enriches in supporting easy for integrations, minimal data traffic usage, efficient data distribution, utilizes small amount of power particularly MQTT is designed for IoT devices and services.

5.2.3 Emulator Environment

In this research study, we chose a custom docker composer emulator environment for three main reasons initially for ease of simulation multiple IoT devices at a time, second is for resource optimization; and finally for simplicity of cloning and running the attack simulations in docker environment.

In the emulator environment all virtual image-based devices are created and ensured all the devices are up and running to simulate the complete cyberattacks detection. The

process begins with capturing packets from sensor networks then the captured data will be converted to raw data (human readable format) through packet processing module, which processes and generates the version dataset (a final dataset that contains different window size, and scaler types). Then data is consumed for pre-processing and analysis. Subsequently, the final version dataset passed to the model training phase, and the model tuning for re-defining after and before input into the model. Then finally, the model makes its predictions, the inference produces the results.

5.2.4 Development Topology and Environment

To implement the proposed framework, a dedicated and/or virtualized Ubuntu OS version 24.10+ and above is utilized. On top of the Ubuntu OS environment a docker engine; a platform that can manage diverse types of containerized nodes are used as described in Figure 5-2. The following hardware and software specifications are recommended for the development setup including 16 GB of RAM, with Core i7 octa core processor, 1TB SSD hardware resources and Docker Engine and docker composer for containerization, MQTT & Rabbit MQ (for Messaging and streaming), Casandra DB (for data storage), Python version 3.11 or higher, Fast API (for orchestration & alert), VScode (development Environment).

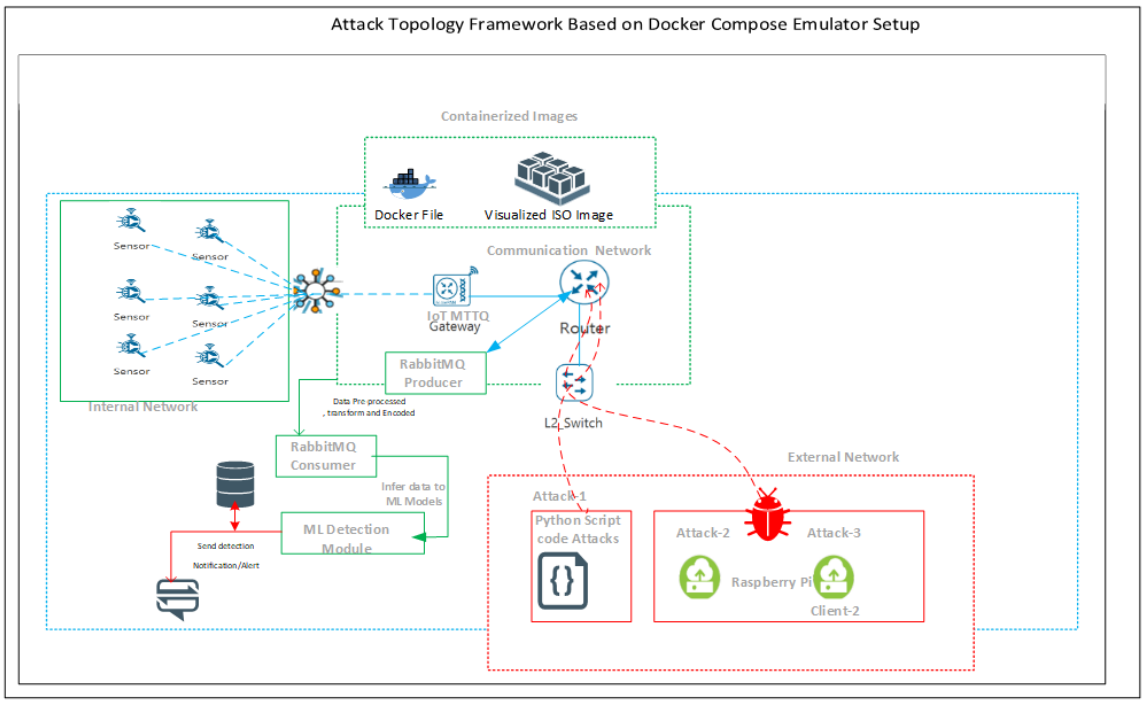

Figure 5-1: Attack Topology Development Setup

5.2.4.1 Development Components Description in Detail

As described above in Figure 5-1 the development environment has three sections: the internal network, external network and communication network. Description of each is presented below. A low level network communication topology is shown on Figure 5-2.

- **Internal Network Components:** In this research we have used the IoT devices as the internal network components that are directly connected to external network via the router using the MTTQ protocol standard. In addition, we have used various seven categories of IoT devices namely, smart camera, smart fridge, smart bulb, smart plug, smart thermostat, smart lock, and smart speaker that gives and includes for comprehensiveness of the research where each smart IoT devices have their own IP address (i.e. the internal network has their own network range 172.16.1.0/24 assigned as shown in the Figure 5-2) that directly communicate with external networks via the router. Furthermore, despite the functional difference among the IoT devices the messaging rule for all the mentioned smart devices used in this research study is the same. Details of the messaging rule are described in Annex-B.
- **Communication Network Components:** The router is of course, the communication component between the internal and external network based on the defined two network interfaces basically eth0 towards external network and eth1 towards internal network that a network address translation is configured on the router emulator container for smooth communication among the connected devices. In addition, the MTTQ Mosquito Hub act is a communication protocol that directly passes the communication to the router.
- **External Network Components:** Here the external network (with own separate network 172.16.2.0/24) is to indicate various IoT attack scenarios that inject an attack to the environment which basically categorized into seven different IoT attack categories namely DDoS attacks, DoS attacks, Recon attacks, BruteForce attacks, web-based attacks, recon attacks, and Miria attacks underneath various attack class types are included (the details of attack types for each category in Annex-C) connected via the layer-2 switch to pass the traffic to the router across the network (both the internal and external).

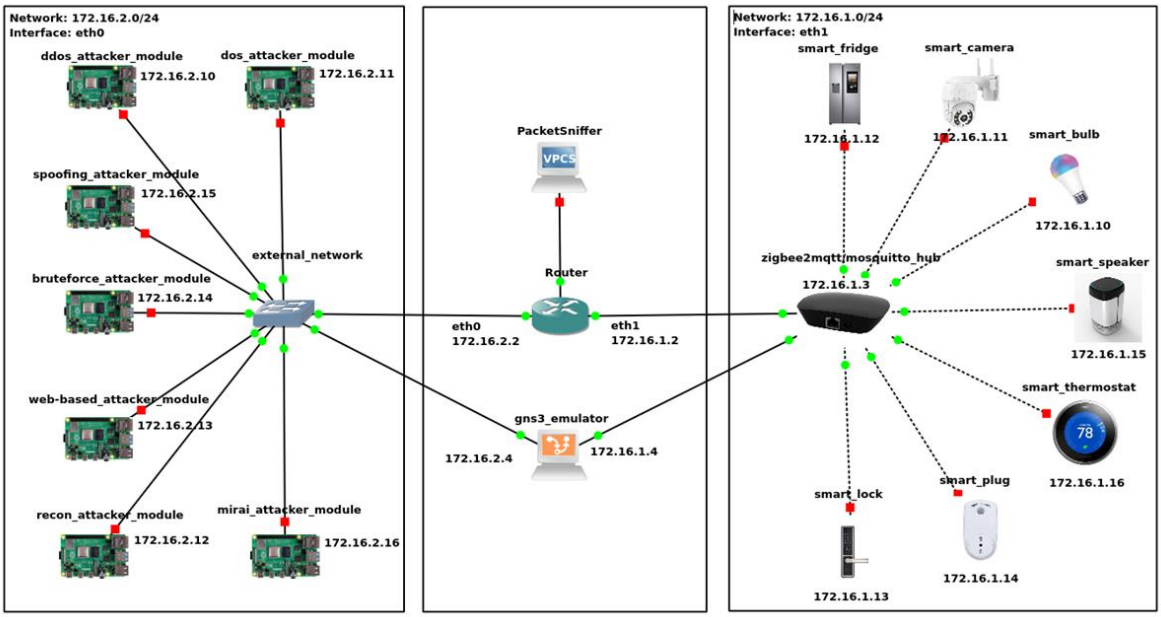

Figure 5-2: Low Level Network Topology

A complete detail docker compose configuration setup is described in Annex-D.

5.3 Prototype Development

To implement the suggested framework, a prototype is developed for detecting multiple cyber-attacks.

5.3.1 Dataset Description

Before utilizing the dataset, a pre-processing information measure shown in Table 5-2 is taken aiming to refine the algorithms for the superior outcomes.

Table 5-2: Dataset Pre-processing Measures

Task	Taken Measures	Details
Null values	Eliminate rows	We eliminated 1 feature (LLC) because of packet level information is irrelevant for the test purpose.
Smooth Noisy Data	Eliminate duplicate values	2733 duplicate records are dropped
	Dropped Attack type	Four attack types are dropped due to insufficient packet sizes.

Header length	0 header information removed	Because of incomplete fragments those who have 0 header length have been dropped
Removing outliers	No outlier	Since it takes the packet from the actual /Emulated hardware devices.
Resolving inconsistencies	Constant Values	3 constant columns values are eliminated
Feature Selection	Based features (18) and drive features (23) are selected	Important to identify stable features that enable effective attack detection
Feature Extraction	Window-based	Focus on the changes in the data flow between the source and destination and we utilize the changes in the network data that occur as each packet arrives by using expanding window size (WS)

After pre-processed attributes of the datasets profile that will be documented accordingly as mentioned in Table 5-3 below and details of the dataset profile are described in Table 5-4.

Table 5-3: Dataset Profile (After pre-processing) Description

Dataset Profile Description	Values
Datapoint Shape size	1.1 million
Number of features	41
Features used for model generation	33
Number of Classes	18
Scale	No Scaler, Standard, MinMax
Window Sizes (packets used for model training)	5,10,20,50,100,200
Null value count	0

Duplicate count	0
Missing Values & Zero Variance column	0

A train_test_split Information (80/10/10):

- Train shape: X_train = (879989, 33), y_train = (879989)
- Test shape: X_test = (109990, 33), y_test = (109990)
- Validation: X_val= ([110008,33) = (110008)

In this research study datasets are divided using an 80% training set, 10% testing set, and 10% validation ratio. The reason to 80% training 10% validation, and 10% training is that primary, 80//10/10 split ensures the test set represents the data distribution. This helps assess the model's performance on unseen data. The bigger training set (80%) gives the model more data to learn from, making it more robust and generalizable. In addition, the 10% evaluation set is utilized for hyperparameter adjustment and model selection. Since the model is not optimized on the test set, this prevents overfitting. Finally, a conventional 80/10/10 split keeps test set conditions similar, making it easy to compare new model performance to earlier research.

The proposed farmwork is trained and evaluated based on one original and 18 a final version dataset created based on six window size of 5, 10, 20,50,100, and 200 and three scaler type (no scaler, minmax scaler, and standard scaler for each window size) with a total size of 60GB (see the details of dataset attack captured data in Annex-A) and key dataset features are selected accordingly as mentioned in Table 5-5.

Table 5-4: Detail Dataset Profile Description

Number	Dataset Version	Window Size	Input Train Shape Dataset	Scaler
1	dataset_original.csv	0	(879989, 22)	No Scalar
2	dataset_ws_5.csv	5	(879989, 33)	No Scalar
3	dataset_ws_5_mms.csv	5	(879989, 33)	MinMax
4	dataset_ws_5_ss.csv	5	(879989, 33)	Standard
5	dataset_ws_10.csv	10	(879989, 33)	No Scalar
6	dataset_ws_10_mms.csv	10	(879989, 33)	MinMax
7	dataset_ws_10_ss.csv	10	(879989, 33)	Standard
8	dataset_ws_20.csv	20	(879989, 33)	No Scalar
9	dataset_ws_20_mms.csv	20	(879989, 33)	MinMax
10	dataset_ws_20_ss.csv	20	(879989, 33)	Standard

11	dataset_ws_50.csv	50	(879989, 33)	No Scalar
12	dataset_ws_50_mms.csv	50	(879989, 33)	MinMax
13	dataset_ws_50_ss.csv	50	(879989, 33)	Standard
14	dataset_ws_100.csv	100	(879989, 33)	No Scalar
15	dataset_ws_100_mms.csv	100	(879989, 33)	MinMax
16	dataset_ws_100_ss.csv	100	(879989, 33)	Standard
17	dataset_ws_200.csv	200	(879989, 33)	No Scalar
18	dataset_ws_200_mms.csv	200	(879989, 33)	MinMax
19	dataset_ws_200_ss.csv	200	(879989, 33)	Standard

5.3.2 Features and Class Information

The feature of the dataset column with its data types has a total count of forty-one and are listed in Table 5-5 with an attack class information described in Table 5-6. Detail of each feature description is described in Annex-E.

Table 5-5: Dataset Feature Attributes with Data Type

#	Column	Dtype	#	Column	Dtype
1	timestamp	float64	22	duration	float64
2	src_mac	object	23	sum_packet_size	float64
3	dst_mac	object	24	max_packet_size	float64
4	src_ip	object	25	avg_packet_size	float64
5	dst_ip	object	26	std_packet_size	float64
6	protocol_type	float64	27	min_duration	float64
7	fin_flag	int64	28	max_duration	float64
8	syn_flag	int64	29	sum_duration	float64
9	rst_flag	int64	30	avg_duration	float64
10	psh_flag	int64	31	std_duration	float64
11	ack_flag	int64	32	fin_count	float64
12	urg_flag	int64	33	syn_count	float64
13	src_port	int64	34	rst_count	float64
14	dst_port	int64	35	psh_count	float64
15	ICMP	int64	36	ack_count	float64
16	Other	int64	37	urg_count	float64
17	TCP	int64	38	tcp_count	float64
18	UDP	int64	39	udp_count	float64
19	encoded_label	int64	40	icmp_count	float64
20	header_length	float64	41	other_count	float64
21	Ttl	float64			

In this research features are categorized and identified as raw features, device specific features (are features that are not relevant for training of model but used for notification purpose only) , non-scale features (features that are not be able to scaled), train features (

features that are used for training of the model), and scale features (features that are used for scaling purpose). The identified features are described in Table 5-6.

Table 5-6: Dataset Features Categories

S. No	Feature Categories	Lists of features
1	Device specific features (7)	Timestamp, src_mac, dst_mac, src_ip, dst_ip, src_port, dst_port
2	Features (33) used for model generation and training	Protocol_type, fin_flag, syn_flag, rst_flag, psh_flag, ack_flag, urg_flag, icmp, tcp, other, udp, header_length, ttl, duration, sum_packet_size, max_packet_size, avg_packet_size, std_packet_size, min_duration, max_duration, sum_duration, avg_duration, std_duration, fin_count, syn_count, rst_count, psh_count, ack_count, urg_count, tcp_count, udp_count, icmp_count, other_count
3	Scale feature (22)	Train features except (Protocol_type, fin_flag, syn_flag, rst_flag, psh_flag, ack_flag, urg_flag, icmp, tcp, other, udp,)
4	Non-scale features (15)	Timestamp, encoded_label, protocol_type, fin_flag, syn_flag, rst_flag, psh_flag, ack_flag, urg_flag, src_port, dst_port, icmp, tcp, udp, others.

Table 5-7: Attack Class Count Information's

Lebel	Packet Class	Attack Class	Descriptions
0	'Benign_network'	'Benign'	
1	'Ddos_ack_fragmentation'	'DDoS'	An attack where attackers send large volume of fragmented TCP ACK packets to target network
2	'Ddos_icmp_flood'	'DDoS'	Type of attack that attacker overwhelm a target device with ICMP echo-request packets, causing the target inaccessible
3	'Ddos_icmp_fragmentation'	'DDoS'	Use datagram mechanisms to overwhelm the network using ICMP protocol
4	'Ddos_pshack_flood' '	'DDoS'	Disruption of network activity by saturating bandwidth & resources on stateful devices by sending PSH-ACK packets.
5	ddos_rstfin_flood'	'DDoS'	Disruption of network activity by saturating bandwidth & resources on

			stateful devices by sending RST-FIN packets.
6	'Ddos_syn_flood'	'DDoS'	Make targets unavailable to legitimate traffic by consuming all available resources
7	'Ddos_synonymousip_flood'	'DDoS'	Target server receives many spoofed TCP SYN packets with the same source and destination address in the header that the target consumes the resources to process those packets.
8	'Ddos_tcp_flood'	'DDoS'	Use datagram mechanisms to overwhelm the network using TCP protocol
9	'Ddos_udp_flood'	'DDoS'	Use datagram mechanisms to overwhelm the network by using UDP protocol
10	'Ddos_udp_fragmentation'	'DDoS'	Type of attack that attackers overwhelm a target device with UDP echo-request packets, causing the target inaccessible
11	'Dos_syn_flood'	'DoS'	Attackers' floods targets with several requests without finalizing the connections
12	'Dos_tcp_flood'	'DoS'	Seek to exploit TCP to three-way handshake mechanism and overwhelm the target unresponsive to legitimate requests
13	'Dos_udp_flood'	'DoS'	Seek to exploit UDP protocol mechanism and overwhelm the target unresponsive to legitimate requests
14	'Mirai_greeth_flood'	'Mirai'	Includes a layer 2 frame Transparent Ethernet Bridging over GRE-encapsulated packets.
15	'Mirai_greip_flood'	'Mirai'	The attack encapsulates a 512-byte payload inside of GRE packets.
16	'Mirai_udp_plain_flood'	'Mirai'	An attack that uses a randomization of the source port and the destination ports. When combined with multiple source IPs (coming from multiple bots) the result is a flood of UDP traffic.
17	'Recon_port_scan'	'Recon'	Find and figure out open ports whether they are receiving or sending data.

5.4 Model Evaluation

For model evaluation, four deep learning models CNN, FFNN, LSTM, and hybrid model are selected, and an implicit learning technique is applied for each individual model to produce an efficient and effective detection across multiple IoT cyberattacks as per the defined selected features.

For a model generation hyperparameters (such as hyperband, and random search) are outlined below in Table 5-8, with detailed input values provided for each method and/or model.

Table 5-8: Model Hyperparameters Values

Inputs	Models			
	FNN	**CNN**	**LSTM**	**Hybrid**
Max Trials	10			
Tuner & Train Epoch	10,25			
Dense Units	(64, 256, step=64)			
Dropout	(0.2, 0.5, step=0.1)			
Hidden Layer Activation	ReLU tanh	ReLU tanh		ReLU Tanh
Hidden Layer Count	(1, 3)	3	3	3
Output Method	Dropout	Dense	Dense	Dense
Output Unit	1	(64, 256, step-64)		
Output Activation	Softmax, Sigmoid			
Optimizer	(adam, rmsprop, sgd, dadam, adadelta, adagrad)			
Pool Size L2				2
Kernal size L1				7
Attention Layer & Dimention				96, 2
Num heads				6

5.4.1 Model Evaluation Metrics

To examine the performance of attack detection models trained with the features sets we employ evaluation of the deep learning models and configuration given that TP represents the True Positives, TN the True Negatives, FP the Falsas Positive, and FN False Negatives metrices are used are recall, accuracy, and F1- score, and precision based on window-based features. Models will also be evaluated using cross-validation to achieve a near-perfect discrimination for selected attacks using the selected model.

- **Accuracy:** responsible for evaluating the classification models by depicting the proportion of correct predictions in each dataset and is based on the following expression:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

- **Recall:** the ratio of correctly identified labels to the total number of occurrences of that label:

$$Rec = \frac{TP}{TP + FN} \quad (2)$$

- **Precision:** the ratio of correctly identified labels to the total number of positive classifications:

$$Pre = \frac{TP}{TP + FP} \quad (3)$$

- **F1-Score:** geometric average of precision and recall:

$$F1 = 2 \times \frac{Pre \times Rec}{Pre + Rec} \quad (4)$$

Optimal hyperparameters identified include 25 epochs, 5000 batch size, 0.00001 learning rate, and evaluation every 8 iterations, as presented in Table 5-11 below.

Table 5-9: Optimal Hyperparameter Values

Hyperparameter	Best Value	Description
Epochs	25	The number of times the entire dataset is passed forward and backward through the deep learning model.
BATCH_SIZE	5000	Determines the number of samples that will be propagated at once in the model.
LR (learning rate)	0.00001	Manages how much to update the model in response to the estimated error every time the model weights are updated.
ITERS_PER_EVAL	8	Specifies how often the model's performance is evaluated on the validation set during training. Frequent evaluation can provide insights into the model's progress but adds overhead.
LAMBDA	0.0000001	A regularization term or a weighting factor in the loss function. Regularization helps prevent overfitting by adding a penalty for complexity.
Num_layers	1 input, 3 hidden and 1 output layers	The first layer is the input layer from the raw dataset, with three hidden layers and the last layer is the output layer of our model. executed across these layers.

5.5 Result Discussion

In this section, outcomes of the research are discussed against the research gaps mentioned in the related work section.

5.6.1 Multiple Detection Top Model Performance Results

Our proposed solutions leverage detection of IoT cyberattacks types and notify into the notification dashboard. The DMICAF evaluation is focused and implemented in four IoT attack categories (namely DDoS, DoS, Recon, and Maria) with sub-attack class underneath each attack categories with various parameters such as window-size, scale type, and different models and based on the parameter values the top performing model results are described below in Table 5-10. The hybrid model used is based on CNN and LSTM combinations.

Table 5-10: Top Performing Model Evaluation Results

No	Algorithm	Window Size	Scaler	Accuracy Score	Precision Score	Recall Score	F1 Score
1	CNN	200	Minmax	0.914245	0.897778	0.914245	0.901111
2	CNN	200	Standard	0.896845	0.911978	0.896845	0.890753
3	CNN	100	Standard	0.856714	0.856623	0.856714	0.855165
4	CNN	100	Minmax	0.840831	0.883632	0.840831	0.820334
5	Hybrid	100	Standard	0.827475	0.82688	0.827475	0.812218

For ease of analysis and constituency, we employed to use the nomenclature of WS/ws indicated Window Size, SS/ss indicated Standard Scaler, MM/mm indicated MinMax Scaler, NS/ns-indicated No Scaler used for each model for example, if we use a dataset of WS_200_mm indicated as window size of 200 with MinMax scaler type which applies for other window size and scaler types.

As shown on Figure 5-3 below, when we compare the detection results of WS_200 with minmax and apply the same window size with standard scaler the minmax scaler detection results increased by 1.74%. However, it decreased by -4.01% when we compared to window size of 100. In addition, when we compare window size 100 with standard scaler to window size 200 with minmax scaler the result shows an increased accuracy detection by 5.75% but when we compare the same scaler of minmax but with different window sizes 100, and 200 the result drastically decreased by 7.34%. In other words, if the window size increased the

detection of accuracy increased but again when we compared by applying different scaler types for example, within the same window size of 200 the minmax scaler type detection accuracy results 0.9142 compared to standard scaler type results of 0.8968 which increased by 1.74%.

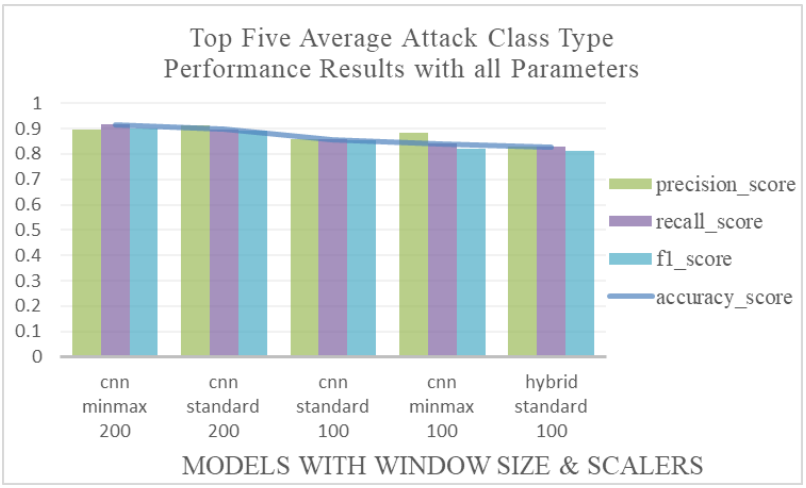

Figure 5-3: Top five Model Performance Results

Below Figure 5-4. Described the CNN model confusion matrix detection results for individual attacks.

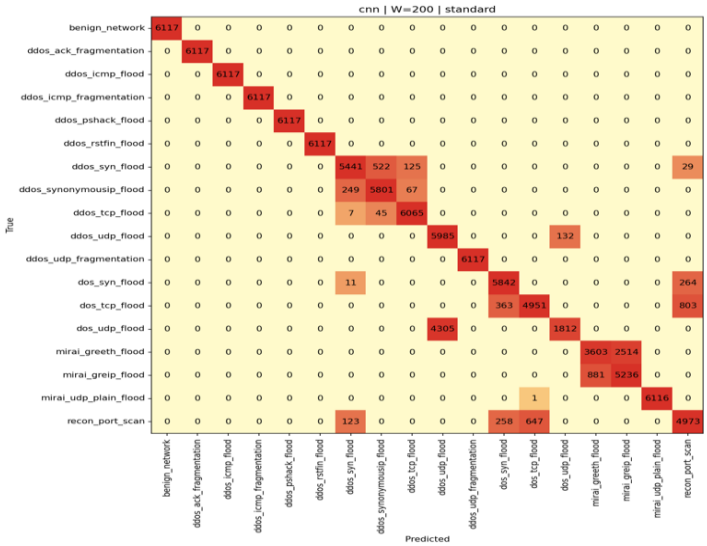

Figure 5-4: Confusion Matrix Results for CNN with Individual Classification

5.6.2 Multiple Model Comparison Based on Algorithm, Window Size & Scaler

- **FFNN & CNN Model Comparison**

Comparing the performance results of FFNN and CNN models with all window size's and without applying a scale the performance of CNN increased compared to FFNN model as described in Figure 5-5.

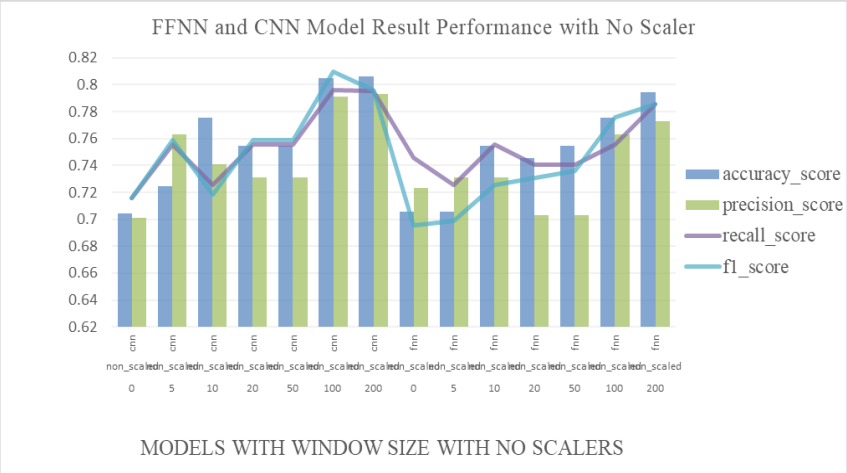

Figure 5-5: FFNN and CNN Model Performance Result with No Scaler

When we compare the individual performance for instance, with window sizes of 100 and 200 the average accuracy of FFNN results 77.55% and 79.46% respectively which is 1.91% accuracy increased on the other hand, the average accuracy of CNN model gives 80.46% and 80.60% accuracy respectively with 0.14% increased within the same window sizes comparison. The detection results between the two algorithms CNN performs 2.90% detection accuracy than FFNN for window size of 100 and 1.14% detection accuracy for window size of 200 respectively.

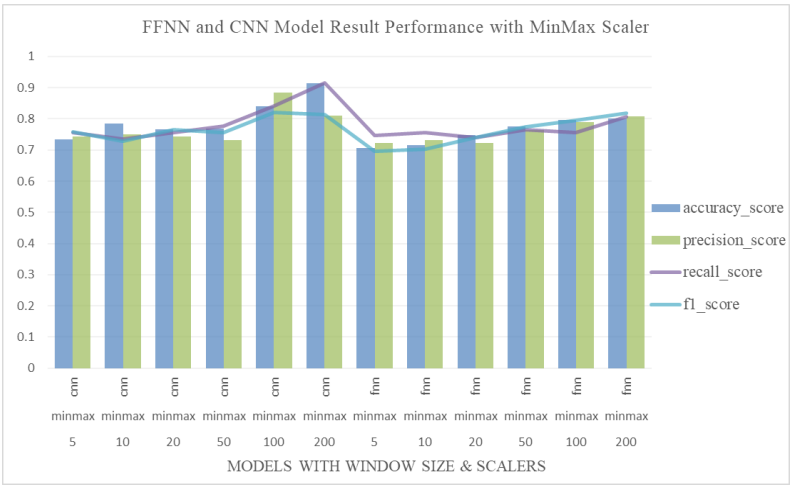

Figure 5-6: FFNN and CNN Model Performance Result with MinMax Scaler

As we can see on Figure5-6, when we compare the performance results with MinMax Scaler type again, CNN performs very promising performance result of 91.42% detection accuracy with a window size of 200 compared to FFNN which is 80.17% detection accuracy which decreased by -11.23%. When we compared with no scaler type and minmax scaler within the same window size of 100, and 200 the detection score results increased by 3.63%, and 10.83% respectively for CNN model and 2.0%, and 0.71% increase for FFNN model.

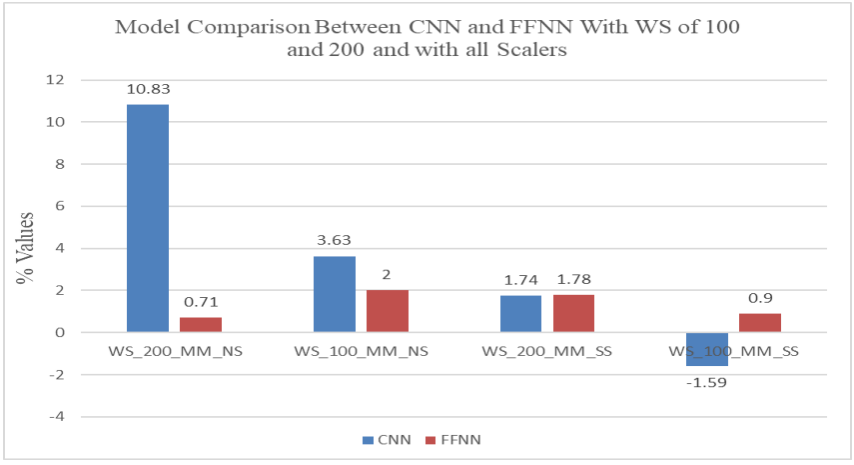

Figure 5-7: CNN and FFNN Performance with WS 100 & 200 with all Scalers

As shown in Figure 5-7 with window sizes of 100, and 200 compared to MinMax and Standard Scaler CNN algorithm the MinMax detection accuracy decreased by -1.59% but increased by 1.74% respectively. Again, for FFNN model it increased by 0.90% for window size 100, and 1.78% for window size of 200.

- **FNN and LSTM Model Comparison**

When we compare the performance results of FFNN and LSTM model with all window size's and without applying a scale the performance of FFNN increased compared to LSTM model as described in Figure 5-9 below.

If we compare the individual performance for instance, with window sizes of 100 and 200 the average accuracy of FFNN results 77.55% and 79.46% respectively which is 1.91% accuracy increased on the other hand, the accuracy of LSTM model gives 66.56% and 71.06% accuracy respectively with 4.45% increased within the same window sizes comparison. The detection

results between the two algorithms FFNN performs 10.99% detection accuracy than FFNN for window size of 100 and 8.4% detection accuracy for window size of 200 respectively.

Figure 5-8: FFNN and LSTM Performance Results with No Scaler

When we compare the performance results with MinMax Scaler type shown in Figure 5-10 FFNN model performs very promising performance result of 80.17% detection accuracy with a window size of 200 compared to LSTM which is 74.83% detection accuracy which is decreased by -5.34%. Additionally, when we compared with no scaler type and minmax scaler within the same window size of 100, and 200 the detection score results increased by 1.55%, and 3.77% respectively for LSTM model and 2.0%, and 0.71% increase for FFNN model.

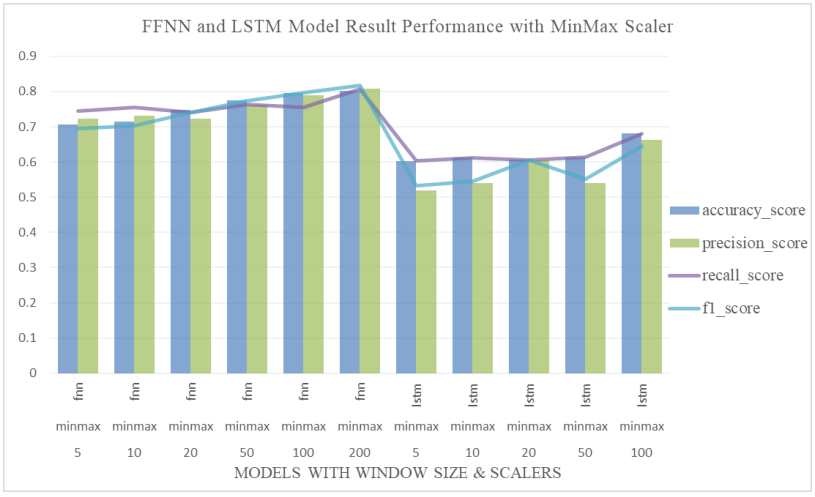

Figure 5-9: FFNN and LSTM Performance with MinMax Scaler

As shown in Figure 5-11, compared to MinMax and Standard Scaler LSTM MinMax detection accuracy decreased by -6.43% for window size 100 but increased by 2.26% for window size 200. Again, for FFNN model it increased by 0.90% for window size 100, and 0.78% for window size of 200.

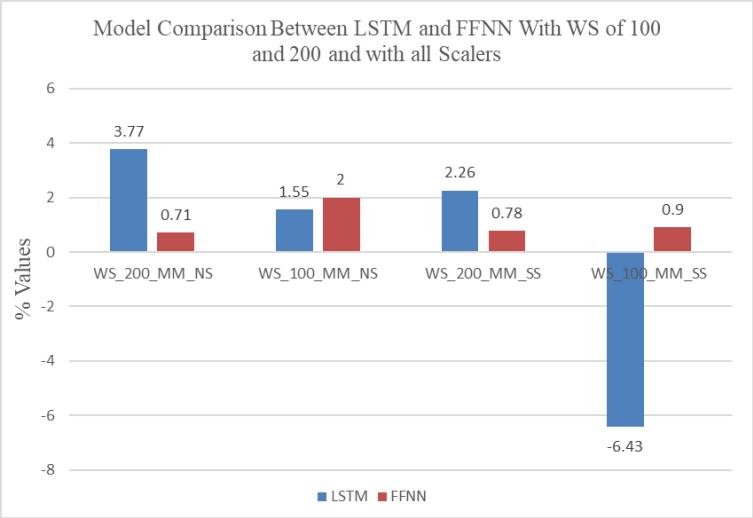

Figure 5-10: LSTM and FFNN Performance with WS 100 & 200 with all Scalers

From the overall results analysis of the two model (FFNN and LSTM) we can easily conclude that when the window size increases the detection accuracy results better performance and applying a scaler type such as MinMax and Standard scaler has a better performance than non-scaler (even if exceptionally LSTM model with no scaler has 3.06% exceeded), the MinMax scaler has better detection accuracy than Standard Scaler and finally, with all parameters FFNN model has a better result than LSTM model which has 0.45% increased, 7.33% increased within MinMax and NS of 100, and MinMax and SS of 100 WS respectively however, -1.48% decreased for 200 window size between MinMax and standard scaler.

- **CNN and LSTM Model Comparison**

When we compare the performance results of CNN and LSTM model with all window size's and without applying a scale the performance of CNN model exceeds LSTM model as described in Figure 5-13 below.

If we compare the individual performance for instance (without no scale is applying), with window sizes of 100 and 200 the average accuracy of CNN results 80.46% and 80.60%

respectively which is 0.14% accuracy increased on the other hand, the accuracy of LSTM model gives 66.56% and 71.06% accuracy respectively with 4.45% increased within the same window sizes comparison.

Figure 5-11: CNN and LSTM Performance with No Scaler

But, from the detection results between the two algorithms CNN performs 13.90% detection accuracy than LSTM for window size of 100 and 9.54% detection accuracy for window size of 200.

Comparing the performance results with MinMax Scaler type shown in Figure 5-14 CNN model performs very high-performance result of 91.42% detection accuracy with a window size of 200 compared to LSTM which is 74.83% detection accuracy which is significantly decreased by -16.60%.

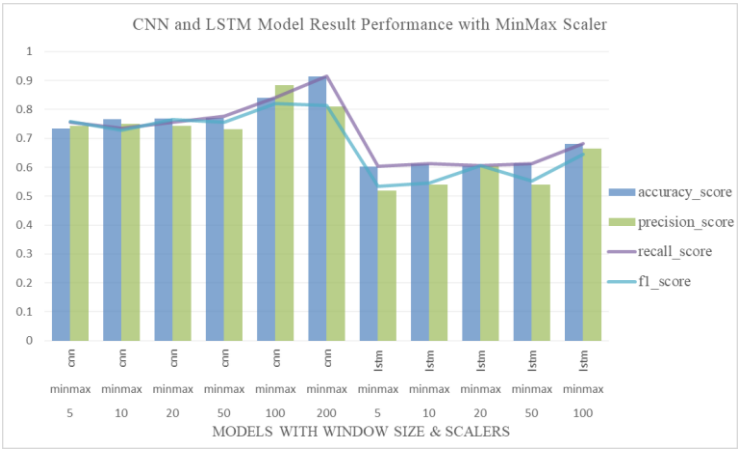

Figure 5-12: CNN and LSTM Performance with MinMax Scaler

Furthermore, when we compared the no scaler type with minmax scaler within the same window size of 100, and 200 the detection score results increased by 1.55%, and 3.77% respectively for LSTM model and 3.36%, and 10.83% increase for CNN model respectively.

Figure 5-13: CNN and LSTM Performance with Standard Scaler

As shown in Figure 5-16, compared to MinMax and Standard Scaler the MinMax detection accuracy for LSTM model decreased by -6.43% for window size 100 but increased by 2.26% for window size 200. But, for CNN model it increased by 3.63% for window size 100, and 10.83% compared with no scaler for window size of 200.

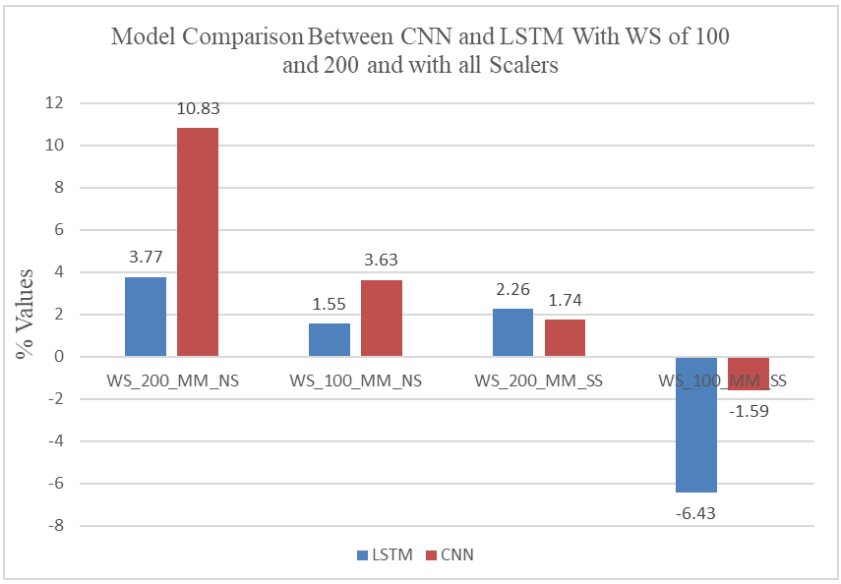

Figure 5-14: CNN and LSTM Percentage with WS 100 & 200 with all Scalers

In conclusion, from the overall results analysis of the two models we can easily conclude that when the window size increases the detection accuracy results in better performance and

applying a scaler type such as MinMax and Standard scaler has a better performance than non-scaler particularly, the MinMax scaler has better detection accuracy than Standard Scaler. Finally, with all parameters CNN model has a better result than LSTM model which has 7.06%, and 2.08% increased within MinMax and NS of 200, 100 WS respectively and 4.84% detection accuracy increased within MinMax and SS of 100 WS however, -0.52% decreased for 200 window size.

5.6.3 Overall Performance Results with Multiple Attributes

Our proposed detection framework applied multiple attributes for the cyberattacks detection and the finding results provide promising and outperformed performance results based on those applied attributes mentioned which are window-size, multiple attack classes, multiple algorithm and different scaler types.

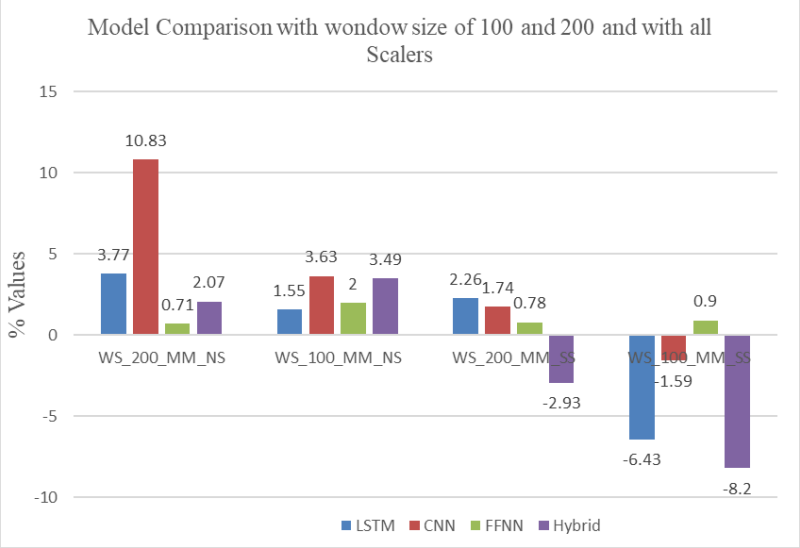

Figure 5-15: All Model with Window size of 100 and 200 with all Scalers

As described above Figure 5-17 when we compare MinMax (MM) and None-Scaler (NS) with window size of 100, 200 CNN models outperforming 3.63%, and 10.85% detection accuracy for MinMax respectively. In other words, the performance detection of applying MinMax scalers has a better performance than with non-scalers whereby CNN model has exceeded the detection accuracy by 7.06% compared with LSTM, 8.76% compared with Hybrid model, and 10.12% compared to FFNN model for window size 200 and 2.08%, 0.14%, and 1.63% for window size 100 respectively. When we compare MinMax (MM) and Standard Scaler (SS)

with window size of 200 the MinMax model result shows that a 2.26%, for LSTM 1.74% CNN, and 0.78% and FNN performance increased than standard scaler but, MinMax scaler for hybrid model decreased by -2.93% which means Standard Scaler has better performance than MinMax scaler in case of hybrid model. Below Figure 5-16 and Figure 5-17 show the train and loss validation of CNN model for window size 200 of Standard scaler and window size 100 of minmax scalar respectively.

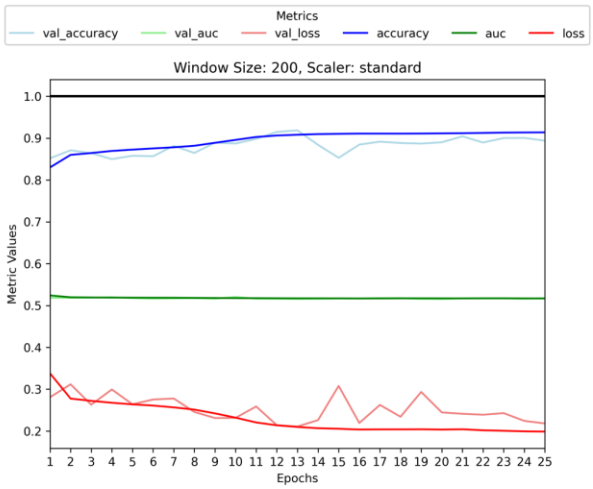

Figure 5-16: Train and Loss validation for WS 200 Standard Scaler

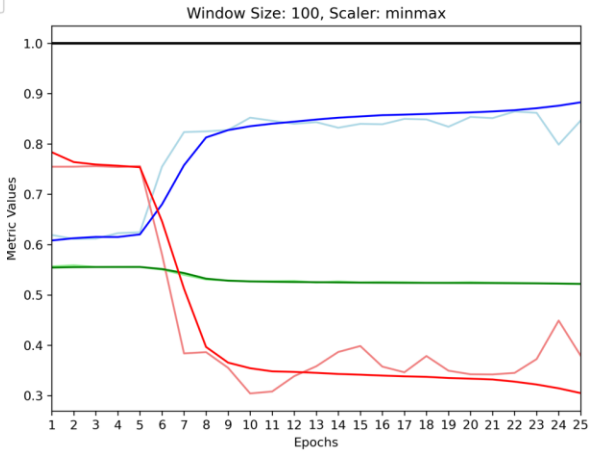

Figure 5-17: Train and Loss validation for WS 100 MinMax Scaler

Based on the optimized hyperparameter values compared to the algorithm performance, CNN is outperforming very well followed by Hybrid, FFNN, and LSTM with an accuracy value of 91.42%, 82.75%, and 78.38% ,74.83% detection accuracy respectively. On the other hand, we realized that when the window size increases the detection accuracy also become increasingly outperformed well in all models. In addition, scaler types also have their own impact on the performance of the detection accuracy where MinMax scaler followed by Standard scaler has better performance compared to non-scaler type.

5.6.4 Individual Attack Class Performance

From the individual attack detection point of view with window size of both 100, and 200 and in both scalers of minmax and standard CNN and hybrid model outperformed almost a 100% detection precision values for attack classes of ack_fragmentation, Ddos_icmp_flood, Ddos_icmp_fragmentation, _Ddos_pshack_flood Ddos_rstfin_flood described in Table 5-11.

Table 5-11: Individual Attack Class Classification Performance Results

CNN (WS_200 Standard Scaler)

Attack Class	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	0.98	0.98
6	0.93	0.88	0.91
7	0.911	0.92	0.92
8	0.969	0.97	0.97
9	0.5816	0.5584	0.5296
10	1.00	0.97	0.97
11	0.9039	0.95	0.92
12	0.8843	0.8	0.84
13	0.9321	0.92	0.9244
14	0.8035	0.589	0.67
15	0.6756	0.856	0.7552
16	1.00	0.96	0.98
17	0.8194	0.82	0.824

CNN (WS_200 MinMax Scaler)

Attack Class	Precision	Recall	F1-score
0	0.99	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	0.98	1.00
5	1.00	0.9689	1.00
6	0.93	0.83	0.88
7	0.87	0.93	0.9
8	0.9	0.98	0.94
9	0.75	0.83	0.79
10	1.00	0.96	1.00
11	0.97	0.92	0.92
12	0.9	0.87	0.88
13	0.86	0.79	0.82
14	0.89	0.92	0.91
15	0.93	0.89	0.91
16	1.00	1.00	1.00
17	0.17	0.6	0.27

CNN (WS_100 Standard Scaler)

Attack Class	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	0.8589	0.7294	0.788
7	0.7837	0.9147	0.8441
8	0.9673	0.9475	0.9573
9	0.6488	0.7144	0.68
10	1.00	1.00	1.00
11	0.8981	0.8753	0.8866
12	0.7578	0.8839	0.816
13	0.6823	0.6134	0.646
14	0.5579	0.4824	0.5174
15	0.5634	0.606	0.584
16	0.9437	1.00	0.9711
17	0.7552	0.6499	0.6986

CNN (WS_100 with MinMax Scaler)

Attack Class	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	0.8502	0.9155	0.8816
7	0.9827	0.9313	0.9564
8	0.8981	0.9787	0.9367
9	0.5882	0.9027	0.7123
10	1.00	1.00	1.00
11	0.9864	0.5458	0.7028
12	0.7857	0.8019	0.7937
13	0.791	0.3681	0.5025
14	0.99	0.3415	0.1154
15	0.513	0.676	0.5834
16	1.00	1.00	1.00
17	0.513	0.676	0.5834

The graphical representation of individual classification report is described in Figure 5-21 below. Even though the individual attack class detection performance for DoS attack has better performance still there is some low detection accuracy across all parameters compared to DDoS individual detections.

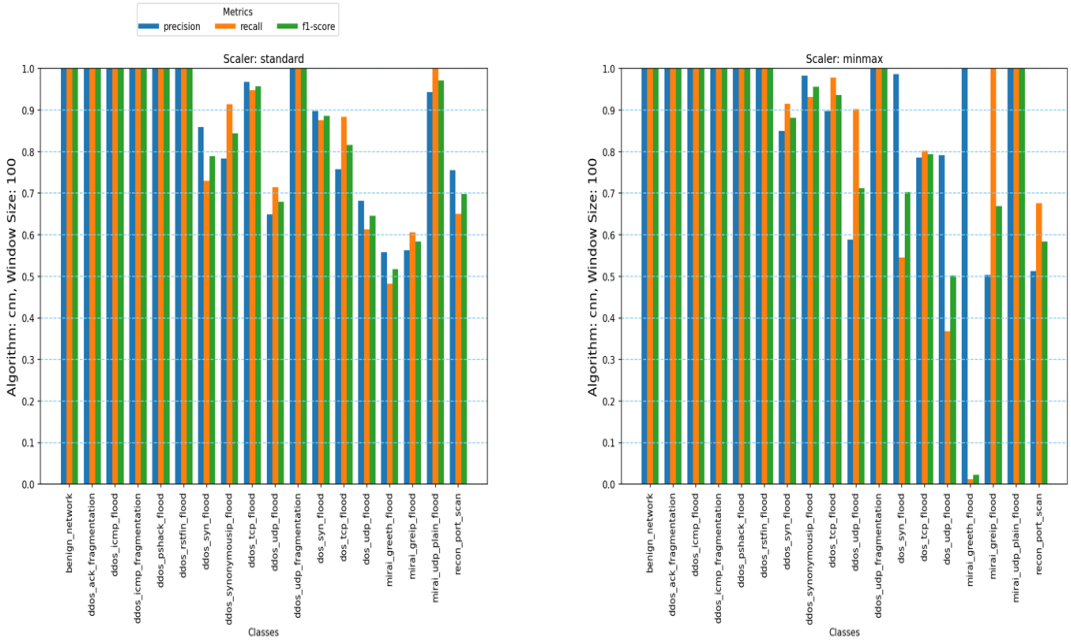

Figure 5-18: Individual Attacks Classification Performance for CNN with WS 100 & 200 of minmax and Standard Scaler

5.6.5 Model Comparison with Existing Research

The proposed model is extensively evaluated based on different parameters and features. Although the existing studies have higher accuracy detection values than the proposed framework, it gives prominent detection results based on those multiple parameters whereby the existing research studies lack incorporate those features and parameters. The detail of the comparison is described in Table 5-12.

Additionally, almost all the existing studies evaluated metrics evaluated and employed on single class attacks, limited number of features representation, and limited parameters which impacted on the performance compared to our proposed framework mainly employed and evaluated on multiple parameters and features. Furthermore, compared to individual level attack detection values, however, our proposed framework also provides greater than 99.99% better accuracy values than the existing studies.

Table 5-12: Comparison of Proposed framework with Existing Study

Authors	Best Model	Accuracy	Hyperparameter Setting Used		Remarks
			Parameters	Values	
[19]	DNN	93%	Not Specified	Not Specified	Individual class Performance report missed
[91]	DT	99.79%	Max feature	7	Used only limited Packet size and parameters
			Min Sample split	2	
			Splitting Criteria	Gini	
			Window Size	32	
[94]	CNN	95.20%	Activation function	ReLU	Limited number of single class attacks and individual class reports are missing Data imbalance issues
			Number of filtering Layers	20	
			Convolutional Layer	5	
[95]	DNN	99.24%	Activation function	ReLU, Sigmoid	
			Number of hidden layers	5	
[90]	SVM	99.71%	Kernal	Sigmoid	
			Parmeter coefficient	Not specified	
			Kernal Coefficient	Not Specified	
[89]	FFNN	99.97%	Activation function	ReLU, Sigmoid	
			Number of hidden layers	3/4/5	
			Optimizer	Adam	
			Epoch	10/15/20/30	
[96]	BLSTM	99.41%	Activation function	(ReLU, Sigmoid)	
			Hidden Nodes	60	
			Batch Size	100	
			Optimizer	Adam	
			Epoch	200	
	Proposed Framework (DMICA)	91.42%	Window Size	5/10/20/50/100/200	Proposed model used multiple parameters such as multiple window size, scalar types & Multiple optimizer
			Scaler type used	3	
			Activation function	ReLU, Sigmoid	
			Hyperparameter	Hyperband	
			Optimizer (Adam)	Adam, adadelta	
			Epoch	25	

Chapter 6 Conclusion and Future Work

This Chapter summarizes the major findings of this research work, contribution of the proposed framework, and future work are outlined.

6.1 Conclusion

IoT environments are characterized by interconnection of everything through sensors and ubiquitous computing that gives humans seamless communication very easily. Undoubtably, the current and emerging IoT technologies are creating a flexible, scalable, and easily manageable that contributes a lot to day-to-day human life activities. Even though the interconnection of everything by itself doesn't guarantee in all aspects as technology grows an increased cyberattacks becoming the new weapons of human warfare across the globe.

A single cyberattack detection framework is relatively high detection performance for specific IoT cyberattacks, however, the proposed framework still lacks detecting multiple cyberattacks while it includes multiple parameters. As a result, with an intensive review of literature and related works we identified a gap in detection of IoT cyberattacks by considering multiple parameters, network behavioral patterns, and heterogenous nature of datasets features used are unsolved issues of IoT environments. Thus, we proposed a framework to detect multiple IoT cyberattacks that could fill the identified gaps.

The proposed framework contains three primary main modules; primary module responsible for capturing and pre-processing the captured data and ready for the construction of the model, then the core engine moule orchestrates the detection of cyberattacks. The third module, notify and displays the results in a dashboard.

This research study used multiple parameters including multiple attack classes, network packet patterns, and three scalar types namely no scaler, MinMax, Standard and regardless of the defined parameters used minmax scaler followed by standard scaler gives better detection performance than models trained with no scaler. In terms of the network packets patterns, we used different packet sizes of 5,10,20,50,100, and 200 whereby we realize that as the number of packet size increased the detection accuracy also increased with respect to any of the other additional combined parameters.

In conclusion, based on multiple parameters values the proposed framework is trained and tested with different models including CNN, Hybrid, FFNN, and LSTM provides a value of 91.42%, 82.75%, and 78.38%, and 74.83% detection accuracy respectively where CNN outperforms the optimal results followed by hybrid and FFNN.

6.2 Contribution of Research

The main contribution of the research study is described as follows:

- **Use in IoT Cybersecurity Detection:** Integrating the proposed framework into multiple IoT cyberattack detection solution has a substantial contribution across IoT environment this is because packet patterns, scalers, attack classes, and dataset features have a significant impact for efficient multiple IoT attack detection.
- **Real-time Integration:** Improving defenses against evolving IoT threats lead to enhanced security and safety of IoT devices, software, and services, thus the proposed framework has the capability of integration in terms of IoT cyberattacks detection.

6.3 Future work

A framework on detecting multiple IoT cyberattacks proposed in this study could be used in various IoT cyberattacks detection with a proper configuration and setup. Even though the proposed framework presents the detection of multiple attack scenarios, this research has limited specific areas that need further improvements described below:

- **Tuning and Optimization:** By incorporating various deep learning models and additional parameter values, it needs further tuning and optimization for better results and improvement.
- **Additional Attack Classes:** Adding additional attack class type (such as web attack class) is also another improvement area that needs to optimize for emerging cyberattacks type techniques.
- **Additional Datasets:** The proposed framework is based on a single IoT dataset. It is very important to incorporate two or more datasets other than the dataset used in this research study which results in an improvement in multiple cyberattack detection patterns.

References

[1] P. Newman, "The internet of things 2020," Business Insider,, 2020.

[2] E. SavitEricz, "Forbes," 04 January 2013. [Online]. Available: https://www.forbes.com/sites/ericsavitz/2012/10/23/gartner-top-10-strategic-technology-trends-for-2013/. [Accessed 25 Aug 2023].

[3] P. High, "Forbes," Forbs, 25 November 2014. [Online]. Available: https://www.forbes.com/sites/peterhigh/2013/10/14/gartner-top-10-strategic-technology-trends-for-2014/. [Accessed 25 August 2023].

[4] M. Roopak, G. Y. Tian and J. Chambers, "An intrusion detection system against ddos attacks in IOT Networks," *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, January 2020.

[5] A. Aldhaheri, F. Alwahedi, M. Ferrag and A. Battah, "Deep learning for cyber threat detection in IOT Networks: A Review," *Internet of Things and Cyber-Physical Systems*, pp. 110-128, 2024.

[6] P. Friess and O. Vermesan, "Internet of things applications - from research and innovation to market deployment," 2022.

[7] W. B. Group, "worldbank.org," [Online]. Available: https://documents1.worldbank.org/curated/en/610081509689089303/pdf/120876-REVISED-WP-PUBLIC-Internet-of-Things-Report.pdf.. [Accessed 23 August 2023].

[8] A. N. OZALP, Z. ALBAYRAK, M. CAKMAK and E. OZDOGAN, "Layer-based examination of cyber-attacks in IoT," *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2022.

[9] A. Wang, A. Mohaisen and S. Chen, "XLF: A Cross-layer framework to secure the internet of things (IoT)," *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[10] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," *2014 International Conference on Future Internet of Things and Cloud*, 2014.

[11] B. B. Gupta and M. Quamara, "Multi-layered cloud and fog based secure integrated transmission and Storage Framework for IOT based applications," *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2018.

[12] I. Yaqoob, E. Ahmed, I. A. Hashem, A. I. Ahmed, A. Gani, M. Imran and M. Guizani, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10-17, 2017.

[13] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu and D. Qiu, "Security of the internet of things: Perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481-2501, 2014.

[14] S. S. , R. A., G. L. and C.-P. A. , "Security, privacy and trust in internet of things," *Computer Networks*, vol. 70, pp. 146-164, 2015.

[15] Y. Z., Z. P. and V. A. , "A survey on trust management for internet of things," *Journal of Network and Computer Applications,*, vol. 42, pp. 120-134, 2014.

[16] S. Z., U. I., L. H., L. A. and K. K, "Blockchain based solutions to mitigate distributed denial of service (ddos) attacks in the internet of things (IOT): A survey," *Sensors*, vol. 22, p. 1092, 2022.

[17] Abu-Sharkh, N. H. A and O. M, "Code injection attacks in wireless-based internet of things (IOT): A comprehensive review and practical implementations," *Sensors*, vol. 23, no. 13, p. 6067, 2023.

[18] S. S, . P. Y and P. Y , "A secure, lightweight, and Anonymous User Authentication Protocol for IOT Environments," *Sustainability*, vol. 13, no. 16, p. 9241, 2021.

[19] G. Bhandari, A. Lyth, A. Shalaginov and T.-M. Grønli, "Distributed deep neural-network-based middleware for cyber-attacks detection in Smart iot ecosystem: A novel framework and performance evaluation approach," *Electronics*, vol. 12, no. 2, p. 298, 2023.

[20] X. Li, Z. Hu, M. Xu, Y. Wang and J. Ma, "Transfer learning based Intrusion Detection Scheme for Internet of vehicles," *Information Sciences*, vol. 547, pp. 119-135, 2021.

[21] S. Gou, Y. Wang, L. Jiao, J. Feng and Y. Yao, "Distributed Transfer Network learning based Intrusion Detection," *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2009.

[22] S. Latif, Z. E. Huma, S. S. Jamal, F. Ahmed, J. Ahmad, A. Zahid, K. Dashtipour, M. U. Aftab, M. Ahmad and Q. H. Abbasi, "Intrusion Detection Framework for the internet of things using a dense random neural network," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6435-6444, 2022.

[23] J. Vom Brocke, A. Hevner and A. Maedche, "Introduction to design science research," *Progress in IS*, pp. 1-13, 2020.

[24] E. C. P. Neto,, S. Dadkhah, R. Ferreira, A. Zohourian,, R. Lu and A. A. Ghorbani, "CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment.," *Sensors*, vol. 23, p. 5941, 2023.

[25] T.-A. Tran, T. Ruppert, G. Eigner and J. Abonyi, "Retrofitting-based development of Brownfield Industry 4.0 and industry 5.0 solutions," *IEEE Access*, vol. 10, pp. 64348-64374, 2022.

[26] K. Khujamatov, D. Khasanov, E. Reypnazarov and N. Axmedov, "Industry digitalization consepts with 5G-based IOT," *2020 International Conference on Information Science and Communications Technologies (ICISCT)*, 2020.

[27] G. Paolone, D. Lachetti, R. Paesani, F. Pilotti, M. Marinelli and P. Di Felice, "A holistic overview of the internet of things ecosystem," *IoT*, vol. 3, no. 4, pp. 398-434, 2022.

[28] C. Sobin, "A survey on architecture, Protocols and challenges in IOT," *Wireless Personal Communications*, vol. 112, no. 3, pp. 1383-1429, 2020.

[29] P. Sivagami, P. Illavarason, R. Harikrishnan and G. Reddy, "IOT ecosystem- A survey on classification of IOT," *Proceedings of the Fist International Conference on Advanced Scientific Innovation in Science, Engineering and Technology, ICASISET 2020, 16-17 May 2020, Chennai, India*, 2021.

[30] S. Bansal and D. Kumar, "IoT ecosystem: A survey on devices, gateways, operating systems, middleware and Communication," *International Journal of Wireless Information Networks*, vol. 27, no. 3, pp. 340-364, 2020.

[31] M. Burhan, R. A. Rehman, B. Khan and B.-S. Kim, "IoT elements, layered architectures and security issues: A comprehensive survey," *Sensors*, vol. 18, no. 9, p. 2796, 2018.

[32] T.-H. Nguyen and M. Yoo, "A hybrid prevention method for eavesdropping attack by link spoofing in software-defined internet of things controllers," *International Journal of Distributed Sensor Networks*, vol. 13, no. 11, 2017.

[33] R. Hassan, F. Qamar, M. K. Hasan, A. H. Aman and A. S. Ahmed, "Internet of things and its applications: A comprehensive survey," *Symmetry*, vol. 12, no. 10, p. 1674, 2020.

[34] S. P. Dash, "The impact of IOT in healthcare: Global technological change & the roadmap to a networked architecture in India," *Journal of the Indian Institute of Science*, vol. 100, no. 4, pp. 773-785, 2020.

[35] K. Demestichas, N. Peppes and T. Alexakis, "Survey on security threats in agricultural IOT and smart farming," *Sensors*, vol. 22, no. 20, p. 6458, 2020.

[36] N. K. Suryadevara and G. R. Biswal, "Smart plugs: Paradigms and applications in the smart city-and-smart grid," *Energies,* vol. 12, no. 10, p. 1957, 2019.

[37] S. B. Atitallah, M. Driss, W. Boulila and H. B. Ghézala, "Leveraging deep learning and IOT big data analytics to support the Smart Cities Development: Review and Future Directions," *Computer Science Review,* vol. 38, 2020.

[38] M. A. Rahim, M. A. Rahman, M. Rahman, T. A. Asyhari, M. Z. Bhuiyan and D. Ramasamy, "Evolution of IOT-enabled connectivity and applications in Automotive Industry: A Review," *Vehicular Communications,* vol. 27, p. 100285, 2021.

[39] A. R. Khan, M. Kashif, R. H. Jhaveri, R. Raut, T. Saba and S. A. Bahaj, "Deep learning for intrusion detection and security of internet of things (IOT): Current analysis, challenges, and possible solutions," *Security and Communication Networks,* pp. 1-15, 2022.

[40] M. Catillo, A. Pecchia and U. Villano, "Botnet detection in the internet of things through all-in-one deep autoencoding," *Proceedings of the 17th International Conference on Availability, Reliability and Security,* 2022.

[41] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le and D.-H. Nguyen, "A survey of IOT malware and detection methods based on static features," *ICT Express,* vol. 6, no. 4, pp. 280-286, 2020.

[42] B. T. S. , "Internet of Things (IoT): A critical review," *Int. J. Sci. Technol. Res.,* vol. 8, no. 10, pp. 227-232, 2019.

[43] O. B., S. B. and P. B. ,. G. , "Implementation of IoT in Smart Homes," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 6, no. 12, 2017.

[44] Radwan and Nael, "A Study: The Future of the Internet of Things and its Home Applications," *International Journal of Computer Science and Information Security,* 2020.

[45] M. Dhanaraju, P. Chenniappan, K. Ramalingam, S. Pazhanivelan and R. Kaliaperumal, "Smart farming: Internet of things (iot)-based sustainable agriculture," *Agriculture,* vol. 12, no. 10, p. 1745, 2022.

[46] S. Gupta and S. Gupta, "Smart agriculture and farming services using IOT," *Advances in Environmental Engineering and Green Technologies,* pp. 154-165, 2021.

[47] V. Dankan Gowda, M. Sandeep Prabhu, M. Ramesha, J. M. Kudari and A. Samal, "Smart Agriculture and smart farming using IOT Technology," *Journal of Physics: Conference Series,* vol. 2089, no. 1, 2021.

[48] A. Ullah, S. M. Anwar, J. Li, L. Nadeem, T. Mahmood, A. Rehman and T. Saba, "Smart cities: The role of internet of things and machine learning in realizing a data-centric smart environment," *Complex & Intelligent Systems,* vol. 10, no. 1, pp. 1607-1637, 2023.

[49] A. S. Syed, D. Sierra-Sosa, A. Kumar and A. Elmaghraby, "IoT in Smart Cities: A Survey of Technologies, practices and challenges," *Smart Cities,* vol. 4, no. 2, pp. 429-475, 2021.

[50] X. Yang, X. Wang, X. Li, D. Gu, C. Liang, K. Li, G. Zhang and J. Zhong, "Exploring emerging IoT Technologies in smart health research: A knowledge graph analysis," *BMC Medical Informatics and Decision Making,* vol. 20, no. 1, 2020.

[51] Y. Cheng, X. Zhao, J. Wu, H. Liu, Y. Zhao, M. Al Shurafa and I. Lee, "Research on the smart medical system based on Nb-IOT Technology," *Mobile Information Systems,* pp. 1-10, 2021.

[52] A. Rejeb, K. Rejeb, H. Treiblmaier, A. Appolloni, S. Alghamdi, Y. Alhasawi and M. Iranmanesh, "The internet of things (IoT) in Healthcare: Taking Stock and moving forward," *Internet of Things,* vol. 22, 2023.

[53] V. Rigworo Kebande, "Industrial internet of things (IIoT) forensics: Challenges, opportunities, and future directions," 2023.

[54] O. Peter, A. Pradhan and C. Mbohwa, "Industrial internet of things (IIoT): Opportunities, challenges, and requirements in manufacturing businesses in emerging economies," *Procedia Computer Science,* vol. 217, pp. 856-865, 2023.

[55] L. Center, 20 December 2023. [Online]. Available: https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/.

[56] ExtraHop, 21 January 2022. [Online]. Available: https://hop.extrahop.com:443/resources/attacks/remote-code-execution/.

[57] L. Muscat, "Acunetix," 06 March 2023. [Online]. Available: https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/.

[58] T. Gaber, A. El-Ghamry and A. E. Hassanien, "Injection attack detection using machine learning for smart IoT Applications," *Physical Communication,* vol. 52, 2022.

[59] S. Dave and P. S. Chauhan, "Intrusion detection and prevention system in IoT environment," *International Journal of Research in Advent Technology,* vol. 7, no. 3, pp. 1498-1502, 2019.

[60] N. Mishra and S. Pandya, "Internet of things applications, security challenges, attacks, intrusion detection, and future visions: A systematic review," *IEEE Access*, vol. 9, pp. 59353-59377, 2021.

[61] T. Sasi, A. H. Lashkari, R. Lu, P. Xiong and S. Lqbal, "A comprehensive survey on IoT attacks: Taxonomy, detection mechanisms and challenges," *Journal of Information and Intelligence*, 2023.

[62] L. Gyongyo, "Heimdal Security Blog," 13 September 2023. [Online]. Available: https://heimdalsecurity.com/blog/remote-code-execution-rce/.

[63] M. Msgna, "Anatomy of attacks on IoT systems: Review of attacks, impacts and countermeasures," *Journal of Surveillance, Security and Safety*, vol. 3, no. 4, 2022.

[64] Unit 42, "2020 unit 42 IoT threat report," 2020.

[65] T. Sakhardande, "Redfox Security," [Online]. Available: https://redfoxsec.com/blog/xml-external-entity-injectionsxxe-attacks/.

[66] H. A. Noman and O. M. Abu-Sharkh, "Code injection attacks in wireless-based internet of things (IoT): A comprehensive review and practical implementations," *Sensors*, vol. 23, no. 13, 2023.

[67] M. Burhan, R. A. Rehman, B. Khan and B.-S. Kim, "IoT elements, layered architectures and security issues: A comprehensive survey," *Sensors*, vol. 18, no. 9, 2018.

[68] A. Wood, J. Stankovic and S. Son, "Jam: A jammed-area mapping service for Sensor Networks," *Proceedings. 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748).*

[69] K. Bu, M. Xu, X. Liu, J. Luo, S. Zhang and M. Weng, "Deterministic detection of cloning attacks for anonymous RFID systems," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1255-1266, 2015.

[70] M. V. Bharathi, R. C. Tanguturi, C. Jayakumar and K. Selvamani, "Node capture attack in wireless sensor network: A survey," *2012 IEEE International Conference on Computational Intelligence and Computing Research*, 2012.

[71] D. Puthal, S. Nepal, R. Ranjan and J. Chen, "Threats to networking cloud and edge datacenters in the internet of things," *IEEE Cloud Computing*, vol. 3, no. 3, pp. 64-71, 2016.

[72] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701-716, 2005.

[73] S. Golestani Najafabadi, H. Naji and A. Mahani, "Sybil attack detection: Improving security of WSNS for Smart Power Grid Application," *2013 Smart Grid Conference (SGC)*, 2013.

[74] G. S and G. B.B., "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art," *Int J Syst Assur Eng Manag*, vol. 8, no. 1, pp. 512-530, 2017.

[75] D. D., "Improved Layered Architecture for Internet of Things," *Int. J. Comput. Acad. Res (IJCAR)*, vol. 4, pp. 214-223, 2015.

[76] J. R. Nurse, A. Erola, L. Agrafiotis, M. Goldsmith and S. Creese, "Smart insiders: Exploring the threat from insiders using the internet-of-things," *2015 International Workshop on Secure Internet of Things (SIoT)*, 2015.

[77] B. Stojanović, J. Božić, K. Hofer-SchmiBrankatz, K. Nahrgang, A. Weber, A. Badii, M. Sundaram, E. Jordan and J. Runevic, "Follow the trail: Machine learning for fraud detection in Fintech applications," *Sensors*, vol. 21, no. 5, 2021.

[78] M. Hasan, M. M. Islam, M. I. Zarif and M. Hashem, "Attack and anomaly detection in IOT sensors in IoT sites using machine learning approaches," *Internet of Things*, vol. 7, 2019.

[79] A. Haldorai, A. Ramu and M. Suriya, "Organization internet of things (IoTs): Supervised, unsupervised, and reinforcement learning," *Business Intelligence for Enterprise Internet of Things*, pp. 27-53, 2020.

[80] S. Rathore and J. H. Park, "Semi-supervised Learning Based Distributed Attack Detection Framework for IoT," *Applied Soft Computing*, vol. 72, pp. 79-89, 2018.

[81] M. Lopez-Martin, B. Carro and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," *Expert Systems with Applications*, p. 141, 2020.

[82] K. Yang, J. Ren, Y. Zhu and W. Zhang, "Active learning for wireless IoT intrusion detection," *IEEE Wireless Communications*, vol. 25, no. 6, pp. 19-25, 2018.

[83] B. Bhati and C. S. Rai, "Analysis of support vector machine-based intrusion detection techniques," *Arabian Journal for Science and Engineering*, vol. 45, no. 4, pp. 2371-2383, 2019.

[84] M. M. Aboelwafa, K. G. Seddik, M. H. Eldefrawy, Y. Gadallah and M. Gidlund, "A machine-learning-based technique for false data injection attacks detection in industrial IoT," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8462-8471, 2020.

[85] L. Liu, J. Yang and W. Meng, "Detecting malicious nodes via gradient descent and support vector machine in internet of things," *Computers & Electrical Engineering*, vol. 77, pp. 339-353, 2019.

[86] M. B. Farukee, M. S. Shabit, M. R. Haque and A. H. Sattar, "DDoS attack detection in IoT networks using deep learning models combined with Random Forest as feature selector," *Communications in Computer and Information Science*, pp. 118-134, 2021.

[87] P. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Computing Surveys,* vol. 51, no. 3, pp. 1-36, 2018.

[88] N. Sahar, R. Mishra and S. Kalam, "Deep learning approach-based network intrusion detection system for fog-assisted IoT," *Proceedings of International Conference on Big Data, Machine Learning and their Applications,* pp. 39-50, 2020.

[89] O. Jullian, B. Otero, E. Rodriguez, N. Gutierrez, H. Antona and R. Canal, "Deep-learning based detection for cyber-attacks in IoT Networks: A distributed attack detection framework," *Journal of Network and Systems Management,* vol. 31, no. 2, 2023.

[90] M. Bagaa, T. Taleb, J. Bernabe and A. Skarmeta, "A Machine Learning Security Framework for IOT systems," *IEEE Access,* vol. 8, pp. 114066-114077, 2020.

[91] W. L. Khedr, A. E. Gouda and E. R. Mohamed, "FMDADM: A multi-layer ddos attack detection and mitigation framework using Machine Learning for stateful SDN-based IOT Networks," *IEEE Access,* vol. 11, pp. 28934-28954, 2023.

[92] L. Nie, W. Sun, S. Wang, N. Ning, J. Rodrigues, Y. Wu and S. Li, "Intrusion detection in green internet of things: A deep deterministic policy gradient-based algorithm," *IEEE Transactions on Green Communications and Networking,* vol. 5, no. 2, pp. 778-788, 2021.

[93] M. Anwer, S. M. Khan, M. U. Farooq and W. Waseemullah, "Attack detection in IoT using machine learning," *Engineering, Technology & Applied Science Research,* vol. 11, no. 3, pp. 7273-7278, 2021.

[94] S. M. Nizamudeen, "Intelligent intrusion detection framework for multi-clouds – IoT environment using swarm-based deep learning classifier," *Journal of Cloud Computing,* 2023.

[95] A. Nagisetty and G. P. Gupta, "Framework for detection of malicious activities in IOT networks using Keras Deep Learning Library," *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC),* 2019.

[96] O. Alkadi, N. Moustafa, B. Turnbull and K.-K. R. Choo, "A deep blockchain framework-enabled collaborative intrusion detection for protecting IoT and cloud networks," *IEEE Internet of Things Journal,* vol. 8, no. 12, pp. 9463-9472, 2021.

[97] D. K. Reddy, H. S. Behera, J. Nayak, P. Vijayakumar, B. Naik and P. K. Singh, "Deep neural network based anomaly detection in internet of things network traffic tracking for the applications of future Smart Cities," *Transactions on Emerging Telecommunications Technologies,* vol. 32, no. 7, 2020.

Annexes

Annex-A: Attack Category Detail Data Collection Status

S. No	Attack Category	Attacker Scripts	Captured pcap size	Remark
0	benign	benign_network	360MB	
1	ddos	ddos_ack_fragmentation.py	217MB	
2	ddos	ddos_icmp_flood.py	288MB	
3	ddos	ddos_icmp_fragmentation.py	390MB	
4	ddos	ddos_pshack_flood.py	155MB	
5	ddos	ddos_rstfin_flood.py	148MB	
6	ddos	ddos_syn_flood.py	155MB	
7	ddos	ddos_synonymousip_flood.py	143MB	
8	ddos	ddos_tcp_flood.py	150MB	
9	ddos	ddos_udp_flood.py	148MB	
10	ddos	ddos_udp_fragmentation.py	203MB	
11	dos	dos_syn_flood.py	23MB	
12	dos	dos_tcp_flood.py	21MB	
13	dos	dos_udp_flood.py	155MB	
14	mirai	mirai_greeth_flood.py	190MB	
15	mirai	mirai_greip_flood.py	174MB	
16	mirai	mirai_udp_plain_flood.py	1.57GB	
17	recon	recon_port_scan.py	4.8MB	

Key Data Collection Assumptions

- Total number of classes which have captured pcap data: 18 attack classes
- Total size of collected data: 60GB
- Time duration to collect each data: 1 hour

Annex-B: IoT Device MQTT Messaging Rules

smart_plug

- subscribe(zigbee2mqtt/smart_plug/set)
- commands [ON, OFF]
- publish(zigbee2mqtt/smart_plug/status), (zigbee2mqtt/smart_plug/power_usage)

smart_speaker

- subscribe(zigbee2mqtt/smart_speaker/set)
- commands [PLAY_MUSIC, STOP_MUSIC]
- publish(zigbee2mqtt/smart_speaker/status)

smart_thermostat

- subscribe(zigbee2mqtt/smart_thermostat/set)
- commands [SET_TEMP x]
- publish(zigbee2mqtt/smart_thermostat/status)

smart_bulb

- subscribe(zigbee2mqtt/smart_bulb/set)
- commands [ON, OFF]
- publish(zigbee2mqtt/smart_bulb/status)

smart_camera

- subscribe(zigbee2mqtt/smart_camera/set)
- commands [START_RECORDING, STOP_RECORDING]
- publish(zigbee2mqtt/smart_camera/status)

smart_fridge

- subscribe(zigbee2mqtt/smart_fridge/set)
- commands [OPEN_DOOR, CLOSE_DOOR]
- publish(zigbee2mqtt/smart_fridge/status)

smart_locker

- subscribe(zigbee2mqtt/smart_locker/set)
- commands [LOCK, UNLOCK]
- publish(zigbee2mqtt/smart_locker/status)

Annex-C: Attack categories with Sub Attack classes

Attack Category	Sub-Attacks	Descriptions
BruteForce Attacks	DictionaryBruteForce	
DDoS Attacks	DDoS-ICMP_Flood	
	DDoS-UDP_Flood	
	DDoS-TCP_Flood	
	DDoS-PSHACK_Flood	
	DDoS-SYN_Flood	
	DDoS-RSTFINFlood	
	DDoS-SynonymousIP_Flood	
	DDoS-ICMP_Fragmentation	
	DDoS-UDP_Fragmentation	
	DDoS-ACK_Fragmentation	
	DDoS-HTTP_Flood	
	DDoS-SlowLoris	
DoS Attacks	DoS-UDP_Flood	
	DoS-TCP_Flood	
	DoS-SYN_Flood	
	DoS-HTTP_Flood	
'Mirai' Attacks	Mirai-greeth_flood	
	Mirai-udpplain	

	Mirai-greip_flood
Recon Attacks	Recon-HostDiscovery
	Recon-OSScan
	Recon-PortScan
	VulnerabilityScan
	Recon-PingSweep
Spoofing Attacks	MITM-ArpSpoofing
	DNS_Spoofing
Web Attacks	BrowserHijacking
	CommandInjection
	SqlInjection
	XSS
	Backdoor_Malware
	Uploading_Attack

Annex-D: Emulator Setup Configuration: Docker Compose

```
#name: iot_simulator
services:
  mosquitto_service:
    image: eclipse-mosquitto: latest
    container_name: mosquitto_container
    ports:
      - "1883:1883” MQTT port
      - "9001:9001” WebSockets (optional)
    volumes:
      - mosquitto_data:/mosquitto/data
      - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
      - mosquitto_log:/mosquitto/log
    networks:
      iot_network:
        ipv4_address: 172.16.1.3

  smart_bulb_service:
    build:
      context:/iot_devices/smart_bulb  # The directory containing Dockerfile.smart_bulb
      dockerfile: Dockerfile.smart_bulb # If the default name for the file is not Dockerfile, it
should be specified explicitly
    image: smart-bulb-image # Custom image name
    container_name: smart_bulb_container
    depends_on:
      - mosquitto_service
      - router
    environment:
      - MQTT_SERVER=mqtt://mosquitto_service:1883
#       - MQTT_BROKER=mosquitto_service
#       - MQTT_PORT=1883
      - MQTT_TOPIC=zigbee2mqtt/smart_bulb/set
    networks:
      iot_network:
        ipv4_address: 172.16.1.10
    cap_ad:
      - NET_ADMIN
    command: >
      bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
               python3 /app/smart_bulb_emulator.py &&
               bash"

  smart_camera_service:
    build:
      context:/iot_devices/smart_camera
```

```
    dockerfile: Dockerfile.smart_camera
  image: smart-camera-image
  container_name: smart_camera_container
  depends_on:
    - mosquitto_service
  environment:
    - MQTT_SERVER=mqtt://mosquitto_service:1883
    - MQTT_TOPIC=zigbee2mqtt/smart_camera/set
  networks:
    iot_network:
      ipv4_address: 172.16.1.11
  cap_ad:
    - NET_ADMIN
  command: >
    bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
             python3 /usr/src/app/smart_camera_emulator.py &&
             bash"

smart_fridge_service:
  build:
    context:/iot_devices/smart_fridge
    dockerfile: Dockerfile.smart_fridge
  container_name: smart_fridge_container
  image: smart-fridge-image
  depends_on:
    - mosquitto_service
  environment:
    - MQTT_SERVER=mqtt://mosquitto_service:1883
    - MQTT_TOPIC=zigbee2mqtt/smart_fridge/set
  networks:
    iot_network:
      ipv4_address: 172.16.1.12
  cap_ad:
    - NET_ADMIN
  command: >
    bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
             python3 /usr/src/app/smart_fridge_emulator.py &&
             bash"

smart_locker_service:
  build:
    context:/iot_devices/smart_locker
    dockerfile: Dockerfile.smart_locker
  container_name: smart_locker_container
  image: smart-locker-image
  depends_on:
```

```
    - mosquitto_service
  environment:
    - MQTT_SERVER=mqtt://mosquitto_service:1883
    - MQTT_TOPIC=zigbee2mqtt/smart_lock/set
  networks:
    iot_network:
      ipv4_address: 172.16.1.13
  cap_ad:
    - NET_ADMIN
  command: >
    bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
             python3 /usr/src/app/smart_locker_emulator.py &&
             bash"

smart_plug_service:
  build:
    context: ./iot_devices/smart_plug
    dockerfile: Dockerfile.smart_plug
  image: smart-plug-image
  container_name: smart_plug_container
  depends_on:
    - mosquitto_service
  environment:
    - MQTT_SERVER=mqtt://mosquitto_service:1883
    - MQTT_TOPIC=zigbee2mqtt/smart_plug/set
  networks:
    iot_network:
      ipv4_address: 172.16.1.14
  cap_add:
    - NET_ADMIN
  command: >
    bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
             python3 /usr/src/app/smart_plug_emulator.py &&
             bash"

smart_speaker_service:
  build:
    context: ./iot_devices/smart_speaker
    dockerfile: Dockerfile.smart_speaker
  image: smart-speaker-image
  container_name: smart_speaker_container
  depends_on:
    - mosquitto_service
  environment:
    - MQTT_SERVER=mqtt://mosquitto_service:1883
    - MQTT_TOPIC=zigbee2mqtt/smart_speaker/set
```

```
  networks:
    iot_network:
      ipv4_address: 172.16.1.15
  cap_add:
    - NET_ADMIN
  command: >
    bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
             python3 /usr/src/app/smart_speaker_emulator.py &&
             bash"

smart_thermostat_service:
  build:
    context: ./iot_devices/smart_thermostat
    dockerfile: Dockerfile.smart_thermostat
  image: smart-thermostat-image
  container_name: smart_thermostat_container
  depends_on:
    - mosquitto_service
  environment:
    - MQTT_SERVER=mqtt://mosquitto_service:1883
    - MQTT_TOPIC=zigbee2mqtt/smart_thermostat/set
  networks:
    iot_network:
      ipv4_address: 172.16.1.16
  cap_add:
    - NET_ADMIN
  command: >
    bash -c "ip route add 172.16.2.0/24 via 172.16.1.2 dev eth0 &&
             python3 /usr/src/app/smart_thermostat_emulator.py &&
             bash"

gns3_service:
  build:
    context: ./gns3
    dockerfile: Dockerfile.gns3
  image: gns3-simulator-image
  container_name: gns3_simulator_container
  environment:
    - USERNAME=ubuntu
    - PASSWORD=ubuntu
    - DISPLAY  # Inherits the DISPLAY environment variable from the host.
  networks:
    iot_network:
      ipv4_address: 172.16.1.4
    external_network:
      ipv4_address: 172.16.2.4
```

```
    ports:
      - "3080:3080"  # Port for GNS3
    volumes:
      - /tmp/.X11-unix:/tmp/.X11-unix  # Allows GUI applications to connect to the host's X server.
      - gns3_data:/home/ubuntu/gns3_data  # Persistent storage for GNS3 data.
      - /run/user/$(id -u)/at-spi:/run/user/$(id -u)/at-spi  # Mounts the at-spi directory for GUI apps.
      - /var/run/docker.sock:/var/run/docker.sock  # Mounting this Docker socket inside gns3 container allows access to host's docker socket so that other docker containers can be seen by the gns docker
    devices:
      - "/dev/kvm:/dev/kvm"  # Provides access to host devices like KVM if required
    #network_mode: "host"
    privileged: true  # Allows the container to perform Docker operations and access certain system capabilities.
    tty: true  # Keeps the container interactive
    stdin_open: true
#    command: >
#      bash -c "echo $PASSWORD | sudo -S -k libvirtd -d &&
#               echo $PASSWORD | sudo -S service dbus start &&
#               echo $PASSWORD | sudo -S service docker start &&
#               bash"
    command: >
      bash -c "sudo libvirtd -d &&
               sudo service dbus start &&
               sudo service docker start &&
               bash"

  ddos_attacker_module_service:
    build:
      context: ./attacker_module
      dockerfile: Dockerfile.raspberrypi4
      platforms:
        - linux/amd64
    image: sisayz/ddos-attacker-image
    cap_add:  # Give the container the necessary privileges for raw socket access.
      - NET_RAW
      - NET_ADMIN
    container_name: ddos_attacker_container
    privileged: true  # Grants the container more extensive access (not secure like cap_add)
    depends_on:
      - mosquitto_service
      - router
    stdin_open: true  # Keep STDIN open even if not attached
    tty: true  # Allocate a pseudo-TTY
```

```
    volumes:
      - ./attacker_module/attack-scripts/ddos:/usr/src/app/ddos
      - ./pcap_data/ddos:/usr/src/app/pcap_data/ddos  # Mount for storing the pcap files
    networks:
      #iot_network:
        #ipv4_address: 172.16.2.80
      external_network:
        ipv4_address: 172.16.2.10
    #command: python3 /usr/src/app/attack_launcher.py  # This specification overrides the
CMD of Dockerfile
    # command: >
      # bash -c "tcpdump -i eth0 -w /usr/src/app/pcap_data/ddos_capture.pcap &
      # python3 ddos_http_flood.py http://<target_device_ip>"  # Capture packets and run the
attack script
    command: >
      bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
               python3 attack_launcher.py &&
               bash"

  dos_attacker_module_service:
    build:
      context: ./attacker_module
      dockerfile: Dockerfile.raspberrypi4
      platforms:
        - linux/amd64
    image: dos-attacker-image
    container_name: dos_attacker_container
    privileged: true
    depends_on:
      - mosquitto_service
      - router
    stdin_open: true
    tty: true
    volumes:
      - ./attacker_module/attack-scripts/dos:/usr/src/app/dos
      - ./pcap_data/dos:/usr/src/app/pcap_data/dos
    networks:
      external_network:
        ipv4_address: 172.16.2.11
    command: >
      bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
               python3 attack_launcher.py &&
               bash"

  recon_attacker_module_service:
    build:
```

```
    context: ./attacker_module
    dockerfile: Dockerfile.raspberrypi4
    platforms:
      - linux/amd64
  image: recon-attacker-image
  container_name: recon_attacker_container
  privileged: true
  depends_on:
    - mosquitto_service
    - router
  stdin_open: true
  tty: true
  volumes:
    - ./attacker_module/attack-scripts/recon:/usr/src/app/recon
    - ./pcap_data/recon:/usr/src/app/pcap_data/recon
  networks:
    external_network:
      ipv4_address: 172.16.2.12
  command: >
    bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
             python3 attack_launcher.py &&
             bash"

web_based_attacker_module_service:
  build:
    context: ./attacker_module
    dockerfile: Dockerfile.raspberrypi4
    platforms:
      - linux/amd64
  image: web-based-attacker-image
  container_name: web_based_attacker_container
  privileged: true
  depends_on:
    - mosquitto_service
    - router
  stdin_open: true
  tty: true
  volumes:
    - ./attacker_module/attack-scripts/web-based:/usr/src/app/web-based
    - ./pcap_data/web-based:/usr/src/app/pcap_data/web-based
  networks:
    external_network:
      ipv4_address: 172.16.2.13
  command: >
    bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
             python3 attack_launcher.py &&
```

```
            bash"

  brute_force_attacker_module_service:
    build:
      context: ./attacker_module
      dockerfile: Dockerfile.raspberrypi4
      platforms:
        - linux/amd64
    image: brute-force-attacker-image
    container_name: brute_force_attacker_container
    privileged: true
    depends_on:
      - mosquitto_service
      - router
    stdin_open: true
    tty: true
    volumes:
      - ./attacker_module/attack-scripts/brute_force:/usr/src/app/brute_force
      - ./pcap_data/brute_force:/usr/src/app/pcap_data/brute_force
    networks:
      external_network:
        ipv4_address: 172.16.2.14
    command: >
      bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
               python3 attack_launcher.py &&
               bash"

  spoofing_attacker_module_service:
    build:
      context: ./attacker_module
      dockerfile: Dockerfile.raspberrypi4
      platforms:
        - linux/amd64
    image: spoofing-attacker-image
    container_name: spoofing_attacker_container
    privileged: true
    depends_on:
      - mosquitto_service
      - router
    stdin_open: true
    tty: true
    volumes:
      - ./attacker_module/attack-scripts/spoofing:/usr/src/app/spoofing
      - ./pcap_data/spoofing:/usr/src/app/pcap_data/spoofing
    networks:
      external_network:
```

```
      ipv4_address: 172.16.2.15
  command: >
    bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
             python3 attack_launcher.py &&
             bash"

mirai_attacker_module_service:
  build:
    context: ./attacker_module
    dockerfile: Dockerfile.raspberrypi4
    platforms:
      - linux/amd64
  image: mirai-attacker-image
  container_name: mirai_attacker_container
  privileged: true
  depends_on:
    - mosquitto_service
    - router
  stdin_open: true
  tty: true
  volumes:
    - ./attacker_module/attack-scripts/mirai:/usr/src/app/mirai
    - ./pcap_data/mirai:/usr/src/app/pcap_data/mirai
  networks:
    external_network:
      ipv4_address: 172.16.2.16
  command: >
    bash -c "ip route add 172.16.1.0/24 via 172.16.2.2 dev eth0 &&
             python3 attack_launcher.py
             bash"

# Bridge container that connects both networks
router:
  build:
    context: ./router
    dockerfile: Dockerfile.router
  image: router-image
  container_name: router_container
  privileged: true
  cap_add:
    - NET_ADMIN
  #stdin_open: true
  #tty: true
  networks:
    iot_network:
      ipv4_address: 172.16.1.2
```

```
      external_network:
        ipv4_address: 172.16.2.2
    volumes:
      - ./pcap_data:/usr/src/data/pcap
    command: >
      sh -c "iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE &&
             tail -f /dev/null"  # Keeps the container alive without consuming significant
resources.

networks:
  iot_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.1.0/24
          gateway: 172.16.1.1
  external_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.2.0/24
          gateway: 172.16.2.1

volumes:
  mosquitto_data:
  mosquitto_log:
  gns3_data:
  pcap_data:
```

Annex-E: Dataset Feature Attributes with Data Type and descriptions

#	Column	Dtype	Descriptions
1	timestamp	float64	The timestamp to process the packets
2	src_mac	object	Identify the source of mac address attack origin
3	dst_mac	object	Identify mac address of target attack
4	src_ip	object	Identify source IP address of attack origin
5	dst_ip	object	Identify destination IP address of target
6	protocol_type	float64	UDP, HTTP, HTTPS, ICMP, or unknown
7	fin_flag	int64	Indicate connection is being torn down
8	syn_flag	int64	Packet used to initiate a connection
9	rst_flag	int64	Signify connection is down or not accepting request
10	psh_flag	int64	Indicate incoming data directly passed to application instead of getting buffered
11	ack_flag	int64	Used to confirm packet data is received
12	urg_flag	int64	Indicate packet data processed immediately by TCP stack
13	src_port	int64	Identify the source port of attack
14	dst_port	int64	Identify destination port of the target
15	ICMP	int64	Identify if network layer protocol is ICMP
16	Other	int64	
17	TCP	int64	Indicate transport layer protocol is TCP
18	UDP	int64	Indicate transport layer protocol is UDP
19	encoded_label	int64	
20	header_length	float64	Indicate the header level values
21	Ttl	float64	The duration of time-to-live
22	duration	float64	Indicates duration of packet flow
23	sum_packet_size	float64	Total packet size in the flow
24	max_packet_size	float64	Maximum packet size in the flow
25	avg_packet_size	float64	Average packet size in the flow
26	std_packet_size	float64	Standard deviation of packet size in the flow
27	min_duration	float64	Minimum packet length in the flow
28	max_duration	float64	Maximum packet length in the flow
29	sum_duration	float64	Total packet length
30	avg_duration	float64	Average packet length in the flow
31	std_duration	float64	Standard deviation of packet length in the flow
32	fin_count	float64	Number of packets with fin flag set in same flow
33	syn_count	float64	Number of packets with syn flag set in same flow
34	rst_count	float64	Number of packets with rst flag set in same flow
35	psh_count	float64	Number of packets with psh flag set in same flow
36	ack_count	float64	Number of packets with ack flag set in same flow
37	urg_count	float64	Number of packets with urg flag set in same flow
38	tcp_count	float64	Number of TCP count in the packet flow
39	udp_count	float64	Number of UDP count in the packet flow
40	icmp_count	float64	Number of ICMP count in the packet flow
41	other_count	float64	

Annex-F: Model Configuration Setup Sample Python Code

A. Configuration Setup

```
import os

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__))) # Calling twice makes the base dir to locate one level up
BASE_DIR = os.path.join(BASE_DIR, 'data')

MODEL_DIR = os.path.join(BASE_DIR, 'dl_model')
DATASET_DIR = os.path.join(BASE_DIR, 'dataset')
PCAP_FILE_DIR = os.path.join(BASE_DIR, 'pcap_data')
UNPROCESSED_CSV_DIR = os.path.join(BASE_DIR, 'csv_data')
SAMPLE_DATASET_DIR = os.path.join(BASE_DIR, 'sample_dataset')
LOG_DIR = os.path.join(BASE_DIR, 'logs')
MODEL_RESULT_DIR = os.path.join(MODEL_DIR, 'model_result')
SEARCH_RESULT_DIR = os.path.join(MODEL_DIR, 'search_result')
GRAPHS_RESULT_DIR = os.path.join(MODEL_DIR, 'graph')

ORIGINAL_DATASET_PATH = os.path.join(DATASET_DIR, 'dataset_original.csv')
BEST_DATASET_PATH = os.path.join(DATASET_DIR, 'dataset_ws_200_ss.csv')
BEST_TRAINED_MODEL_PATH = os.path.join(MODEL_RESULT_DIR, 'best_model_trained.h5')
REPORT_PATH_FOR_BEST_MODEL = os.path.join(MODEL_RESULT_DIR, 'best_model_report.log')
BEST_SCALER_PATH = os.path.join(MODEL_RESULT_DIR, 'ss_for_dataset_ws_200.pkl')
LABEL_ENCODER_PATH = os.path.join(MODEL_RESULT_DIR, 'label_encoder.pkl')

RAW_FEATURES = ['timestamp', 'header_length', 'src_mac','dst_mac', 'src_ip', 'dst_ip',
        'protocol_type', 'ttl', 'fin_flag', 'syn_flag', 'rst_flag', 'psh_flag',
        'ack_flag', 'ece_flag', 'cwr_flag', 'urg_flag', 'src_port', 'dst_port',
        'protocol', 'ipv', 'llc']

DEVICE_SPECIFIC_FEATURES = ['timestamp', 'src_mac', 'dst_mac', 'src_ip', 'dst_ip',
            'src_port', 'dst_port']  # Needed later for attribution

NON_SCALE_FEATURES = ['timestamp', 'encoded_label', 'protocol_type', 'fin_flag',
            'syn_flag', 'rst_flag', 'psh_flag', 'ack_flag', 'urg_flag',
            'src_port', 'dst_port', 'ICMP', 'Other', 'TCP', 'UDP',]

TRAIN_FEATURES = ['protocol_type', 'fin_flag', 'syn_flag', 'rst_flag', 'psh_flag',
            'ack_flag', 'urg_flag', 'ICMP', 'Other', 'TCP', 'UDP', 'header_length',
            'ttl', 'duration', 'sum_packet_size', 'max_packet_size',
            'avg_packet_size', 'std_packet_size', 'min_duration', 'max_duration',
            'sum_duration', 'avg_duration', 'std_duration', 'fin_count',
```

```
            'syn_count', 'rst_count', 'psh_count', 'ack_count', 'urg_count',
            'tcp_count', 'udp_count', 'icmp_count', 'other_count']

SCALE_FEATURES = ['header_length', 'ttl', 'duration', 'sum_packet_size',
'max_packet_size',
            'avg_packet_size', 'std_packet_size', 'min_duration', 'max_duration',
            'sum_duration', 'avg_duration', 'std_duration', 'fin_count', 'syn_count',
            'rst_count', 'psh_count', 'ack_count', 'urg_count', 'tcp_count',
            'udp_count', 'icmp_count', 'other_count']

WINDOW_SIZE = [5, 10, 20, 50, 100, 200]

class Params:
    FILTERS_L1 = 128
    KERNEL_SIZE_L1 = 7
    POOL_SIZE_L2 = 2
    FILTERS_L3 = 128
    KERNEL_SIZE_L3 = 1
    POOL_SIZE_L4 = 2
    DENSE_UNITS_L5 = 64
    HIDDEN_LAYER_ACTIVATION = ['relu', 'tanh']
    OUTPUT_LAYER_ACTIVATION = 'softmax'
    OPTIMIZER = ['adam', 'rmsprop', 'sgd', 'nadam', 'adadelta', 'adagrad']
    TUNER_EPOCHS = 10
    TRAIN_EPOCHS = 25
    TUNER_BATCH_SIZE = 8
    TRAIN_BATCH_SIZE = 32
    LEARNING_RATE = 0.0001
    UNITS = 128
    NUM_HEADS = 6
    KEY_DIM = 96
    DROPOUT = 0.3
    TUNER_SAMPLE_SIZE = 2_500 # 0 for full dataset size
    TRAIN_SAMPLE_SIZE = 1_000_000
```

B. FFNN Model Generation

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, BatchNormalization
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Nadam, Adadelta,
Adagrad
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
from tensorflow.keras.metrics import AUC
import tensorflow as tf
```

```
from utils.logger import get_logger
from utils.config import LOG_DIR
from dl.preprocessor_utils import measure_duration
from dl.model_builder.custom_metrics import F1Score, WeightedF1Score

build_fnn_model(hp, input_shape, num_classes, tuned_params=None):
    input_shape = (input_shape[1],)
    logger.info(f'Running FNN model builder with input_shape: {input_shape} and
num_classes: {num_classes}')
    model = Sequential()
    model.add(Input(shape=input_shape))

    optimizer = {
            'adam': Adam(learning_rate=hp.Choice('adam_lr', [0.001, 0.0001, 0.01]),
clipvalue=1.0),
            # 'adam': Adam(learning_rate=lr_schedule), }[optimizer_choice]
    # Add Dense layers dynamically
    for i in range(num_dense_layers):
        model.add(Dense(
            units=dense_units,
            activation=dense_activation,
            kernel_regularizer=kernel_regularizer
        ))
        model.add(Dropout(rate=dense_dropout_rate))
        model.add(BatchNormalization(dtype='float32'))
    # Add output layer
    model.add(Dense(num_classes, activation='softmax', dtype='float32'))
    # Compile the model
    model.compile(
        optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=[ 'accuracy',AUC(name='auc', multi_label=True) ] )
    # model.summary()
    return model
```

C. Hybrid Model Generation

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv1D, MaxPooling1D, Flatten, Dense, LSTM, Dropout, Reshape, Input,
    BatchNormalization, GlobalAveragePooling1D, Bidirectional, Layer
)
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Nadam, Adadelta,
Adagrad
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.metrics import AUC
import tensorflow as tf
```

```
from utils.logger import get_logger
from utils.config import LOG_DIR
from dl.preprocessor_utils import measure_duration
from dl.model_builder.custom_metrics import F1Score

def build_cnn_lstm_model(hp, input_shape, num_classes, tuned_params=None):
   input_shape = (input_shape[1], 1)
   logger.info(f'Running CNN_LSTM hybrid model builder with input_shape:
input_shape} and num_classes: {num_classes}')
       conv_activation= tuned_params['conv_activation']
       pool_size = tuned_params['pool_size']
       cnn_dropout = tuned_params['cnn_dropout']
       lstm_units = tuned_params['lstm_units']
       lstm_activation = tuned_params['lstm_activation']
       lstm_dropout = tuned_params['lstm_dropout']
       num_dense_layers = tuned_params['num_dense_layers']
       dense_units = tuned_params['dense_units']
       dense_activation = tuned_params['dense_activation']
       dense_dropout = tuned_params['dense_dropout']
      )
     optimizer = {
          'adam': Adam(learning_rate=hp.Choice('adam_lr', [0.001, 0.0001,
0.01]) }[optimizer_choice]

   # Define model inputs
   inputs = Input(shape=input_shape)
   # CNN Block with multiple Conv1D layers
   x = inputs
   for i in range(num_cnn_layers):
      x = Conv1D(
         filters=conv_filters,
         kernel_size=kernel_size,
         activation=conv_activation,
         padding='same'
      )(x)
      x = BatchNormalization()(x)
      x = MaxPooling1D(pool_size=pool_size, padding="same")(x)
      x = Dropout(rate=cnn_dropout)(x)
   # Global Average Pooling instead of Flatten
   x = GlobalAveragePooling1D()(x)
   # Reshape for LSTM
   x = Reshape((1, -1))(x)  # Adjust shape for LSTM compatibility
   # LSTM Block with Bidirectional LSTM
   x = Bidirectional(LSTM(
      units=lstm_units,
```

```
    return_sequences=True,
    activation=lstm_activation
))(x)
x = Dropout(rate=lstm_dropout)(x)
# Attention Mechanism
x = CustomAttentionLayer()([x, x, x])  # Query, Key, Value passed as same tensor
# Flatten attention output to feed Dense layers
x = Flatten()(x)
# Dense Layer(s)
for i in range(num_dense_layers):
    x = Dense(
        units=dense_units,
        activation=dense_activation,
        kernel_regularizer=kernel_regularizer
    )(x)
    x = Dropout(rate=dense_dropout)(x)
# Output Layer
outputs = Dense(num_classes, activation='softmax')(x)
# Define Model
model = tf.keras.Model(inputs, outputs)
# Compile the model
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy', AUC(name='auc', multi_label=True) ])
# model.summary()
Return Model
```

D. LSTM Model Generation

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Input, Dense, Dropout, LSTM, Bidirectional

from tensorflow.keras.optimizers import SGD, Adam

from tensorflow.keras.callbacks import ReduceLROnPlateau

from dl.model_builder.custom_metrics import F1Score, WeightedF1Score

Def build_lstm_model(hp, input_shape, num_classes, tuned_params=None):

    input_shape = (input_shape[1], input_shape[2])

   logger.info(f"Running LSTM model builder with input_shape: {input_shape}, and
num_class: {num_classes}")

   model = Sequential()

   model.add(Input(shape=input_shape))
```

```
    optimizer = {'adam': Adam(learning_rate=hp.Choice('adam_lr', [0.001, 0.0001, 0.01]), clipnorm=1.0), }[optimizer_choice]

    # Add LSTM layers dynamically

    for i in range(num_lstm_layers):

        return_sequences =  i < (num_lstm_layers - 1)  # Only return sequences for all but the last LSTM layer (short for  True if i < (num_lstm_layers - 1) else False)

        model.add(Bidirectional(LSTM(

            units=lstm_units,

            activation=lstm_activation,

            return_sequences=return_sequences,

            dropout=lstm_dropout_rate,

            recurrent_dropout=recurrent_dropout )))

        # model.add(Dropout(rate=lstm_dropout_rate))

    # Add Dense layers dynamically

    for i in range(num_dense_layers):

        model.add(Dense(

            units=dense_units,

            activation=dense_activation

        ))

        model.add(Dropout(rate=dense_dropout_rate))

    # Add output layer

    model.add(Dense(num_classes, activation='softmax'))

    # Compile the model

    model.compile(

        optimizer=optimizer,

        loss='sparse_categorical_crossentropy',

        metrics=[ 'accuracy', AUC(name='auc', multi_label=True) ] )

    # model.summary()

    Return Model
```

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all sources of materials used for the thesis have been duly acknowledged.

Declared by:

Name: Yonas Mekonnen _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: Mesfin Kifle (PhD) _____

Signature: _____

Date: _____