ENHANCED SECURITY MONITORING IN INTERNET OF THINGS SYSTEMS THROUGH INTRUSION DETECTION MODEL

JOSEPH GITONGA IMATHIU

A Thesis Submitted in Partial Fulfilment of the Requirement for Conferment of the Master of Science in Information Technology of Meru University of Science and Technology

2025

DECLARATION

This thesis is my original work and has not been presented for a degree award in any other institution.

CT402/202192/21

Signed……...……. Date……...…….....

Joseph Gitonga Imathiu

DECLARATION BY UNIVERSITY SUPERVISORS

This thesis has been submitted with our approval as University Supervisors.

Signed…… Date……...…….…….

Professor Amos Odhiambo Omamo, Ph.D.

Meru University of Science & Technology

Signed...…….... Date…….…….…….…….

Dr. Amos Chege Kirongo, Ph.D.

Meru University of Science & Technology

ACKNOWLEDGEMENT

In this work I acknowledge God for His Mercies and Grace, my supervisors Prof. Amos Omamo, PhD, Dr. Amos Chege, PhD and my family for moral support. Also, greatly appreciating my colleagues in the ICT Department starting with the ICT in-charge and my section team for supporting me when I was so greatly occupied in this research work. Also greatly appreciated is Mr. Maina from the Marambi Library for lending his hand in editing, formatting and plagiarism checking of this Thesis report.

DEDICATION

I dedicate this work to my family. My wife Jane, sons Fidelis, Elvis and Emmanuel for sacrifice and giving me that piece of mind I needed most in this research work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FORMULA

LIST OF APPENDICES

LIST OF ABBREVIATION

DL	Deep Learning
ML	Machine Learning
IOT	Internet of Things
IDS	Intrusion Detection System
DBN	Deep Belief Network
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
RBM	Restricted Boltzmann Machine
GAN	Generative Adversarial Network
ANASTA	Advanced Networked Agents for Security and Trust Assessment
NIDS	Network Intrusion Detection System
MMT	Montimage Intrusion Detection System
LSTM	Long Short–Term Memory
RNN	Recurrent Neural Network
HIDS	Hybrid Intrusion Detection System
SIDS	Signature Intrusion Detection System
AIDS	Anomaly-based Intrusion Detection System
ROC AUC	Receiver Operating Characteristic - Area Under the Curve
FAR	False Alarm Rate
WEKA	Waikato Environment for Knowledge Analysis

DEFINITION OF TERMS

Intrusion	is any unauthorized or malicious activity that attempts to compromise the confidentiality, integrity, or availability of a computer system, network or data.
Dataset	is a structured collection of labeled or unlabeled data that represents both normal and malicious activities within a network or system, used to train, test, and evaluate the performance of intrusion detection models.
Intrusion Detection System (IDS)	is a security mechanism designed to monitor and analyze network or system activity for signs of malicious behavior, unauthorized access, or policy violations.
Machine Learning (ML)	is a branch of artificial intelligence (AI) that focuses on the development of algorithms and models that enable computers to learn from data and make decisions or predictions without being explicitly programmed for every task.
Internet of Things (IoT)	refers to a network of physical devices—such as sensors, machines, appliances, and other objects—that are embedded with software, sensors, and connectivity features, enabling them to collect, exchange, and act on data over the internet or other communication networks.

ABSTRACT

Intrusion detection systems (IDS) play a pivotal role in identifying and mitigating potential threats and vulnerabilities in IoT devices. IoT devices holds crucial data. Internet of Things (IoT) has rapidly evolved into a field that involves the interconnection and interaction of smart objects, which have embedded sensors, on-board data processing capability, and a means of communication, to provide automated services and applications. The proliferation of Internet of Things (IoT) devices usage has exponentially increased the attack surface, necessitating robust security mechanisms. This is because many IoT devices operate with limited computational resources, constrained memory, and low-power capabilities, making them susceptible to security breaches. Additionally, the sheer diversity of device types, communication protocols, and deployment scenarios introduces a complex attack surface, providing malicious actors with numerous entry points to exploit. Traditional security measures have proved insufficient against sophisticated attacks targeting generally IoT ecosystems because of their high demand on resources. Therefore, this research explores on enhancing security for IoTs which are resource constraints with a model optimized for efficiency, minimizing computational overhead and energy consumption by employing an Intrusion Detection System (IDS). The model approach involves designing and developing of an IDS using machine learning algorithms for anomaly detection of the network traffic patterns to identify potential security breaches. The dataset was downloaded from Kaggle after creation of an account and cleaned to get relevant structured data for training and evaluating the model. The model training utilized pro deep neural network architecture and NSL-KDD dataset which was split into training (70%), validation (15%), and testing (15%) subsets using stratified sampling to maintain the class distribution across all subsets. The model's performance was assessed using multiple metrics like accuracy, precision, recall, F1-Score, FAR and Receiver Operating Characteristic Area Under the Curve) providing a comprehensive understanding of its capabilities. The model achieved Accuracy of 97.8%, Precision 0.976, false alarm rate of 1.54, 0.95, F1-score of 96.28% and an overall ROC AUC of 0.97. This results provided insights into the ability to correctly classify network traffic as either normal or malicious, as well as its effectiveness in detecting specific attack types in IoTs.

CHAPTER ONE: INTRODUCTION

1.1. Background to the Study

Intrusion detection is system is used to monitor network assets to detect anomalous behavior and misuse in network. This concept has been around for nearly twenty years but only recently has it seen a dramatic rise in popularity and incorporation into the overall information security infrastructure Betke et al., (2007). Intrusion detection systems is an inevitable processing unit in recent wireless networks due to lack of security and increased number of intruders Dr. S. Smys et al., (2020). Few research models are focused towards multiple attack detection and its computation cost is quite high which makes the system not suitable for wide range of applications. Intrusion detection techniques (IDS) play a pivotal role in identifying and mitigating potential threats and vulnerabilities. These techniques leverage cutting-edge technologies such as machine learning, artificial intelligence, and behavioral analysis to enhance the accuracy and responsiveness of security systems. IDSs can be based on either cross-checking events by comparing them to a database of known intrusion experiences, also known as signature-based, or on learning the system's typical behavior and reporting whether any abnormal occurrences occurred, also known as anomaly-based (Spadaccino & Cuomo, n.d.).

An Intrusion Detection System (IDS) is a tool which monitors, analyzes and detects the abnormalities in the network activities (Anitha & Arockiam, 2019). It detects acts that intruders take against information systems. These operations, known as intrusions, are intended to gain unauthorized access to an IoT system. Intruders might be external or inside. Internal intruders are network users who have some lawful access and attempt to escalate their access privileges in order to abuse non-authorized privileges. External intruders are users outside of the target network who attempt to obtain unauthorized

access to system information. In this context, the objective is to enhance the security of IoT devices and the data they handle, ensuring the integrity, confidentiality, and availability of information in wireless networks (Tabassum et al., 2019). The heterogeneity of IoT ecosystems introduces unique challenges for traditional security approaches (Gyamfi et al., 2022). Advanced intrusion detection techniques aim to overcome these challenges by developing adaptive and context-aware mechanisms capable of discerning normal behavior from potential security breaches in IoT devices. The increasing integration of Internet of Things (IoT) devices has led to a surge in data connectivity through Wireless Local Area Networks (WLANs). IoT has rapidly transformed into a field that refers into the interconnection and interaction of intelligent objects. These objects or devices have built-in sensors, internal data processing capabilities and communication tools to provide automated services and applications (Butun et al., 2020). IoTs are applied in Medical, cities, devices, Smart homes, objects, e-commerce, grids, e-banking, e-government and among other applications (Khan et al., 2022). Internet of Things (IoT) devices work mainly in wireless mediums (Ponnusamy et al., 2022).

While the convenience and efficiency offered by IoT are undeniable, the growing network of interconnected devices also raises concerns about security vulnerabilities in IoT devices because of the crucial data for the users that they hold. To address these challenges, a focus on enhanced security in IoT connected via WLANs has become imperative, with a particular emphasis on employing intrusion detection techniques.

Various IoT applications focus on automating different tasks and are trying to empower the inanimate physical objects to act without any human intervention. The existing and upcoming IoT applications are highly promising to increase the level of comfort, efficiency, and automation for the users Hassija et al., (2019). The applications of IoT

are increasing very rapidly and penetrating most of the existing industries. Although operators support these IoT applications through existing networking technologies, several of these applications need more stringent security support from technologies they use. Some of the application areas include but not limited to Smart Cities, Smart Environment, Smart Metering and Smart Grids, Security and Emergencies, Smart Retail, Smart Agriculture and Animal Farming and Home Automation. Major security threats encountered by IoTs include Eavesdropping and Interference, DoS Attack, Man-in-the-Middle Attack, Flooding Attack in Cloud, Data Thefts, Malicious Code Injection Attacks among others.

Machine learning algorithms, for instance, can analyze patterns of device interactions, detect anomalies, and predict potential security threats based on historical data (Spadaccino & Cuomo, n.d.). Additionally, real-time monitoring and response mechanisms are crucial components of enhanced security in WLAN-connected IoT environments. Rapid detection of intrusions allows for immediate countermeasures, reducing the likelihood and impact of security incidents (Muzammal et al., 2021). Furthermore, privacy concerns are paramount in the context of IoT devices, considering the wealth of sensitive information they often handle. Enhanced security measures must not only focus on preventing unauthorized access but also prioritize the protection of user data (Khan et al., 2022). Encryption, secure authentication mechanisms, and robust access controls are integral aspects of securing IoT devices in WLANs. Due to their restricted bandwidth capacity and worldwide connectivity, traditional intrusion detection solutions cannot ensure security in Internet of Things applications. In order to safeguard IoT devices from intrusions, a sophisticated Intrusion Detection System (IDS) is built. When suspicious activity or abnormalities occur, the system administrator can be notified by the IDS. Adaptive network security, or IDS, provides

the network administrator with insightful information on novel attack vectors (Roy et al., 2023).

Nowadays, IoT has been widely used in different applications to improve the quality of life. However, the IoT becomes increasingly an ideal target for unauthorized attacks due to its large number of objects, openness, and distributed nature. Therefore, to maintain the security of IoT systems, there is a need for an efficient Intrusion Detection System (IDS). IDSs implements detectors that continuously monitor the network traffic. There are various traditional IDs methods with different algorithms like Aho-Corasick, Boyer-Moore, Snort and Zeek. However, they have a disadvantages in terms of detection accuracy and time overhead (Ramadan & Yadav, 2020).

Classification of IDS based on detection methods can be as signature-based, anomaly-based, or specification-based. Attacks can be identified using signatures and recognized attack patterns by using signature-based techniques. A system for detecting anomalies picks up on departures from predefined normal behaviors. Similar to this, an administrator-defined set of rules is used by a specification-based system (Roy et al., 2023). Adaptable techniques like deep learning (DL) and machine learning (ML) are utilized to maximize the benefits of IDS. A single classifier, a model that uses only one classifier or a multi-classifiers model which uses several classification models concurrently can serve as the foundation for the machine learning model. Key IDSs algorithms like Snort, Zeek Aho-Korasik can perform intrusion detection but then at the expense of lower throughput hence compromising IoT potential.

The IDS model is further classified into two categories: binary and multiclass classification models. Binary classification distributes traffic into two categories: normal (0) and abnormal (1). In contrast, multiclass categorization identifies the type of attack. The multiclass model is more advanced than the binary model, which results

in reduced accuracy for unknown attacks when the data does not contain enough attack cases to train the model. Furthermore, security assaults are classified as active or passive attacks (Alwarafy et al., 2021).

An active attack appears during run-time conditions. It can disturb and harm the physical equipment. Active assaults are more difficult to perform and detect than passive attacks, with the most frequent examples being denial of service, message modification, packet replay, and spoofing. Passive attacks observe and monitor information about a specific target. The attackers hide and keep the communication line open in order to gather information, but the data remains untouched. The most typical risks include eavesdropping, man-in-the-middle, network mapping, and traffic analysis. To mitigate the impact of these attacks on IoT devices and consumers, a real-time network anomaly-based intrusion detection system is required to identify and prevent adversaries. IoT devices can assist make data-driven choices, automate procedures, and improve resource management. When installing IoTs, data security and privacy must come first (Ram et al., 2023) .

The Internet of Things (IoT) provides a variety of cutting-edge applications that improve operational efficiency and services (Ram et al., 2023). This is further expounded in the following areas, Smart Management IoT technologies enable real-time tracking of resources by facilitating automated inventory management. Data privacy safeguards make sure that customer information is safely kept and accessible only to those who are authorized.

Environmental Monitoring IoT sensors keep an eye on things like humidity, temperature, and air quality. This information aids in keeping the best possible conditions for safeguarding and preserving fragile materials. Privacy protections make

sure that data is anonymized and safely stored, and that the sensors do not unintentionally gather personal information.

Space Utilization and resource allocation by offering insights into space use, IoT-based occupancy tracking systems assist in making the best possible arrangements for seating, resource allocation, and facility management. Anonymizing occupancy data and getting user consent before collecting personal data help to address privacy concerns. (Mondal, 2021).

Personalized Services by utilizing users data, IoT makes personalized experiences possible. Personalized suggestions, alerts, and help are provided by beacons and location-based services according to the location and preferences of users. In order to protect privacy, data collection must have consent, stringent data access controls must be put in place, and openness regarding data handling procedures must be maintained.

Security and Surveillance IoT-based security systems keep an eye on restricted areas, entrances, and exits to improve security. Unauthorized access is detected by surveillance cameras and sensors, protecting both resources and users. Encrypting data transmission, limiting access to surveillance footage, and adhering to legal requirements are all factors that affect data privacy.

1.2. Problem statement

The Communication Authority of Kenya (CAK) has actively been monitoring and reporting on cyberattacks within Kenya. In the most recent report for third quarter 2024-2025, the National KE-CIRT/CC detected over 2.5 billion cyber threat events between January and March 2025, a 201.85% increase from the previous period. These attacks primarily target critical information infrastructure service providers.

IoT Intruders, who are spread across the Internet have become a major threat in our world, a number of techniques such as firewall and encryption to prevent such penetration and protect the infrastructure has been put into place, but with this, the intruders still manage to penetrate (Rama Devi & Abualkibash, 2019). Information availed via IoT devices is insecure due to inability of the existing Intrusion Detection Systems not being able to address the dynamics of the resource-constrained nature of IoT devices as they have low Accuracy levels and high False Alarm Rates (FAR). According to (Gyamfi et al., 2022) many IoT devices operate with limited computational power, memory and energy resources. Advanced Networked Agents for Security and Trust Assessment (ANASTA) monitory tool utilizes Software Defined Networks (SDN) and Network Function Virtualization (NFV) and requires significant computational resources and memory to operate efficiently, which is a limitation, especially in resource-constrained IoT devices (Bernabe et al., 2019).

Hybrid Intrusion Detection Systems (IDS) combine multiple detection techniques, such as signature-based, anomaly-based, and behavior-based detection, to enhance the accuracy and effectiveness of intrusion detection. When these are applied in IoT environments, it becomes complex due to the diverse nature of IoT devices, protocols and network architectures (Denning, 1987). Additionally, there is increased False Positives because while hybrid IDS aim to reduce false positives by combining multiple detection techniques, they can still generate false alarms, especially if the individual detection engines produce conflicting results or if the system is not properly calibrated. Managing alerts to distinguish genuine threats from false alarms can be labor-intensive and may require human intervention (Sahu et al., 2014).

IoT environments can consist of thousands of devices, each generating significant amounts of data. Montimage (MMT) IDS, struggles to scale effectively to handle this

volume of data, leading to potential performance issues and delays in detecting intrusions. Traditional IDSs may not support all the protocols commonly used in IoT environments. This limitation can result in blind spots where malicious activities occurring over unsupported protocols go undetected and therefore need to develop an IDS that is IoT sensitive in terms of resource constrain.

1.3. General objective

The objective of the research is to develop a Machine Learning model for monitoring and detection of intrusions in IoT devices.

1.4. Specific Objectives

i To analyze the existing intrusion detection models for Internet of Things (IoT)

ii To develop an intrusion detection model for monitoring attacks to enhance security for IoT devices.

iii To evaluate the model performance in terms of detection accuracy, precision, recall, F1-score and ROC AUC.

1.5. Research Questions

i What are the existing knowledge regarding Intrusion Detection System in mitigating attacks on IoT devices?

ii How can an intrusion detection system be developed to enhance security in IoT devices?

iii How will the accuracy of the developed IDS be evaluated to be able to distinguish between normal and anomaly traffic?

1.6. Scope of the Study

This researcher aims to develop and validate an IoT model using freely available datasets and a virtual IoT test environment. The research will focus on training a predictive model using NSL-KDD Dataset**,** preprocessing the data to extract relevant

features and evaluating the model's performance in a simulated IoT environment comprising IoT devices. The scope includes comparing the effectiveness of Machine Learning algorithms such as Artificial Neural Networks in predicting outcomes based on dataset test data. Limitations include the unavailability of specific IoT devices for real-world validation and the scope does not cover extensive hardware testing or large-scale deployment scenarios.

1.7. Justification of the study

The increasing prevalence of Internet of Things (IoT) devices usage in and the corresponding rise in attacks calls for an IDS that can be able to identify and mitigate threats in real-time, thereby providing a proactive defense against evolving attacks. IoT devices typically operate with limited computational power, memory, and energy resources hence need for an intrusion detection mechanisms optimized for resource efficiency of the IoTs. The research contributes to knowledge and practice by advancing the understanding of security challenges in IoT-WLAN environments and proposing innovative solutions. Moreover, the practical implementation of these solutions can have a direct impact on enhancing the security posture of real-world IoT deployments.

1.8. Significance of the study

This study is significant to the cybersecurity industry as it aims to develop a more efficient and intelligent Intrusion Detection System (IDS) capable of detecting and responding to sophisticated network attacks. As cyber threats become increasingly complex and frequent, traditional signature-based IDS solutions are often inadequate. This research introduces a machine learning based approach, specifically utilizing Artificial Neural Networks (ANN) to enhance the detection of known threats. For industry practitioners, the study provides an improved Threat Detection where a more adaptive model that increases the accuracy of detecting intrusion attempts thereby

reducing the risk of data breaches and operational disruptions. Detection automation through intelligent models reduces the need for constant manual monitoring leading to cost savings and faster incident response times. This results to operational Efficiency. The research outcomes can be integrated into existing security infrastructures like PFSense and other Firewalls with minimal modification, making it suitable for organizations of different sizes and sectors. Since technology keeps growing with new ideas coming in place, insights from the model can support cybersecurity analysts in identifying threat patterns and improving overall network defense strategies.

1.9. Assumption of the study

The study assumed that IDS model can handle the diversity of IoT devices, which may vary in terms of processing power, memory, and communication capabilities. Secondly, the study assumed IoT network was well defined and stable architecture used, including the types of devices, communication protocols, and network topology. Lastly, the study assumed there was sufficient and appropriate data available for training and testing the IDS. This includes datasets containing normal and anomalous behavior specific to IoT environments.

1.10. Limitation of the study

Threats keeps evolving and with time intrusion detection may become less effective. The model may not capture emerging threats or may even become outdated as attackers develop new techniques.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

This chapter critically reviews existing knowledge from published articles about current state of research and developments in intrusion detection system while adopting funnel approach. The researcher used Google Scholar, Sci Hub, Springer, IEEE to explore online resources to identify papers related to the research questions while articles were maintained using Mendeley desktop for easier referencing. This allows analysis of key literature covering global, regional and local spectra. It further looks on how various methods and technologies have been brought onboard to detect security intrusion and mitigate the serious raised security and privacy risks issues in this evolving field of technology.

2.2 Funnel Approach

Intrusion Detection Systems (IDS) are essential components of modern cybersecurity frameworks. They monitor network traffic for suspicious activity and potential threats, alerting system administrators to intrusions. With the increasing complexity and frequency of cyber-attacks, researchers globally, regionally, and locally have focused on developing advanced IDS models to enhance detection accuracy and efficiency.

2.2.1 Global spectrum

Globally, the evolution of IDS has seen significant advancements driven by the integration of machine learning (ML) and artificial intelligence (AI). Researchers worldwide have contributed to this field, focusing on improving the detection rate, reducing false positives, and enhancing real-time analysis capabilities. Numerous global studies have integrated machine learning algorithms into IDS. For instance, a study by Buczak et al., (2016) provided a comprehensive review of various ML

techniques used in IDS, highlighting their effectiveness in detecting anomalies and classifying intrusions with high accuracy.

The adoption of deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), has also been explored extensively, showing promising results in handling large datasets and identifying complex attack patterns. The integration of big data technologies with IDS has been another significant global trend. Researchers have leveraged Hadoop and Spark frameworks to process vast amounts of network traffic data, enabling real-time intrusion detection Xu et al., (2019). This approach has improved scalability and performance, addressing the challenges posed by the exponential growth of network data. Combining multiple techniques to enhance detection capabilities has gained traction globally. Hybrid IDS models, which integrate anomaly-based and signature-based methods, have shown to be more effective in identifying zero-day attacks and known threats Al-Yaseen et al., (2017a) and Al-Yaseen et al., (2017b). Such models utilize the strengths of both approaches, offering a more robust defense mechanism.

2.2.2 Regional spectrum

Regionally, the focus on IDS has been influenced by specific cybersecurity challenges and regulatory requirements. Different regions have adopted distinct approaches based on their unique needs and threat landscapes. In North America, the emphasis has been on developing IDS models that comply with stringent regulatory standards, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA). Research has focused on ensuring that IDS can protect sensitive data while maintaining compliance (Somade et al., 2019). Additionally, there has been significant investment in AI-driven IDS to address sophisticated cyber threats targeting critical infrastructure. In Europe researchers have concentrated on privacy-

preserving IDS models. The implementation of the GDPR has driven the need for IDS that can detect intrusions without compromising user privacy. Studies have explored the use of homomorphic encryption and differential privacy techniques to achieve this balance (Hassan et al., 2020). Furthermore, the European Union's Horizon 2020 program has funded several projects aimed at advancing IDS technologies. In Asian-Pacific region, the rapid adoption of IoT and smart technologies has spurred research on IDS for these environments (Zhang et al., 2019). There has also been a focus on collaborative IDS frameworks that leverage information sharing across organizations to improve detection rates.

2.2.3 Local spectrum

Locally, research efforts have often targeted specific sectors or organizations, addressing unique cybersecurity needs. For instance, Academic Institutions like local universities and research institutions have played a crucial role in advancing IDS technologies. Collaborations between academia and industries have led to the development of specialized IDS models tailored for local enterprises and government agencies. For example, research investigations have focused on IDS for securing digital payment systems, given the country's push towards a cashless economy (Mahapatra et al., 2021). Small and Medium-sized Enterprises (SMEs) and startups have also contributed to IDS research, often focusing on cost-effective and easily deployable solutions.

In Africa, for example, research has explored the use of open-source IDS tools like Snort and Suricata, customized to meet the needs of local businesses (Dileep et al., 2021). These efforts aim to provide affordable cybersecurity solutions without compromising on effectiveness. Local governments and defense agencies have prioritized the development of IDS for protecting critical national infrastructure. In

countries like Israel, renowned for its cybersecurity expertise, research has focused on developing state-of-the-art IDS models that leverage advanced AI techniques and threat intelligence to safeguard against nation-state attacks (Rrushi, 2022).

2.3 Broad Overview of IoT

Internet of Things (IoT) is providing access to huge amount of data connected over WLANs hence becoming an integral component of the modern technology. The Internet of Things (IoT) is expanding rapidly all over the world, and security is now a major concern. Infested Internet-connected objects pose a threat to not only IoT security but also the entire Internet ecosystem, which may be exploited by malicious entities using infected Things (smart devices) as botnets. For example, distributed denial of service attacks caused the Internet to become unusable and compromised video surveillance devices with the Mirai malware. Security attack vectors have changed recently in terms of both diversity and complexity. Thus, it's critical to examine approaches in the context of the Internet of Things in order to recognize, stop, or detect new attacks. (Chaabouni et al., 2019).

It is predicted that technology will soon be next to people due to its general growth. Nowadays, we see many applications that suit our life, such as smart cars, smart homes, smart traffic management, smart offices, smart medical consultation, smart cities, etc. All such facilities are within the reach of the common man due to the advances in Information and Communication Technology (ICT). Thanks to this development, new computing and communication environments such as the Internet of Things (IoT) have come into the picture. There is a lot of research in the field of IoT, which contributes to the overall development of society and makes life easy and convenient. However, in the resource-constrained environment of Wireless Local Area Network (WLAN) and IoT, it is almost impossible to create a fully secure system. Because we move very fast,

technology is increasingly vulnerable to security threats. In the future, the number of people connected to the Internet will be less than the number of smart objects, so in order to keep the above environments safe, we must prepare a strong system and standardize it to manage the communication between IoT objects smoothly (Pundir et al., 2020).

Application of IDS is expanding, and with the increasing numbers of connected devices, exploitation of those devices becomes more common. Since IoT devices and IoT networks are used in many crucial areas in modern societies, ranging from everything between security and military applications to healthcare monitoring, education and production efficiency, the need to secure these devices is of great importance for researchers and businesses (Bull, 2023). The flexibility of networked devices increases the likelihood of continuous attacks against them. The low processing power and small memory of IoT devices have made it difficult for security analysts to trace various attacks against these devices during forensic analysis. Forensic analysis assesses how much damage has been caused to devices by various attacks (Framework et al., 2022). Because of the limitations of IoT devices and the nature of their functioning, conventional security procedures are less effective in preventing threats that are specific to systems (Chowdhury et al., 2021).

The number of networked devices in the world was increasing rapidly, and it was expected that there would be 50 billion devices connected to the Internet by the end of the year 2020. This explosion of devices has led to a spike in IoT-based cyber-attack incidents. To alleviate this challenges, there is a requirement to develop new intrusion detection techniques for detecting attacks initiated from compromised IoT devices (Asharf et al., 2020). Unfortunately, this has attracted the attention of hackers, who make IoT a target of malicious actions, opening the door for a possible attack on the

end nodes. Due to the large number and diverse types of IoT devices, it is a challenging task to protect the IoT infrastructure using a traditional intrusion detection system (Khraisat et al., 2019). Millions of smart devices are installed within complicated networks to provide vibrant functionalities, such as communications, education, monitoring, and control of essential infrastructure. However, because of the limited bandwidth and resources, the exponential growth of devices and the resulting massive data traffic generated at the network's edge placed additional strain on the state-of-the-art centralized cloud computing paradigm. (Alwarafy et al., 2021).

The technological advancement of the Internet of Things (IoT) devices in our world today has become beneficial to many users but has security issues that are left unattended. IoT devices have the ability to connect to other devices on the internet to transmit and share data from anywhere. Hence, the need to secure these devices as they improve the quality of life comfort. Research shows that about 70% of IoT devices are easy to hack (Azumah et al., 2021). The Internet of Things (IoT) is a network of "things" that communicate via the Internet to collect and share data. These "things" can be sensors, actuators, smartphones, wearables, computers, or any object that is connected to perform specific services (Mishra, 2022).

Similarly, Wireless Local Area Network (WLAN), a component of IoT, forwards acquired data after detecting an incident. The scalability and heterogeneity of the Internet of Things provide inadequate protection and make it vulnerable to a variety of attacks, including WLAN-inherited attacks. Furthermore, the IPv6 routing protocol for low power and lossy networks (RPL), which is the de facto routing protocol for IoT networks, has various vulnerabilities due to its features and functionality. Researchers offered a variety of mitigation strategies for safe networks and routing in IoT (Muzammal et al., 2021). To truly benefit from the undeniable innovation potential of

the IDSs new security issues have been raised by the widespread adoption of the IoT paradigm in a number of application domains. These issues should be carefully considered. The introduction of new threats that impact all architectural layers has actually been caused by the heterogeneity of involved technologies, including the integration of various devices and networks with limited resources. This has prompted the design and enforcement of appropriate security countermeasures, including efficient monitoring capabilities. (Casola et al., 2019).

2.4 Concept of IoT Use

Different sectors have had to adjust and preserve a pertinent nearness in our society and adopt lots of technological innovation (Mondal, 2021). IoT is being used more and more to improve user experiences and operations. Intelligent shelf management, self-checkout kiosks, and automated inventory tracking are just a few examples of intelligent management systems that optimize resource use and streamline operations. IoT-enabled environmental monitoring, which keeps an eye on temperature, humidity, and air quality in spaces, guarantees the preservation of fragile materials. Occupancy tracking enabled by IoT offers insightful data on space usage, facilitating efficient resource allocation and planning (Ram et al., 2023). In order to enforce the automated IoT environment, high level of safety and privacy, authentication and recovery from attacks is required (Srinadh et al., 2021). With the goal of shielding this IoTs, IDSs comes in handy.

2.5 Common Types of IoT Attacks

Common types of attacks are denial-of-service (DoS), Distributed DoS (DDoS), Man-in-the-middle (MitM), SQL Injection, Cross-site scripting, Eavesdropping and Malware. There are several classifications for IoT attacks against WLANs Butun et al., (2020). Data security challenges for IoT systems refer to data security issues at different

layers of the Internet of Things. The physical layer faces difficulties with physical damage, hardware malfunctions, and power constraints. Network layer challenges include DoS attacks, sniffing, gateway attacks, and unauthorized access. The application layer is challenged by malicious code attacks, application vulnerabilities, and software bugs (Elrawy & Awad, 2018). Attacks can further be classified as Passive or Active attacks.

2.5.1 Passive attacks

Passive attacks usually involve attackers hidden, and uses lines of communication to gather information. This is because attacks do not cause radio transmissions. Because wireless communication links are easier to tap, wireless networks are more vulnerable to passive attacks such as eavesdropping, which can be easily carried out by listening to the wireless communication between WLAN sensor nodes without eavesdropping on them. Passive attacks can be grouped into eavesdropping, also known as "passive data collection", and include the interception of communications; Destroying a Node: Physically destroying a Node (using electric shock, physical force, or ammunition) by any means so that the Node is no longer functional; Node failure: This can be caused by various factors, such as faulty sensors or sensor overload or loss of power from other DoS attacks; Node Outage: This attack occurs whenever the normal operation of a node fails. For example, if a cluster head in a heterogeneous network fails in normal operation, WLAN protocols must be robust enough to mitigate the negative effects of such node disruptions by selecting new cluster heads and/or providing alternative routes to the network. roads; Traffic Analysis: The traffic pattern of a network can be as valuable as the content of data packets intended for adversaries. Important information about network topology can be obtained by analyzing traffic patterns.

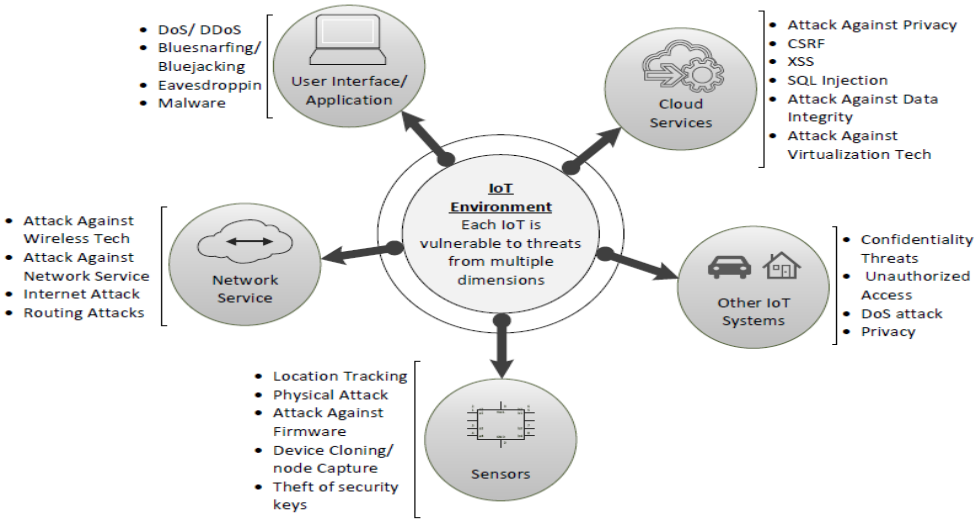
2.5.2 Active attacks

Active attacks are not only against data confidentiality, but also against data integrity. Active attacks can also target the use of unauthorized resources or disrupt the adversary's communications. An active attacker performs a radio transmission or activity that is detected by WLAN elements. Jamming DoS occurs at the physical layer and can interfere with the signal when transmitting on the same frequency; A black hole occurs when a malicious node drops all packets received for transmission. A bottleneck occurs when a malicious node broadcasts to all neighboring nodes that it is the best next hop to send packets to its destination.

Figure 2.1 shows different types of attacks that occur in different IoT environments. The diagram illustrates the security threat landscape in an IoT (Internet of Things) environment, highlighting how different components of an IoT system are susceptible to specific types of cyber threats. At the center is the "IoT Environment", which is described as being vulnerable to threats from multiple dimensions. Surrounding it are five key components, each with its own associated threats. The diagram effectively shows that each component in the IoT ecosystem has unique security concerns. Together, these vulnerabilities highlight the need for a multi-layered and comprehensive approach to securing IoT systems.

Figure 2. 1

IoT Environment Threat Dimensions



Source: *Memos et al., 2022*

2.6 Background of IDS security

Researchers have developed Intrusion Detection Systems (IDS) capable of detecting attacks in several available environments. A boundlessness of methods for misuse detection as well as anomaly detection has been applied. Many of the technologies proposed are complementary to each other, since for different kind of environments some approaches perform better than others. IDS are designed to detect unauthorized access or anomalies in network traffic that could suggest a security breach. In the context of IoT, IDS can be broadly classified into three categories: Network-based IDS (NIDS), Host-based IDS (HIDS), and Hybrid IDS.

2.6.1 Network-based IDS (NIDS)

NIDS monitor network traffic for suspicious activities with key studies areas as Deep Learning Approaches (Bobo et al., 2004) who proposed a deep learning-based NIDS using Convolutional Neural Networks (CNNs) to analyze network traffic patterns. Their model demonstrated high accuracy in detecting both known and unknown attacks.

Signature-based Techniques (Singh et al., 2017) focused on signature-based NIDS, which rely on predefined attack signatures to detect intrusions. While effective for known threats, these systems struggle with zero-day attacks.

2.6.2 Host-based IDS (HIDS)

HIDS monitor activities on individual devices while encompassing behavioral Analysis (Kumar et al., 2019) developed a HIDS using behavioral analysis to detect deviations from normal device behavior. Their system leveraged machine learning algorithms to model typical behavior patterns and identify anomalies. Rule-based Systems (Al-Hayanni et al., 2020) implemented a rule-based HIDS, which used predefined rules to monitor device activities. While straightforward, this approach requires constant updates to the rule set.

2.6.3 Hybrid IDS

Hybrid IDS combine aspects of both NIDS and HIDS to provide comprehensive security coverage. Significant research in this domain includes collaborative Models (Lyu et al., 2019) proposed a collaborative IDS where multiple IoT devices share information to detect intrusions collectively. This approach enhances detection accuracy but raises concerns about data privacy and communication overhead. Multi-layered Detection (Azmoodeh et al., 2019) introduced a multi-layered IDS that integrates network and host-based detection mechanisms. Their model showed improved performance in complex IoT environments.

2.7 Machine Learning in Intrusion Detection

Intrusion Detection Systems (IDS) are vital components in securing computer networks and systems against unauthorized access, attacks, and anomalies. As cyber threats become more sophisticated, traditional signature-based IDS approaches have shown limitations, especially in detecting novel or previously unseen attacks. This challenge

has led to increased research and development of machine learning (ML)-based intrusion detection systems, which offer intelligent, adaptive, and automated security mechanisms.

Machine learning-based IDS work by learning patterns from historical network traffic data and identifying deviations from normal behavior. Unlike static, rule-based methods, ML-based systems can generalize from data, enabling them to detect zero-day attacks, advanced persistent threats, and unknown intrusions. The literature on this subject encompasses a wide variety of algorithms, datasets, and system designs aimed at enhancing detection accuracy, reducing false positives, and ensuring real-time responsiveness. Machine learning (ML) and artificial intelligence (AI) have emerged as powerful tools in developing effective IDS for IoT. These technologies can learn from historical data to identify patterns and predict future threats with the following notable algorithms.

2.7.1 Supervised learning

Supervised learning is a machine learning approach where an algorithm is trained using labeled data. This means that each piece of training data contains both the input features and the correct output or target. The aim is for the model to learn a relationship or mapping between the inputs and outputs so that it can accurately predict the output for new, unseen data. The process begins with feeding the algorithm a dataset that includes various examples. For each example, the model sees the input variables, known as features and the correct answer, which is called the label. Through this training, the model learns patterns in the data. Once the model has been trained, it can be tested on new inputs where the correct output is unknown. Based on what it learned during training, the model makes predictions.

(Doshi et al., 2018) applied supervised learning techniques such as Support Vector Machines (SVM) and Random Forests to detect intrusions in IoT networks. Their models achieved high detection rates but required substantial labeled training data. Unsupervised Learning (Meidan et al., 2017) explored unsupervised learning methods, including clustering algorithms, to identify anomalies without labeled data. This method is very beneficial for identifying unusual attacks but may produce more false positives.

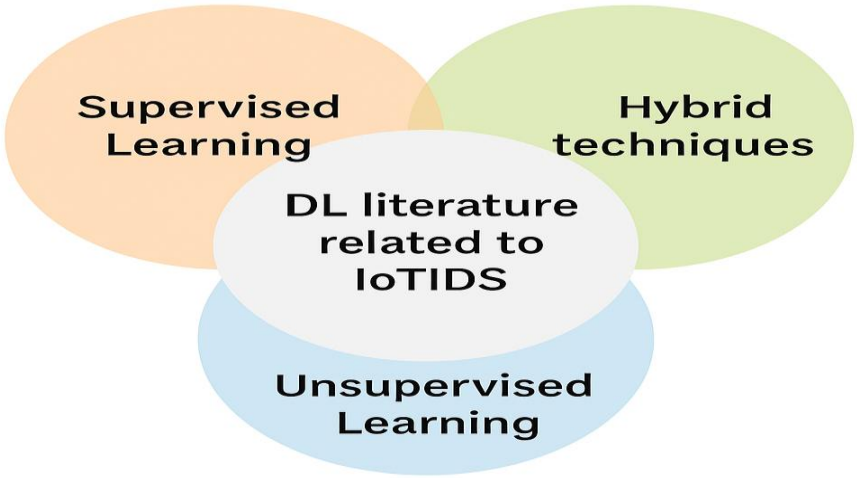
Reinforcement Learning (Ayoub et al., 2020) investigated reinforcement learning for dynamic IDS configuration. Their system adapted to changing network conditions and attack strategies, improving long-term security. An intrusion detection system (IDS) is a hardware or software element that detects malicious activity on networks or computer systems and keeps security in place. While network-based intrusion detection systems (NIDS) target an entire network, host-based intrusion detection systems (HIDS) focus on a single computer system. IDSs function by either learning the typical behavior of the system and reporting if any anomalous events occur, referred to as anomaly-based, or by cross-checking monitored events with a database of known intrusion experiences, known as signature-based. (Spadaccino & Cuomo, n.d.). Hybrid IDS systems combine signature-based and anomaly-based approaches to enhance overall detection capabilities.

Machine learning (ML) techniques have gained prominence in enhancing the efficiency of IDS for IoT. Researchers have explored the application of supervised and unsupervised learning algorithms, including decision trees, support vector machines, and deep learning (DL), to identify malicious activities in IoT networks. The ability of machine learning models to adapt to evolving threats makes them well-suited for the dynamic nature of IoT environments. DL plays the most important role in IoT security

because IoT environments are defined by the generation of large volumes and a wide range of data (Khraisat et al., 2019). Complex feature sets can be automatically modeled by DL using sample data. DL in Internet of Things networks is made possible by DL algorithms, which is another benefit. This makes it possible for designated collaborative tasks to be carried out by IoT-based systems automatically when human intervention is not present. (Memos et al., 2022). DL can be applied in a generative mode with unsupervised learning, a discriminative mode with supervised learning, or a hybrid technique that combines both modes.

Figure 2. 2

Taxonomy of Deep Learning techniques for IoT IDS



Source: *Memos et al., 2022*

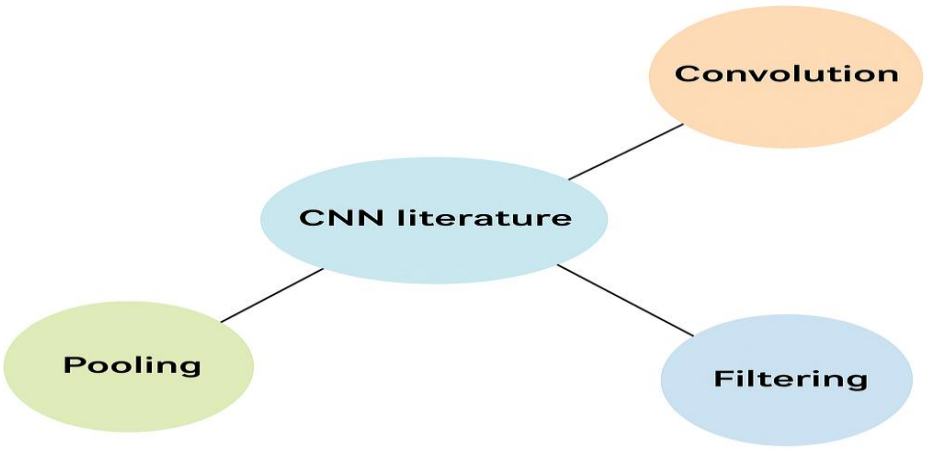
2.7.2 Convolutional Neural Network (CNN)

CNN is a deep learning (DLL) algorithm that was designed to reduce the amount of data inputs needed for a conventional ANN (Artificial Neural Network) by using equivariant representations, sparse interactions and sharing parameters. This makes CNN more scalable and less time-consuming to train. There are three levels of CNN: convolutional, pooling and activation. Convolutional layers employ various kernels to convolve data inputs. Pooling layers reduce the size of succeeding layers by reducing

the number of samples. There are two techniques for pooling: max and average. Max pooling calculates the maximum value for each cluster in past layers after dividing the input into distinctive clusters. Average pooling calculates the average value of each cluster in previous layers. Activation is the non-linearity of the activation function on each feature in a feature set. CNN is the most efficient and fast way to extract features from raw data, but it requires a lot of computing power. Therefore, using CNN on IoT devices with limited resources is very difficult.

Figure 2. 3

Illustration of convolution neural network working



Source: *Memos et al., 2022*

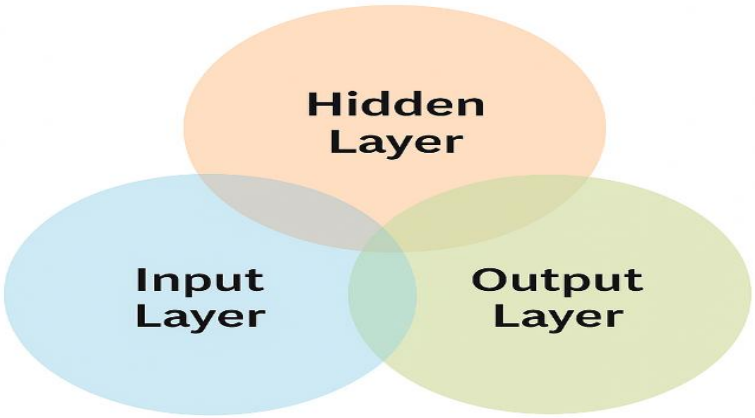
2.7.3 Deep Belief Network (DBN)

Since DBN is developed by layering two or more Restricted Boltzmann Machines (RBMs), it can be considered as unsupervised learning-based generative algorithms. They perform strongly with unsupervised training for each layer separately. The first features for each layer are extracted in a pre-training phase, followed by a fine-tuning phase in which the softmax layer is applied to the higher layer. It mainly consists of two layers ie. visible layer and hidden layer. Although the study discusses the detection

of malicious attacks using DBN with relatively better results than ML algorithms, there is no evidence in the literature that it is applicable in an IoT environment.

Figure 2. 4

Illustration of deep belief network working



Source: *Memos et al., 2022*

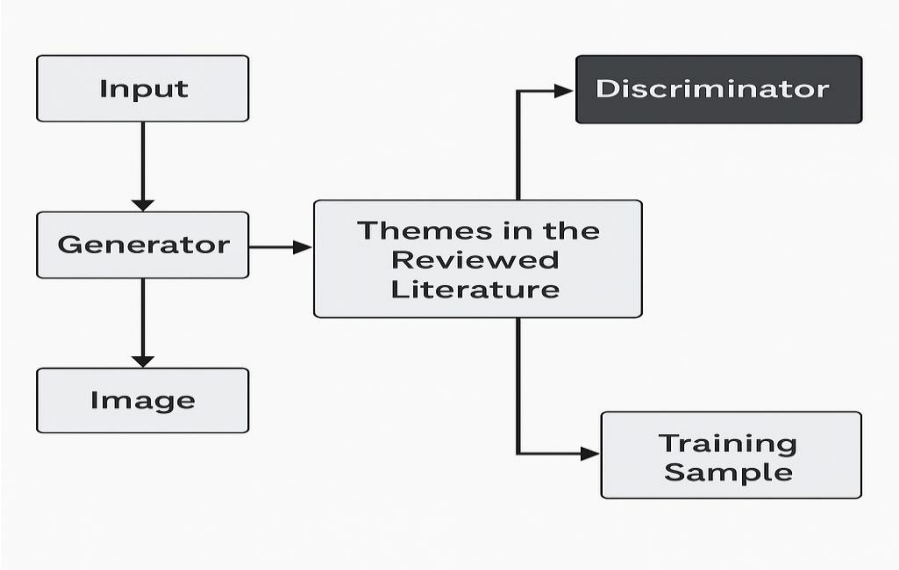
2.7.4 Generative Adversarial Network (GAN)

GAN is a hybrid deep learning method that combines both generative and discriminative models for training. The distributions of the dataset and the samples are obtained using the generative model's predictions about the genuine origin of the training dataset for the given sample, and are made using the discriminant model.When the generative and discriminant models are used to generate a sample using random noise, they both act as antagonists. The discriminant model, on the other hand, attempts to authenticate the actual training data samples using the fake samples generated by the generative model. Here, D(x) represents the binary classification that results in real or false (created). The correct/incorrect classification measure inversely determines the accuracy and efficiency of both models. This results in updating the models during each iteration.

The usefulness of the GAN algorithm is to detect anomalous behavior in IoT environments because they can combat zero-day attacks by creating samples that imitate zero-day attacks, allowing the discriminator to learn different attacks scenarios. However, the challenge of using GAN is that it is difficult to train and produces unstable results.

Figure 2. 5

Illustration of generative adversarial network (GAN)



Source: *Memos et al., 2022*

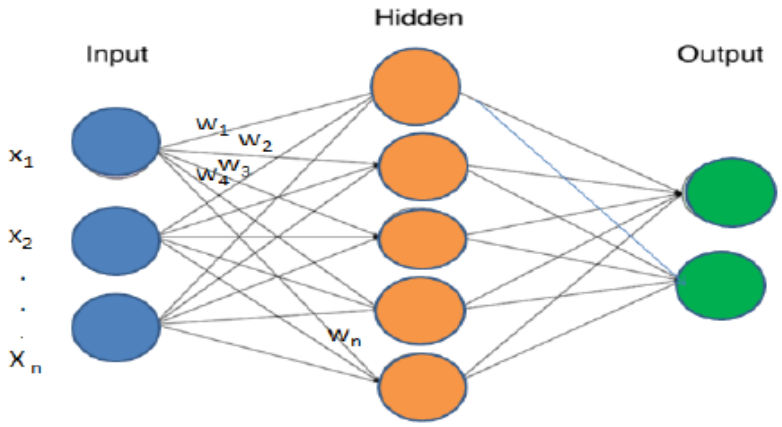
2.7.5 Artificial Neural Network (ANN)

Machine Learning techniques are implemented with the Intrusion detection systems to enhance the performance of IDS. Various studies on IoT reveals that Artificial Neural Network (ANN) provides increased detection rate and accuracy than other approaches (Anitha & Arockiam, 2019). It provides better detection accuracy rate in terms of true and false alarm rates. In neural network, the weight, the associated bias and the number of epochs given for the training phase will determine the accuracy of the classification. The neural network consists of an input layer, number of hidden layers and an output layer. Each layer has number of neurons. The information enters the neural network via

the input layer, it is processed in the hidden layers and the result then be retrieved in the output layer.

Figure 2. 6

Artificial Neural Network



Source: *Anitha & Arockiam, 2019*

ANN is a well-suited for recognizing complex patterns and relationships within data, by being able to learn and identify patterns associated with normal network behavior as well as patterns indicative of malicious activities. These attributes make it preferable in the context of this intrusion detection model. The developed model to monitor intrusion would include parameters like packet headers to provide network traffic behavior; payload content to monitor attack signatures; connections statistics; traffic patterns; user behavior; system logs.

2.7.6 A Support Vector Machine (SVM)

A Support Vector Machine (SVM) classifier is a powerful supervised learning algorithm used for classification tasks and also for regression. It is especially effective in high-dimensional spaces and in cases where the number of features is greater than the number of samples. At its core, an SVM aims to find the best boundary called a hyperplane that separates data points of different classes in the feature space. The "best"

boundary is the one that maximizes the margin, the distance between the hyperplane and the nearest data points from each class. These closest points are known as support vectors, and they are the key elements in determining the hyperplane.

2.7.7 Particle Swarm Optimization (PSO)

PSO is a population-based optimization technique inspired by the natural movement and collective behavior of swarms, such as flocks of birds or schools of fish. In nature, these groups are known to move together toward food sources or away from predators by relying on both their own experiences and the behavior of those around them. PSO borrows this idea and applies it to solving mathematical and computational problems, particularly those that require finding the best solution from a wide range of possibilities. In PSO, each potential solution to the problem is represented by a particle. These particles "fly" through a solution space, each trying to find the optimal or best answer. Every particle has a position which represents a possible solution and a velocity which controls how it moves. Importantly, each particle remembers the best solution it has personally encountered, and it also keeps an eye on the best solution found by any particle in the entire group. The movement of each particle is guided by three factors:

i Its current position.

ii Its personal best position (the best solution it has seen so far).

iii The global best position (the best solution found by the whole swarm).

With each step or iteration, the particles adjust their velocities and move closer toward the personal and global best solutions. This allows them to explore the space intelligently—balancing between trying new areas (exploration) and refining known good areas (exploitation). What makes PSO powerful is its simplicity and adaptability. It doesn’t require knowledge of gradients or complex mathematical models, which makes it suitable for problems where traditional methods fail or where the solution

space is very irregular or multi-dimensional. In the context of machine learning, PSO is widely used for optimizing model parameters, such as the learning rate in neural networks, the kernel parameters in support vector machines, or even selecting the most relevant features from a dataset. Since PSO is not restricted by the type of data or model, it offers a flexible and efficient method to improve performance in many real-world applications.

In essence, PSO is a smart search technique driven by cooperation and experience, where a community of simple agents work together to find the best solution to a problem, much like a flock of birds instinctively finding their way to food.

2.8 Intrusion Detection Systems

IDSs act as an effective way to achieving higher security in detecting malicious activities.

2.8.1 ANASTA monitoring tool

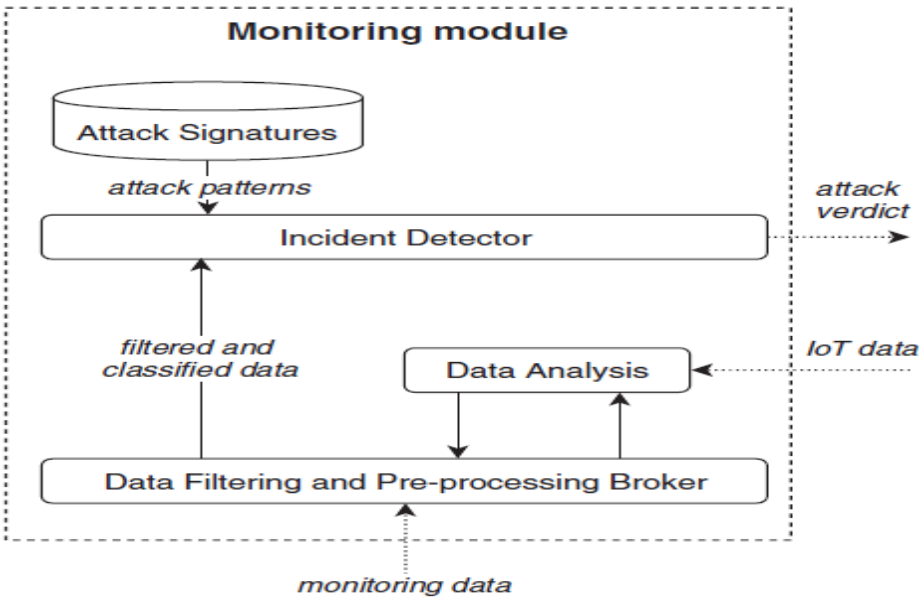
ANASTA provides security monitoring services able to detect potential security breaches and attacks on cyber-physical networks. In particular, the project leverages both anomaly-based detection techniques relying upon artificial intelligence algorithms to better the likelihood of detection and specific technical extensions to cope with different IoT architectural layers and technologies.

Security monitoring services are offered by Advanced Networked Agents for Security and Trust Assessment (ANASTA), which can identify possible security lapses and assaults on cyber-physical networks. An anomaly-based detection tool in the Data Analysis component can identify questionable alterations in sensor behavior. After obtaining the data from the various monitoring agents that are available, the Data Filtering and Pre-processing component performs preliminary pre-processing and aggregation. The module's central component is the incident detector. After gathering

and analyzing the data, it raises an alarm when a security incident is found. Lastly, the repository of attack patterns required for the signature-based analysis is contained in the Attack Signatures (Bernabe et al., 2019).

Figure 2. 7

ANASTA Monitoring Module Architecture



Source: *Butun et al., 2020*

To this end ANASTA has designed a plane-based architecture where the information flows from the data acquisition from the IoT infrastructure to their dissemination and consumption by the monitoring infrastructure and to the data processing by the reaction module to decide about mitigations to enforce. ANASTA used Knowledge Discovery Dataset (KDD Cup 1999) and attained Accuracy of 95.75% and a false alarm rate (FAR) of 1.87%.

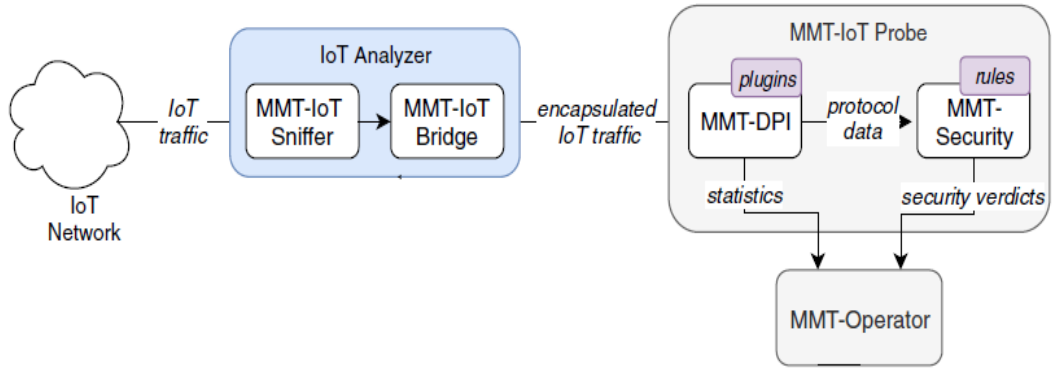
2.8.2 Montimage monitoring tool

The Montimage company created the MMT-IoT, a Network Intrusion Detection System (NIDS). An intrusion detection system (NIDS) is a device that monitors traffic to and from any networked device. It can track and examine the network packets that flow

through a single computer system's network interfaces in addition to the internal workings of the system itself. The software is distinguished by its modular architecture, which offers adaptability and flexibility in accordance with the demands of the domain (Casola et al., 2019).

Figure 2. 8

Montimage monitoring tool



Source: *Butun et al., 2020*

The *MMT-Security* module implements signature-based detection techniques. Each security incident is characterized by a pattern, represented by a particular sequence of messages, commands or header values. In the *MMT- Security* module, an incident is described in form of a security rule, which specifies a sequence of events. It achieved an Accuracy of 95.75% and a false alarm rate (FAR) of 1.87% while using KDD99.

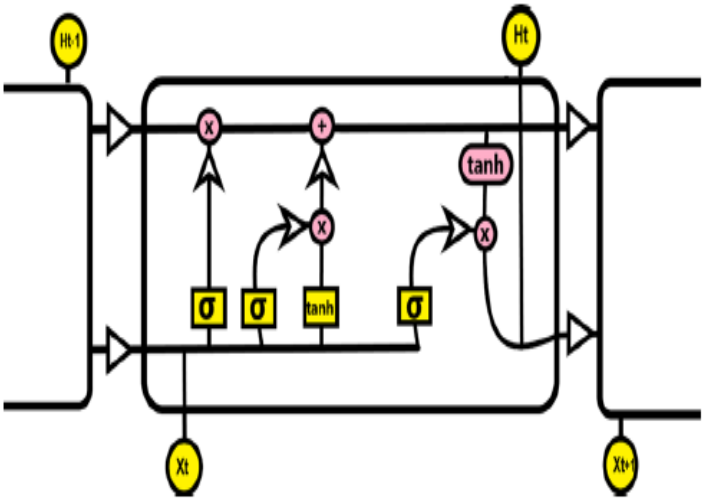
2.8.3 Long short-term memory

The long short–term memory (LSTM) is a type of recurrent neural network (RNN) that can classify and make predictions for temporal-dependent data such as time series datasets and signal datasets. One of the most important characteristics of an LSTM-based RNN is its ability to store information or cell status for future use within the network. As a result, they are well-suited for performing temporal data analysis on

changes in the data over time. This is one of the reasons why LSTM network solutions are preferred for identifying anomalies in time series sequence data.

Figure 2. 9

Long Short-Term Memory (LSTM) Model

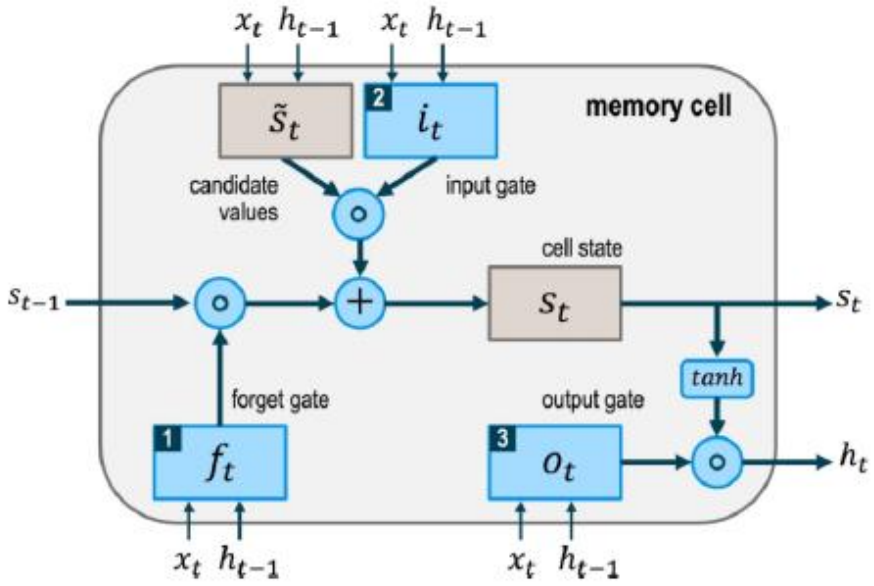


Source: *Laghrissi et al., 2021*

Research proves that comparing LSTM to other algorithms such as recurrent cascade-correlation, neural sequence chunking concludes that the LSTM learns precisely compared to the other algorithms. Furthermore, lags tasks that were recurrent network algorithms in the past could not be broken down and handled using LSTM. The LSTM was introduced by Hochreiter et al and its architecture consists of the input gate, a memory cell, and an output gate. In an LSTM network, the memory cells contain the features of the LSTM network. The memory cells comprise a forget gate (ft), input gate (it), and an output gate (ot), respectively.

Figure 2. 10

LSTM block architecture



Source: *Azumah et al., 2021*

2.8.4 LSTM existing technique

Laghrissi et al., (2021) implemented deep learning solutions for detecting attacks based on Long Short-Term Memory (LSTM). PCA (principal component analysis) and Mutual information (MI) were used as dimensionality reduction and feature selection techniques. The approach was tested on a benchmark data set, KDD99, and the experimental outcomes show that models based on PCA achieve the best accuracy for training and testing, in both binary and multiclass classification. This achieved an Accuracy of 96.6% and False alarm rate (FAR) of 1.7%.

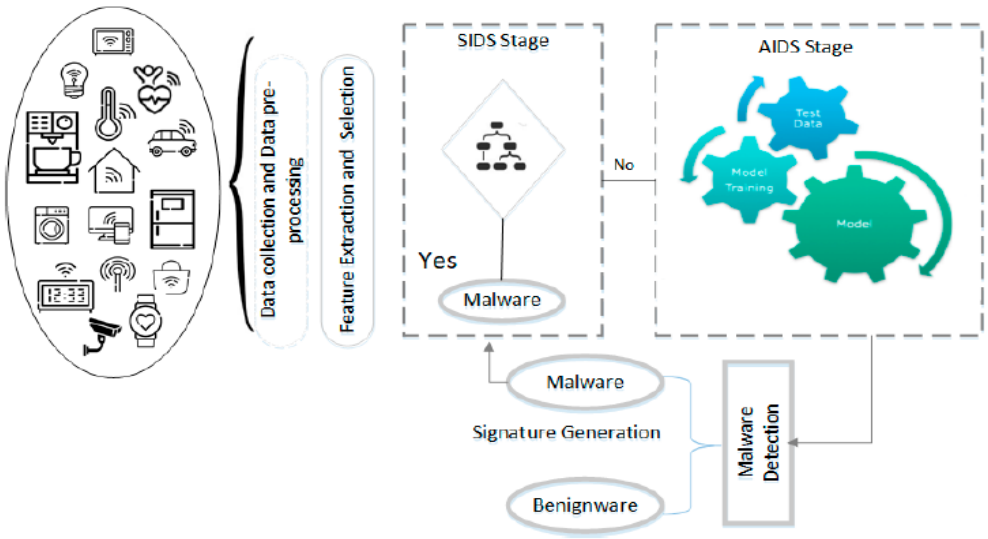
2.8.5 Hybrid intrusion detection system

A C5 classifier and a One Class Support Vector Machine classifier are combined in HIDS. The benefits of both Anomaly-based Intrusion Detection System (AIDS) and Signature Intrusion Detection System (SIDS) are combined in HIDS (Memos et al., 2022) and they detect both the well-known intrusions and zero-day attacks with high

detection accuracy and low false-alarm rates. The suggested HIDS is tested using the Bot-IoT dataset, which includes both legal IoT network traffic and various sorts of assaults.

Figure 2. 11

Hybrid Intrusion Detection System for the IoT ecosystem



Source: *Memos et al., 2022*

Maseno et al., (2022) HIDS technique aims at optimizing the classical algorithms. Integrated-based hybrids are more efficient and give better results compared to other forms of hybrid techniques. Thus, to develop an efficient and effective IDS, integrated-based hybrid should be adopted in developing the IDS. Accuracy of 96.7%, False alarm rate (FAR) of 1.5% while using KDD Cup 1999 as a dataset.

Table 2. 1

Summary of the existing Intrusion detection models

Author	Model	Dataset	Metric	Gap
Bernabe et al., 2019	Advanced Networked Agents for Security and Trust Assessment (ANASTA)	KDD Cup 1999	Accuracy of 95.75% and a false alarm rate (FAR) of 1.87%.	Protocol-dependent hence does not address some IoT protocols e.g Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (CoAP), Data Distribution Service (DDS), Message Queue Telemetry Transport (MQTT). May not detect new, previously unknown vulnerabilities and exploits, known as zero-day attacks, as it relies on known signatures and patterns.
Butun et al., 2020	Montimage IDS monitoring tool.	DARPA	Accuracy of 82%, detection rate and false alarm	Has potential performance issues and delays in detecting intrusions due to huge amounts of data generated by IoT environment.

			rate (FAR) of 2.3%.	Real-time analysis is resource-intensive, potentially affecting the performance of the IDS and the IoT network.
Memos et al., 2022	Hybrid Intrusion Detection System.	KDD Cup 1999	Accuracy of 96.7%, False alarm rate (FAR) of 1.5%	Need to develop lightweight version that are suitable for deployment on resource-constrained IoT devices. Resource Intensive: Anomaly-based detection, most importantly when using machine learning and deep packet inspection, can be resource-intensive, potentially impacting system performance and scalability.
(Laghrissi et al., 2021)	Long Short-Term Memory (LSTM)	KDD Cup 1999	Accuracy of 96.6%, False alarm rate (FAR) of 1.7%	Model implemented deep learning solutions for detecting attacks based on Long Short-Term Memory (LSTM) as a classification algorithm.

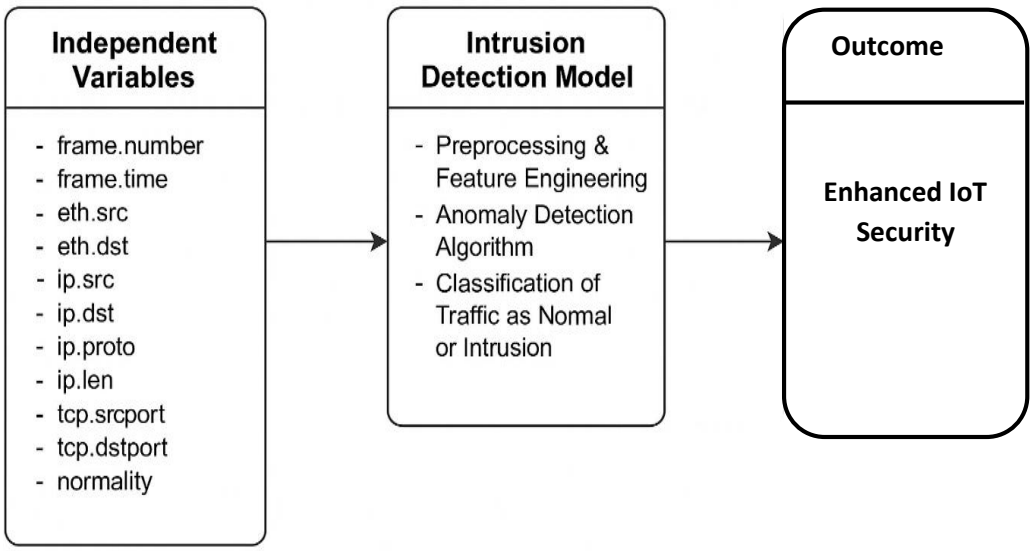
Source: *Researcher, 2024*

2.9 Conceptual Framework

The researcher endeavors to improve the security of IoT devices connected by implementing an Intrusion Detection System (IDS) specifically optimized for resource-constrained IoT devices. Independent variables as key parameters to driving this study are strategically chosen to influence and measure the desired outcomes. They provide rich context and behavioral indicators that enhance the accuracy and robustness of machine learning-based IDS models (Ioannou et al., 2017). They are frame.number, frame.time, frame.len, eth.src, eth.dst, ip.src, ip.dst, ip.proto, ip.len, tcp.len, tcp.srcport, tcp.dstport. The Dependent Variable is enhanced Security for IoT devices. A Model of an Intrusion Detection System (IDS) will be acting as mediating Variable.

Figure 2. 12

Conceptual Framework



Independent variables **Mediating Variable** **Dependent variable**

Source: *Researcher, 2024*

Independent Variables (IVs): Listed on the left are the network and packet attributes.

Intrusion Detection System (IDS): The mediating/moderating variable that processes the Independent Variables (IVs).

Enhanced Security for IoT Devices: The dependent variable, influenced through the IDS.

The arrows indicate the direction of influence from the independent variables through the IDS to the enhanced security for IoT devices. It is anticipated that the deployment of this system will directly impact the dependent variable - enhanced security of IoT devices. By introducing an optimized IDS, the research aims to observe an improvement in the overall security posture of IoT devices, providing a strong protection mechanism against potential intrusions. A foundational independent variable is the availability of literature on existing IDS technologies and IoT security frameworks. This variable influences the dependent variable by enriching the researcher's knowledge on IoT intrusion detection. The insights gained from the literature review will guide the study in understanding the current state of intrusion detection systems for IoT, thus informing the development and optimization of the proposed IDS. Defining parameters for intrusion detection on IoT devices constitutes a critical independent variable. The effectiveness of these parameters is expected to directly influence the dependent variable by enhancing of security. By establishing precise parameters, the research aims to tailor the intrusion detection process to the unique characteristics and constraints of IoT devices, optimizing the system's ability to detect and respond to potential threats. These fields have been explained as below: -

The variable frame.number represents the sequential position of a packet in the capture file. It serves as a unique identifier for each frame, helping to track the order of packets in network sessions, which is important when analyzing communication patterns or identifying unusual activity sequences.

The frame.time variable indicates the exact timestamp at which the packet was captured. This is vital for temporal analysis of network traffic, enabling the detection

of attacks that are time-dependent, such as synchronized distributed denial-of-service (DDoS) attacks or brute-force login attempts that occur during off-peak hours.

Another essential variable is frame.len, which measures the total size of a packet in bytes. Abnormal frame lengths, whether excessively large or unusually small, can indicate attempts to evade detection, perform fragmentation attacks, or deliver malicious payloads.

The variable eth.src refers to the Ethernet source address, which is the Media Access Control (MAC) address of the device that originally transmitted the packet. This identifier is crucial in detecting spoofing attacks or tracing the origin of suspicious traffic within a local network.

Similarly, eth.dst represents the Ethernet destination address—the MAC address of the intended recipient at the data link layer. Monitoring eth.dst can help uncover anomalies such as packets being redirected to unintended destinations, which may suggest man-in-the-middle attacks or ARP poisoning.

At the network layer, ip.src denotes the source IP address from which the packet originated. This variable is crucial for identifying the source of potentially malicious traffic, tracing botnet participants, or detecting spoofed IP addresses.

Conversely, ip.dst represents the destination IP address, which reveals the target of a communication. Abnormal destination patterns may indicate attempts to exfiltrate data, probe devices, or communicate with command-and-control servers.

Finally, ip.proto identifies the specific protocol being used at the IP layer, with common values including 6 for TCP, 17 for UDP, and 1 for ICMP. By analyzing this field, IDS models can detect the misuse of protocols, such as ICMP tunneling or UDP floods, and more effectively classify attack types based on protocol behavior.

Moving up the OSI model, ip.len denotes the total length of the IP packet, including both header and data. Anomalies in IP packet length—such as unusually large or malformed packets—can indicate attacks like fragmentation-based evasion or buffer overflow exploits.

On the transport layer, tcp.len measures the size of the TCP segment's payload, excluding headers. This variable helps identify suspicious patterns, such as TCP packets with zero payload used in SYN scanning, or excessively large payloads that could be transferring sensitive information during a data exfiltration attempt.

The tcp.srcport variable captures the source port number, indicating which port on the sender's machine initiated the connection. While source ports are often ephemeral and randomly assigned, a pattern of specific or rapidly changing source ports may signal port spoofing or botnet communication.

Meanwhile, tcp.dstport is the destination port number that points to the service being accessed on the target machine—such as port 80 for HTTP, 443 for HTTPS, or 22 for SSH. This is one of the most critical variables for intrusion detection, as unauthorized or abnormal access attempts to sensitive ports often signal an exploit attempt, malware communication, or reconnaissance activity.

The design of the intrusion detection model represents another independent variable. This variable is positioned to influence the dependent variable by impacting the real-time monitoring capability and accuracy of the intrusion detection system. A well-designed model is anticipated to enhance the system's efficiency, ensuring timely and accurate identification of security threats in the IoT environment. Simulated scenarios and test environments serve as independent variables in this study. They are designed to influence the dependent variable by providing a controlled environment to validate the performance metrics of the intrusion detection system. Through simulated

scenarios, the research aims to assess the system's efficacy in various situations, ensuring its reliability and effectiveness in real-world applications.

In summary, these independent variables collectively form the conceptual framework for this research, guiding the exploration of the research objectives. Their interactions with the corresponding dependent variables are instrumental in achieving the primary goal of enhancing the security of IoT devices connected via WLANs through the development and implementation of an Intrusion Detection System.

2.10 Research Gap

Traditional IDSs have been used to detect intrusion across varied IoT devices but none has taken serious attention to the resource constrained IoTs. Mechanisms like encryption and authentication are insufficient to protect IoT devices and therefore IDSs are necessary to enhance security. This is even emphasized by the fact that general Dataset like KDD has continued to be used to train these models, with none so far dedicated to IoT devices only. IDS can be considered as the last line of defense when other tools are broken. When it comes to deploying IDSs, need for scalable solutions that can operate efficiently within the resource constraints of IoT devices becomes a challenge.

2.11 Summary

The integration of IDS in IoT environments is crucial for safeguarding against an ever-evolving landscape of cyber threats. While significant progress has been made in developing effective IDS models, challenges such as data privacy, false positives, and real-time processing remain. Future research should leverage emerging technologies like federated learning, edge computing, blockchain and explainable AI to enhance IDS capabilities and ensure robust security for IoT networks. As IoT continues to grow, so too must our efforts to protect these interconnected systems from malicious actors.

Based on literature review, of late there has been an increase in intrusions in IoTs as they hugely get deployed in various domains hence exposing users' data by compromising on Confidentiality, Integrity and Authorization. Past researchers have expressed interest in coming up with solutions to address issues of intrusion. However, the intrusion detection systems developed have been inadequate and do not address specifically the resources constrained IoTs devices. IDSs developed in the past requires high processing power, huge storage, long power sustainability and strong inbuilt security mechanism.

CHAPTER THREE: RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents the methodology used to develop and evaluate an Intrusion Detection System (IDS) using a machine learning approach, specifically an Artificial Neural Network (ANN). It describes the research design, data sources, preprocessing techniques, model development process, evaluation metrics, and tools used. The aim is to provide a structured framework that supports the study's objectives and ensures reproducibility.

3.2 Research Design

The study adopted a mixed methodology that involved experimental and deductive research. This approaches were appropriate because they involved training and testing the ANN model on network traffic data to determine its effectiveness in detecting intrusions. The process includes dataset selection, data preprocessing, model training, validation, testing, and performance evaluation.

The choice of literature review was to give a summary of the literature related to the study inquiry so as to identify the gap that is to form the basis of the study topic. The deductive research approach is more specific one where one begins with the analysis of the theory and then the analysis of the results of the theory and as such it is considered more of a scientific approach (Mardiana, 2020). In the case of an inductive approach, the researcher starts from a specific point of view and moves to general aspects. Therefore, it is considered from the bottom up. Results obtained were used to form basis for the study as it identified the gap that existed.

3.3 Research Strategy

The model was developed using PyCharm (Vr2024.2.5) Integrated Development Environment (IDE) which provided tools for writing, debugging, and managing the

Python projects efficiently. The model developed was trained on detecting intrusions and being able to classify as either normal or anomaly traffic. The model performance was then evaluated using various performance metrics.

3.4 Model Implementation

This allows the IDS model to be tested comprehensively under conditions that mimic actual deployment environments. The hardware and software configurations, as well as the methodology and specific configurations employed are detailed.

3.4.1 Hardware components requirement

i The experiments were performed on a high-performance computing system to expedite the training process and enable efficient parallel computations.

ii CPU: 2 x Intel Xeon E5-2690 v4 (or equivalent) with 12 cores each

iii RAM: 64 GB DDR4

iv Storage: 2 x 1 TB SSD (RAID 1 for OS) + 4 x 2 TB SSD (RAID 10 for data storage)

v NIC: 2 x 10 Gigabit Ethernet NICs

vi GPU: NVIDIA Tesla T4 (if using machine learning)

vii Power Supply: Redundant power supplies

viii Cooling: For advanced cooling system with multiple fans or liquid cooling

3.4.2 Software components requirement

The researcher used several libraries installed to provides a robust framework for data manipulation, preprocessing, and visualization, training and building the deep learning models.

i Microsoft Windows 11Pro Operating System a platform on which other installations was to be based.

ii Integrated development environments (e.g., Visual Studio Code, PyCharm (Vr 2024.2.5)) for coding and debugging.

iii Support for relevant programming languages such as Python 3.12 which provided a robust framework for building and training deep learning models.

iv Installation of necessary libraries and frameworks for IDS development, including NumPy, Pandas, Scikit-learn, and Matplotlib for data manipulation, preprocessing, and visualization.

v Virtual Machines Support for VMs (e.g., VMware, VirtualBox, Packet Tracer) for testing different environments.

vi Wireshark for network traffic analysis.

vii High-speed Internet reliable and fast internet connection for downloading updates, datasets, and for remote collaboration.

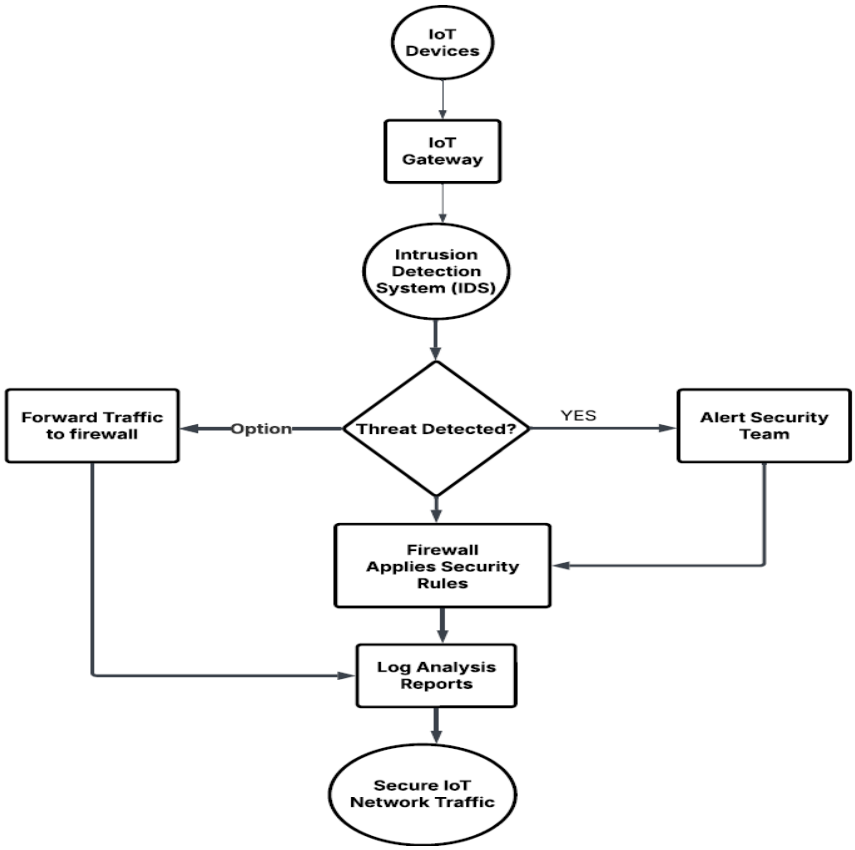
viii Configurable firewall to protect the development environment from external threats.

3.5 Model Experimental Setup

IoT Devices Communicate using IoT protocols like MQTT, CoAP and HTTP while IoT Gateway acts as a bridge between IoT devices and the network. The Intrusion Detection System (IDS) using ANN runs to detect unusual behavior in IoT traffic. If Threat are/is detected, Security Team is alerted and Firewall Rules updated and applied. In the situation that no attack is detected, traffic is forwarded to the firewall for log analysis.AT the very end of the process Iot network traffic is secured. Figure 3.3 shows the a flowchart for the performance of the model.

Figure 3. 1

Flowchart of the performance of the model



Source: *Researcher, 2024*

3.6 Model Algorithm

1. Start
2. Initialize IoT devices
3. Connect IoT devices to the IoT Gateway
4. Forward all incoming and outgoing network traffic to the Intrusion Detection System (IDS)
5. IDS analyzes the traffic
6. If IDS detects a threat, then:
    - Alert the Security Team

- Send the traffic to the Firewall for further action

7. Else (no threat detected):
    - Forward the traffic to the Firewall for standard processing
8. Firewall applies relevant security rules to the traffic
9. Log analysis and generate reports
10. Ensure the resulting traffic is secured
11. End

3.7 IDS Performance

The model is based on signature-based IDS which rely on known patterns of attacks. The anomaly-based systems monitors network traffic or system activity for deviations from established baselines. It works by establishing baseline through observing and analyzing normal network behaviors to establish a baseline of what is considered normal activity. This baseline can include factors such as typical network traffic patterns, system resource usage, user behavior and application usage. The model continuously monitors network traffic or system activity in real-time while looking for deviations or anomalies from the established baseline. These anomalies can indicate potential security threats such as suspicious network traffic, unauthorized access attempts or abnormal system behavior.

When the model detects an anomaly, it generates alerts or notifications to inform administrators or security personnel about the potential security issue. These alerts typically include information about the detected anomaly, such as the type of activity the affected system or network and the severity level of the anomaly. Based on the alerts generated by the model, administrators can investigate the detected anomalies further to determine if they represent genuine security threats. If a threat is confirmed,

appropriate response and mitigation measures can be taken to contain the threat, prevent further damage and restore the security of the network or system.

3.8 Dataset

The NSL-KDD dataset was used to introduce the anomalous traffic with various types of malicious activities (e.g., DDoS attacks, malware communication, unauthorized access) into the network to generate labeled datasets for training and testing the IDS model. The NSL-KDD dataset is not publicly available from the original source (University of New Brunswick) anymore but can be found on other platforms like Kaggle and IEEE DataPort.

The dataset for availability in the above platforms was downloaded from Kaggle after creating an account. NSL-KDD datasets has following characteristic as show in table 3.1. These features were used to train the model and are frame.number, frame.time, frame.len, eth.src, eth.dst, ip.src, ip.dst, ip.proto, ip.len, tcp.len, tcp.srcport, tcp.dstport, value and normality.

Table 3. 1

Characteristic of NSL-KDD dataset

Characteristic	**Description**	**Values**
Dataset Size	Number of records in the selected subset. Consideration of the implications of dataset size on model training and representativeness was considered.	KDDTrain+: 125, 973 records KDDTest+: 22,544 records
Number of Features	Number of attributes describing each network traffic record for the potential	41 features

	impact of dimensionality on model complexity.	
Normal Traffic Samples	Number of records representing normal network activity for the sufficient representation for the model to learn typical behavior.	67, 343 records
Attack Samples	Number of records representing the various attack types with adequate samples exist for each attack category to enable effective detection.	58, 630 records
Attack Categories	List the specific attack types present within the dataset while reflecting common threats and potential vulnerabilities for IoT devices.	DoS (e.g., Smurf, Teardrop) Probe (e.g., Nmap, Ipsweep) U2R (e.g., Buffer Overflow, Rootkit) R2L (e.g., Password Guessing, FTP-Write)
Class Distribution	Analyzes the proportion of normal vs. anomaly samples with significant imbalances that might impact model performance, warranting mitigation strategies.	Normal: 53.5%, Attack: 46.5%

Data Types	Categorizes features as numerical (continuous or discrete) or categorical (nominal or ordinal) so as to influence model selection and preprocessing.	Numerical (e.g., duration, src_bytes) Categorical (e.g., protocol_type, service, flag)
Feature Relevance	Indicate the most impactful features identified for intrusion detection for distinguishing normal vs. attack traffic.	Top features include: protocol_type, service, duration, flag, src_bytes, dst_bytes

Source: *Ranaweera & Sakuntala, 2022*

The NSL-KDD dataset, an improved version of the KDD CUP 99 dataset, is a recognized comparison for intrusion detection research (Saseed et al., 2023). The research utilized this datasets because of the following characteristics: -

3.8.1 Addressing KDD CUP 99 issues

The NSL-KDD dataset resolves significant shortcomings present in the original KDD CUP 99 dataset. Primarily, it eliminates redundant and duplicate records that could introduce bias during the training of machine learning models. This refinement ensures a more representative dataset for reliable model evaluation.

3.8.2 Wide range of attack types

NSL-KDD encompasses a diverse set of network attack categories: Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L) as summarized in Table 3.2. While not explicitly generated from IoT devices, these attacks hold significant relevance for IoT environments operating over WLANs. DoS attacks can disrupt

service availability, probes can map the network for vulnerabilities, and U2R/R2L attacks could compromise IoT devices. The dataset is managed by MIT Lincoln Laboratory. The data contains a standard group of data that includes a high range of intrusions in a IoT network environment. Each network connection consists of 41 features with the details of time and Window-based features and basic TCP features. In this dataset, the attacks are grouped into certain types and its percentages in the dataset as well as its attack name.

Table 3. 2

Representation of attacks in NSL-KDD dataset

No.	Attack Type	Ratio of Attacks	Attack Name
1.	Normal	20%	Normal
2.	Denial of Service (DoS)	79%	Neptune, Teardrop, Land, Back, Smurf, Pod
3.	U2R	0.01%	Buffer overflow, LoadModule, Perl, Rootkit
4.	R2L	0.19%	Guess password, Ftpwrite, Imap, Phf, Multihop, Probe, Warezmaster, Warezclient
5	Probe	0.8%	Portsweep, Ipsweep, Nmap, Satan

Source: *Ramadan & Yadav, 2020*

3.8.3 Community acceptance and benchmarking

The widespread adoption of NSL-KDD within the research community enables the objective comparison and benchmarking of the developed intrusion detection model against other related research endeavors. This facilitates a broader understanding of the model's performance contextually.

3.8.4 Dataset preparation

Dataset preparation plays a critical step in building any machine learning model, including an Artificial Neural Network (ANN). A well-prepared dataset ensures the model learns effectively and makes accurate predictions. The following is a step-by-step explanation of the dataset preparation process for the IDS model development.

Regarding data collection, a NSL-KDD Dataset was utilized. The dataset comprises of network traffic and attacks which was used for training and testing of the model. Data was cleaned by handling missing values, removing of rows and columns with too many missing entries, removing duplicate, removing inconsistencies and many more. Data Labeling was done considering this is a supervised learning to assign a label to the target output to each data instance. In this case of the research Intrusion detection was labelled as ‘0’ for Normal traffic or ‘1’ for an attack traffic. Relevant features for the IDS were selected and new ones created from the existing ones to better capture the traffic patterns.

Data Normalization was done by standardizing the range of values to ensure balanced learning. Data Splitting was done to comprise of training set, Testing set and Validation set. Training set was 70%, Validation set was 15% and was used to tune hyperparameters and avoid overfitting while Test set 15% was used to evaluate the final model's performance. One-Hot Encoding was used to convert categories to integers and

binary vectors like in the case of traffic protocols ['TCP', 'UDP', 'ICMP' to [1,0,0], [0,1,0], [0,0,1.

Handling Imbalanced Data was done to address situations of Oversampling and Undersampling. This reduced minority class and majority class. Synthetic Minority Over-Sampling Technique (SMOTE) was adopted to handle data imbalance. After the above processes, final output of dataset preparation was a ready data to feed into the ANN model.

3.9 Intrusion Detection Techniques

The model was trained on the normal traffic data and their ability to detect deviations caused by malicious activities tested.

3.9.1 Evaluation metrics

The researcher employed use of a number of evaluation metrics including detection Rate to measure the proportion of actual intrusions that are properly identified by the IDS, False Positive Rate to calculate the rate of benign activities incorrectly flagged as intrusions, False Negative Rate to Assess the rate of intrusions that the IDS fails to detect, Precision to calculate the accuracy of positive predictions, recall to balance detection accuracy and False Alarm rate to determine the proportion of benign events that are incorrectly classified as malicious.

3.9.2 Scenario simulation

Attack traffic was replayed in the simulated environment to evaluate the IDS performance under controlled conditions.

3.9.3 Continuous improvement

Feedback loop was incorporated from the detection results to continuously update and refine the IDS model.

3.10 Data Analysis

Dataset was selection based on the freely available NSL-KDD dataset while ensuring the dataset contains both normal and attack data and representing various types of intrusions. Dataset was interrogated and summarized to understand the structure, feature types, and distribution of classes. Distribution of normal vs. attack instances was checked to understand class imbalance. Missing values were handled so as to assist in deciding on a plan for dealing with missing values (e.g., removal or imputation. Relevant traits that contribute primarily to the forecast were selected and new features from existing ones created. Splitting the dataset into training and testing sets was done after which Artificial Neural Networks (ANN) model was selected. The developed robust IDS model was assessed using accuracy, precision, recall, F1-score, and ROC-AUC as performance metrics.

3.11 Model Evaluation Metrics

Multiple performance metrics were used to give a fuller picture of how well the model performed. They helped to determine whether the model was making accurate and useful predictions by understanding how effective the model distinguished between normal and malicious behavior.

Table 3. 3

Performance Metrics for IoT model

S/No	Indicator	Description
1.	Accuracy	Accuracy measures the overall correctness of the model's predictions, representing the proportion of correctly classified instances both normal and anomaly traffic out of the total number of instances.

2.	Precision	Precision focuses on the accuracy of positive predictions, calculating the proportion of true positives correctly identified attacks out of all instances predicted as positive.
3.	False Alarm Rate (FAR)	False Alarm rate refers to the proportion of benign events that are incorrectly classified as malicious.
4.	Recall	Recall, also known as sensitivity, measures the model's ability to identify all actual positive instances attacks.
5.	F1-score	The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance.
6.	ROC AUC	It measures the model's ability to distinguish between positive and negative classes.

Source: *Jude Chukwura Obi, 2023*

3.12 Tools and Technologies

The model was developed using Python 3.10 for coding, TensorFlow for building and training the ANN, Pandas and NumPy for data handling, Scikit-learn for preprocessing and metric calculation and Jupyter Notebook for experimentation and visualization.

3.13 Ethical Considerations

The dataset used in the model is publicly available and anonymized. No personal or sensitive user information was involved. Ethical standards were maintained by avoiding any use of offensive attack simulation or network penetration in live environments.

Ethical principles were strictly adhered to in this study. Since Intrusion detection contains crucial information about users, the researcher endeavored to maintain data privacy and limited himself to the data that only touched on intrusion.

The researcher sort approval from Meru University Institutional Research Ethics Review Committee (MIRERC) before proceeding with the data collection.

The researcher appreciated materials sourced from other researcher by properly acknowledging using the APA 7th edition citation style. To prevent unintentional plagiarism, the researcher used DrillBit software to check for plagiarism and improve on originality.

The researcher ensured that the data to be analyzed and reported in the research was through openness and honesty. Access to the data was restricted to only the researcher using password and after data use the disk drive were permanently deleted.

CHAPTER FOUR: RESEARCH RESULTS AND DISCUSSION

4.1 Overview

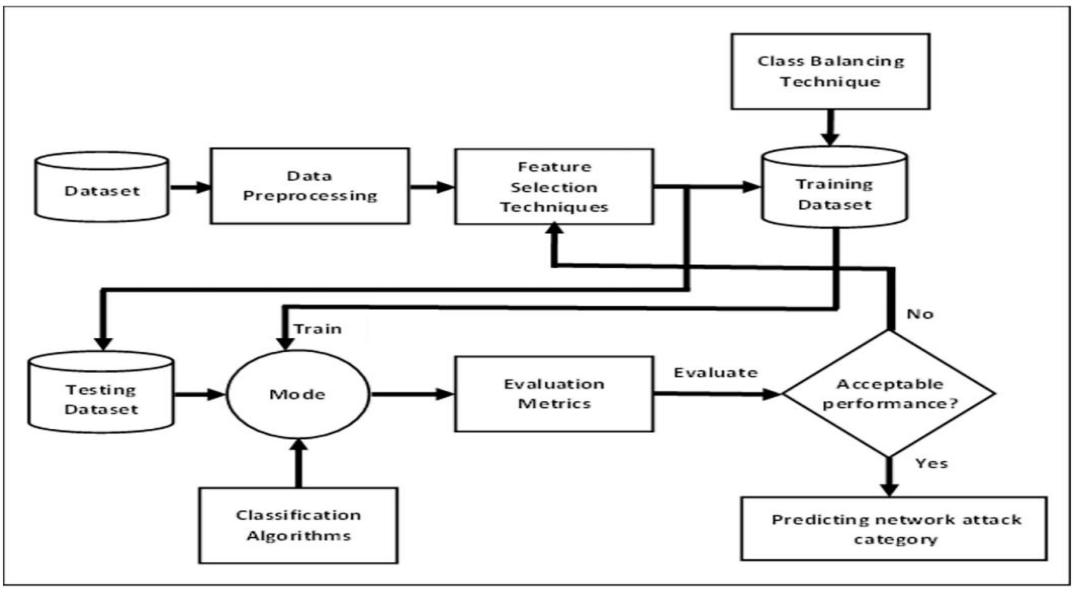
This chapter presents the results acquired from the experimental evaluation of the developed intrusion detection model for IoT devices, along with detailed analysis and discussion. The model's performance is assessed using various metrics to determine its ability to detect cyberattacks in a resource-constrained environment. First, the dataset used and the experimental setup are described. Subsequently, the results are presented and analyzed, considering their implications for real-world security in the context of enhancing security for IoT applications.

4.2 Model Design

A comprehensive design outline includes the architecture, functional modules, and methodologies for developing an effective IDS for IoT environments. The model encompasses data collection, preprocessing, feature extraction, various detection methods, and an alert system. By integrating these components, the IDS can effectively detect and respond to security threats in IoT networks. However, regular updates and continuous monitoring are essential to ensure the system remains effective against evolving threats.

Figure 4. 1

Design for the developed Intrusion Detection Model



Source: *Patel, D., & Bhavsar, M., 2019*

Figure 4.1 shows a flowchart that represents the design process for developing an Intrusion Detection System (IDS) model, specifically for detecting and categorizing network attacks using machine learning techniques. Below is an explanation of each step in the process, along with examples of techniques and algorithms that were used at each stage:

STEP 1: Dataset: NSL-KDD an improved version of KDD99 was used to train the model.

STEP 2: Data Preprocessing: One-Hot Encoder was used for Encoding categorical features and converts categorical variables into a format that can be provided to machine learning algorithms to do prediction.

STEP 3: Random-Forest-Classifier was adopted for Feature Selection Techniques. This provided a powerful, flexible and robust solution for intrusion detection, able to handle complex data while reducing overfitting.

STEP 4: Class Balancing Technique: Oversampling techniques Synthetic Minority Over-Sampling Technique (SMOTE) was used. This is because Network attack datasets are often imbalanced, where normal traffic dominates over attack samples. Dataset split to maintain the class distribution across all subsets (Hasan et al., 2022).

STEP 5: Model Training using Supervised learning algorithms to trained and learn patterns from the labeled training data and make predictions about new, unseen data as whether network traffic is normal or an attack.

STEP 6: Evaluation Metrics used are Accuracy, Precision, Recall, F1-Score & ROC-AUC Curve

STEP 7: Acceptable Performance Check: If yes proceed else do adjustments of feature selection technique and class balancing technique.

STEP 8: Predicting Network Attack Category and then deploy to detect intrusion.

4.3 ANN Model Architecture

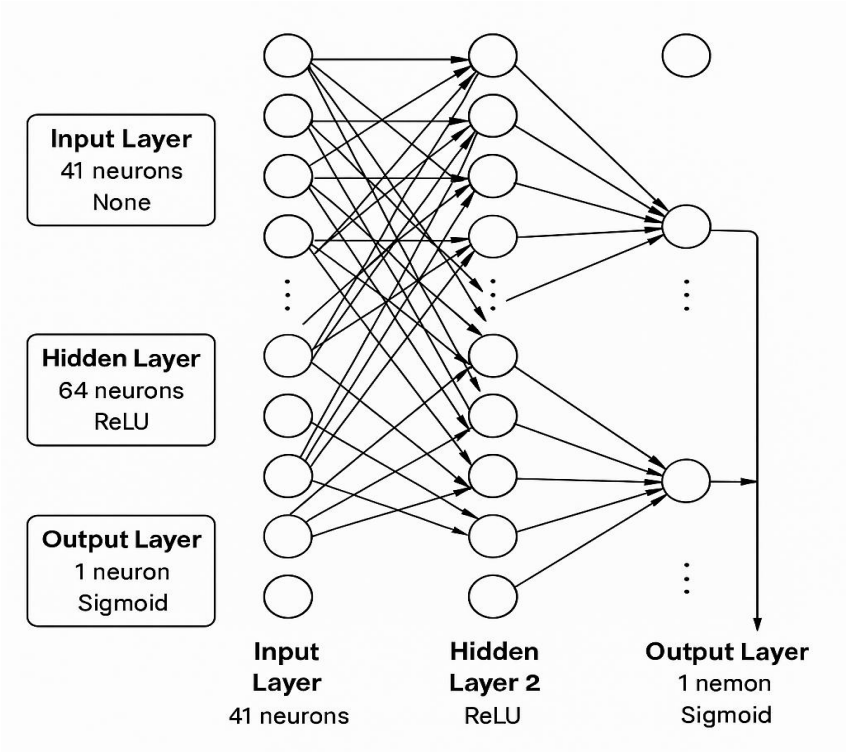
The Artificial Neural Network (ANN) architecture consists of four main components, the input layer, two hidden layers and an output layer, each with interconnected neurons used for classification of intrusion detection as illustrated in figure 4.2. Input data enters the network through the input layer after which the data is processed and transformed through two hidden layers using ReLU. Prediction is made in the output layer using a sigmoid activation function.

In this research ANN quantifies the loss function that determines the difference between predicted and actual outputs during training. ANN learns by minimizing this loss function using an Adam optimizer by adjusting the weights through backpropagation. Several training details have been considered to ensure the model is accurate, generalizable and efficient. Training details like Data preprocessing (Feature Encoding and Class Balancing), model hyper parameter tuning (learning rate, epoch, optimizer)

regularization technique (L2 regularization, early stopping), model evaluation metrics (accuracy, recall, precision, f1-score and ROC-AUC, training strategy (Data splitting) have been explained.

Figure 4. 2

ANN model architecture



Source: *Sammany et al., 2007*

4.4 One-Hot-Encoder Data Preprocessing

One-Hot Encoding was selected as the preferred method for preprocessing categorical variables within the dataset. The NSL-KDD dataset used for training the Intrusion Detection System (IDS) model contains several categorical features, such as protocol_type, service, and flag. These variables consist of non-numeric values, which cannot be directly processed by machine learning algorithms such as Artificial Neural Networks (ANNs), which require numerical input.

One-Hot Encoding addresses this challenge by converting each unique category within a feature into separate binary columns, where a value of '1' indicates the presence of a specific category and '0' indicates its absence. For instance, if the protocol_type feature contains values such as TCP, UDP, and ICMP, One-Hot Encoding transforms this into three separate columns representing each protocol type. This approach ensures that the model does not assign any ordinal relationships among the categories, as would be the case with label encoding, which might incorrectly imply that one category is greater or less than another.

The choice of One-Hot Encoding is also justified by its compatibility with ANN models. Neural networks are sensitive to the structure and scale of input data, and binary features encoded through One-Hot Encoding fall within a normalized range that supports efficient learning. This method improves model interpretability and learning accuracy by ensuring that all categories are treated equally, without introducing misleading weight or importance to any category.

Despite the possibility of increased dimensionality, particularly for features with a high number of unique categories like service, this drawback was mitigated by removing infrequent categories and applying feature selection techniques to maintain a manageable input size. Furthermore, neural networks are well-suited to handle high-dimensional data, making One-Hot Encoding a practical and effective choice for this study because of the key strengths as shown in figure 4.3.

Table 4. 1

One-Hot Encoder data preprocessing features

No	Strength	Narrative
1.	Handling Categorical Data	This is by effectively transforms categorical data into a numerical format, which is necessary for most machine learning algorithms that require numerical input.
2.	Enhanced Detection Capabilities	By accurately representing categorical data, one-hot encoding helps in better capturing the trends and inconsistencies within the data, which is needful for identifying potential intrusions.
3.	Efficiency	One-hot encoded data is sparse, meaning most of the values are zeros. This sparsity can be leveraged by certain algorithms and data structures to improve computational efficiency.
4.	Interpretability	One-hot encoded features are more interpretable because each binary feature represents the presence or absence of a specific category. This can aid in understanding and explaining the model's predictions.
5.	Flexibility	One-Hot Encoder can handle both nominal and some ordinal categorical data, making it a

		versatile tool in preprocessing pipelines for IDS development.
6.	Model Compatibility	Many machine learning algorithms, such as logistic regression, decision trees, and neural networks, perform better with one-hot encoded features because it allows them to treat each category independently.

Source: *Researcher, 2024*

4.5 Model Development

4.5.1 NSL-KDD Dataset preprocessing

The following preprocessing steps were performed on the NSL-KDD dataset before using it for model training and evaluation to yield data as shown in figure 4.3.

i **Data Cleaning:** Removal of inconsistent, incomplete, or erroneous records to guarantee data quality and avoid misleading patterns during training.

ii **Normalization:** Numerical features were scaled to a common range (e.g., 0 to 1) using techniques such as min-max scaling or standardization. This normalization step enhances model convergence during training, particularly for gradient-descent-based algorithms.

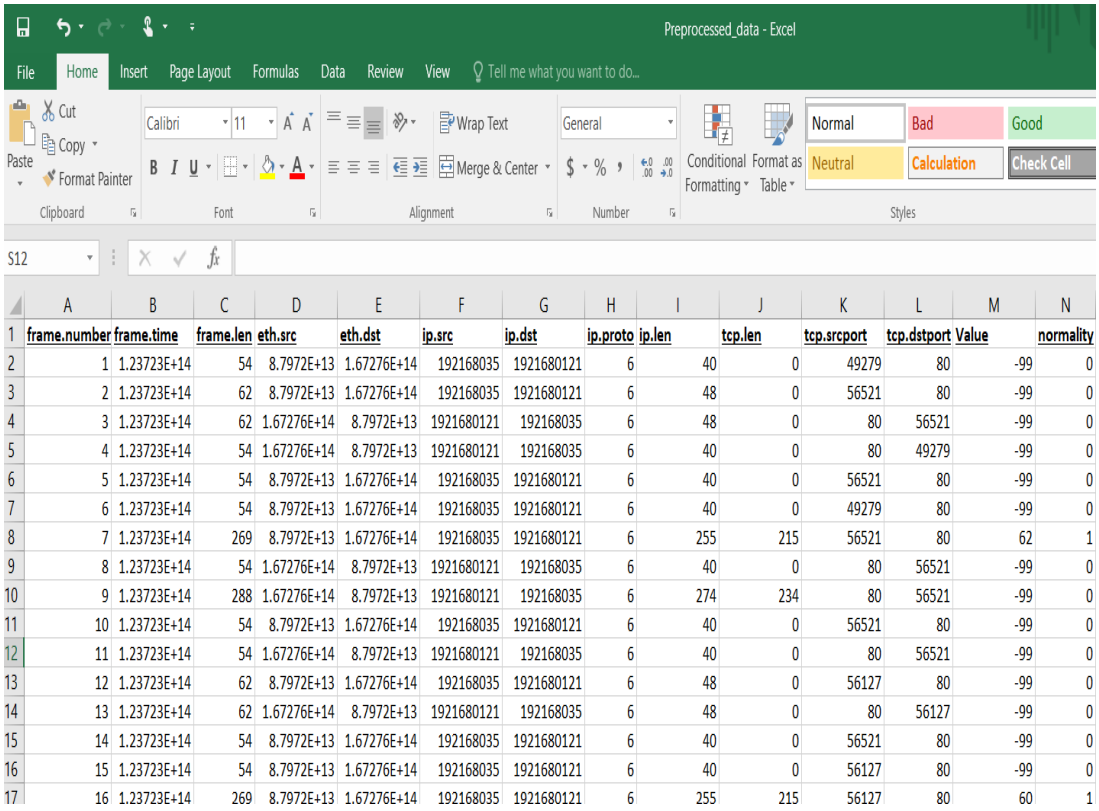
iii **One-hot Encoding:** Categorical features were transformed into a numerical representation suitable for machine learning algorithms. One-hot encoding creates new binary features for each unique category, ensuring these qualitative features are effectively interpreted by the model.

iv **Feature Selection:** Feature selection techniques were explored to potentially reduce the dimensionality of the dataset and enhance model performance. Random-Forest-Classifier provides a powerful, flexible, and robust solution for

intrusion detection, able of handling complex data, reducing overfitting, and offering insights into feature importance, making it an excellent choice for IDS development.

Figure 4. 3

Showing Preprocessed data



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	frame.number	frame.time	frame.len	eth.src	eth.dst	ip.src	ip.dst	ip.proto	ip.len	tcp.len	tcp.srcport	tcp.dstport	Value	normality
2	1	1.23723E+14	54	8.7972E+13	1.67276E+14	192168035	1921680121	6	40	0	49279	80	-99	0
3	2	1.23723E+14	62	8.7972E+13	1.67276E+14	192168035	1921680121	6	48	0	56521	80	-99	0
4	3	1.23723E+14	62	1.67276E+14	8.7972E+13	1921680121	192168035	6	48	0	80	56521	-99	0
5	4	1.23723E+14	54	1.67276E+14	8.7972E+13	1921680121	192168035	6	40	0	80	49279	-99	0
6	5	1.23723E+14	54	8.7972E+13	1.67276E+14	192168035	1921680121	6	40	0	56521	80	-99	0
7	6	1.23723E+14	54	8.7972E+13	1.67276E+14	192168035	1921680121	6	40	0	49279	80	-99	0
8	7	1.23723E+14	269	8.7972E+13	1.67276E+14	192168035	1921680121	6	255	215	56521	80	62	1
9	8	1.23723E+14	54	1.67276E+14	8.7972E+13	1921680121	192168035	6	40	0	80	56521	-99	0
10	9	1.23723E+14	288	1.67276E+14	8.7972E+13	1921680121	192168035	6	274	234	80	56521	-99	0
11	10	1.23723E+14	54	8.7972E+13	1.67276E+14	192168035	1921680121	6	40	0	56521	80	-99	0
12	11	1.23723E+14	54	1.67276E+14	8.7972E+13	1921680121	192168035	6	40	0	80	56521	-99	0
13	12	1.23723E+14	62	8.7972E+13	1.67276E+14	192168035	1921680121	6	48	0	56127	80	-99	0
14	13	1.23723E+14	62	1.67276E+14	8.7972E+13	1921680121	192168035	6	48	0	80	56127	-99	0
15	14	1.23723E+14	54	8.7972E+13	1.67276E+14	192168035	1921680121	6	40	0	56521	80	-99	0
16	15	1.23723E+14	54	8.7972E+13	1.67276E+14	192168035	1921680121	6	40	0	56127	80	-99	0
17	16	1.23723E+14	269	8.7972E+13	1.67276E+14	192168035	1921680121	6	255	215	56127	80	60	1

Source: *Researcher, 2024*

A total of 14 features were used to train the model. These dataset features are commonly used in network-based intrusion detection for detecting abnormal patterns in traffic and training machine learning models. Description of the dataset used is as shown in Table 4.2.

Table 4. 2

Description of Dataset features

No.	Feature Name	Description
1.	frame.number	Sequential number of the captured frame (packet) in a PCAP or traffic capture file. Useful for tracking packet order.
2.	frame.time	Timestamp indicating when the packet was captured. Helps analyze timing-related behaviors (e.g., burst attacks).
3.	frame.len	Total length (in bytes) of the frame. Can indicate unusually large or small packets.
4.	eth.src	Ethernet source MAC address. Identifies the sending device on the local network.
5.	eth.dst	Ethernet destination MAC address. Identifies the receiving device on the local network.
6.	ip.src	Source IP address. Indicates where the packet originated. Useful for identifying malicious senders.
7.	ip.dst	Destination IP address. Shows where the packet is headed. Can help detect targeting behavior.
8.	ip.proto	IP protocol number (e.g., 6 = TCP, 17 = UDP, 1 = ICMP). Important for protocol-specific analysis.
9.	ip.len	Length of the IP packet. Aids in detecting anomalies or fragmented packets.
10.	tcp.len	Length of TCP segment data (excluding headers). Can help spot abnormal payload sizes.

11.	tcp.srcport	TCP source port. Indicates the sending process on the source machine.
12.	tcp.dstport	TCP destination port. Reveals the service being accessed (e.g., 80 = HTTP, 22 = SSH).
13.	Value	Likely a computed value or metric associated with the packet (could be entropy, score, etc. depending on context).
14.	normality	Target label - typically a binary or categorical value indicating whether the traffic is normal or anomalous (e.g., attack).

Source: *Ranaweera & Sakuntala, 2022*

Data preprocessing sets the foundation for accurate and efficient data analysis and machine learning models. It involved transforming raw data into a clean and usable format as indicated in figure 4.4. This enhances model performance and also ensures that the results are reliable and interpretable.

Figure 4. 4

Illustration of preprocessed data loaded successfully

Preprocessed data loaded successfully.

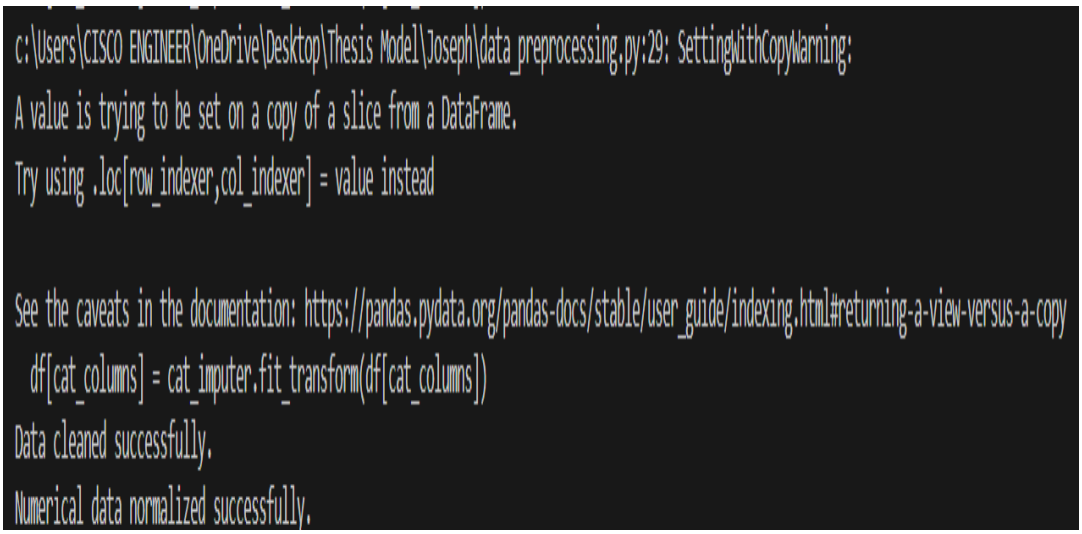
	flow_duration	Header_Length	Protocol Type	Duration	Rate	...	Radius	Covariance	Variance	Weight
0	0.000000	54.00	6.00	64.00	0.329807	...	0.000000	0.000000	0.00	141.55
1	0.000000	57.04	6.33	64.00	4.290556	...	4.010353	160.987842	0.05	141.55
2	0.000000	0.00	1.00	64.00	33.396799	...	0.000000	0.000000	0.00	141.55
3	0.328175	76175.00	17.00	64.00	4642.133010	...	0.000000	0.000000	0.00	141.55
4	0.117320	101.73	6.11	65.91	6.202211	...	32.716243	3016.808286	0.19	141.55
...	...	...	...	...	...	...	...	...	...	...
1048570	1.391925	108.00	6.00	64.00	1.437685	...	0.000000	0.000000	0.00	141.55
1048571	0.000000	2.14	46.70	65.91	0.000000	...	7.497140	469.190222	0.06	141.55
1048572	0.132971	30847.00	17.00	64.00	5978.034950	...	0.000000	0.000000	0.00	141.55
1048573	0.000000	54.00	6.00	64.00	25.672981	...	0.000000	0.000000	0.00	141.55
1048574	128.443556	4264.30	7.10	98.80	13.640648	...	162.030508	13140.079760	1.00	244.60

Source: *Researcher, 2024*

Normalization is an essential step that can lead to better, more reliable, and interpretable models. Different features in a dataset may have different units, scales, or magnitudes. Normalization standardizes these features, ensuring they are on a comparable scale. This consistency is vital for many algorithms that rely on machine learning and training. Many machine learning models, such as k-nearest neighbors (KNN), support vector machines (SVM), and neural networks used in this model, perform better when the input features are normalized like as shown in figure 4.5. It ensures that the model's performance is not biased towards features with larger scales.

Figure 4. 5

Illustration of Data normalized successfully



Source: *Researcher, 2024*

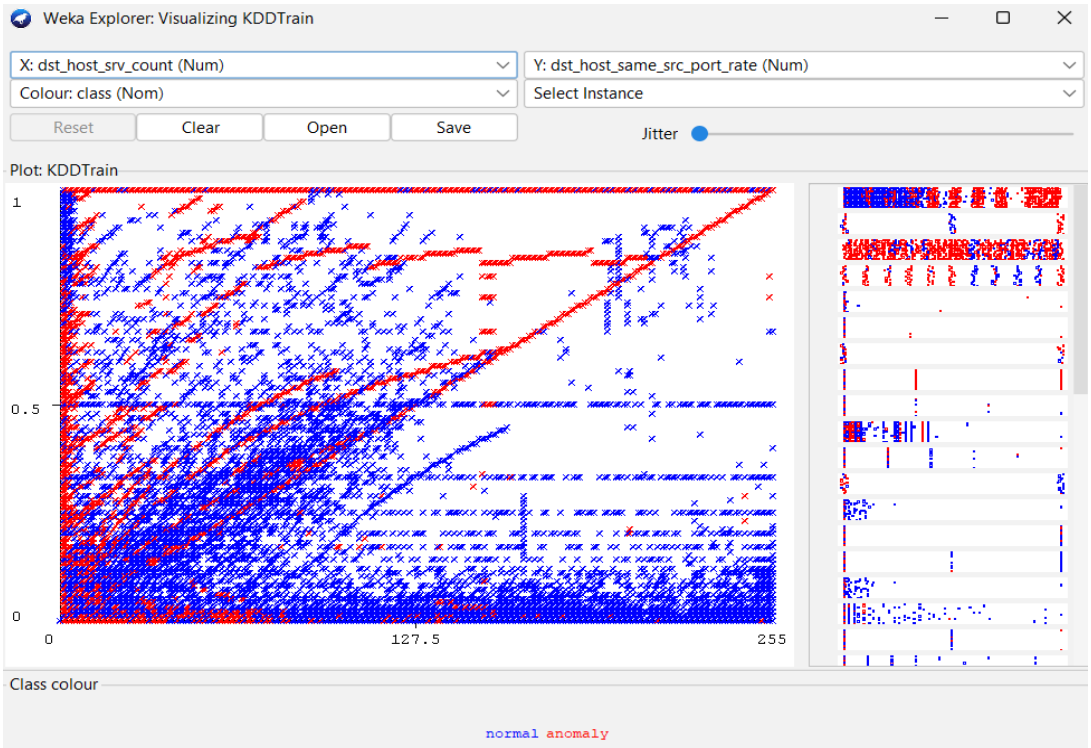
4.5.2 NSL-KDD Dataset analysis

WEKA (Waikato Environment for Knowledge Analysis) is a comprehensive suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. It is designed for data mining and analysis, providing tools for data preprocessing, classification, regression, clustering, association rules, and visualization. The scatter plot visualization has following features, the X-axis showing

Number of connections to the same service on the destination host and Y-axis showing the Rate at which the same source port is used for connections to the destination host. The different color coding, Blue represents normal network traffic and Red represents anomalous (potential attack) traffic. Each point of the scatter plot represents a network connection. WEKA knowledge analysis illustrates normal and anomaly traffic graphically for NSL-KDD dataset as shown in figure Figure 4.6.

Figure 4. 6

Illustration of normal and anomaly traffic



Source: *Aher, S. B., & Lobo, L. M. R. J., 2011*

4.6 Model Training Methodology

The overall methodology for training and testing the intrusion detection model involved the following steps: -

Dataset Split: The NSL-KDD dataset was split into training (70%), validation (15%), and testing (15%) subsets using stratified sampling to maintain the class distribution

across all subsets. This is to enable effective model training, tuning, and unbiased evaluation while mitigating overfitting and providing robustness. The 70% Training Subset provides adequate training data that ensures that the model has enough data to learn from. The more data the model sees during training, the better it can understand the patterns and relationships in the data, leading to a more robust model.

The 15% Validation Subset is used to tune hyperparameters and make decisions about model architecture. By evaluating the model on this subset, researcher can optimize it without overfitting to the training data. The validation set provides a checkpoint to see how well the model generalizes to unseen data before making final adjustments. The 15% Testing Subset is reserved for final evaluation after training and validation to provide an unbiased assessment of the model's performance on unseen data, ensuring that the model's accuracy and other performance metrics are reliable. By keeping the training, validation, and testing subsets separate, the risk of overfitting is minimized. The model is trained on one set of data, validated on another, and tested on a third, ensuring that it generalizes well to new data.

Splitting the dataset ensures that performance metrics (accuracy, precision, recall, F1-score) are calculated based on data that the model hasn't seen during training. This makes the metrics more reliable indicators of real-world performance. Splitting ensures that the model is trained effectively, tuned properly, and evaluated accurately, leading to a robust and reliable IDS. This approach helps balance the need for a large training set to learn from, a validation set to optimize the model, and a test set to evaluate its final performance. Figure 4.7 shows a code snippet regarding data splitting.

Figure 4. 7

Code snippet showing data split

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load NSL-KDD dataset (assuming it's in CSV format)
data = pd.read_csv("NSL_KDD_Dataset.csv")

# Stratified split into training (70%), validation (15%), and testing (15%) sets
train_data, temp_data = train_test_split(data, test_size=0.3, stratify=data['label'])
val_data, test_data = train_test_split(temp_data, test_size=0.5, stratify=temp_data['label'])

# Save the splits as CSV files
train_data.to_csv("train.csv", index=False)
val_data.to_csv("val.csv", index=False)
test_data.to_csv("test.csv", index=False)
```

Source: *Researcher, 2025*

Model Architecture: The intrusion detection model employed a deep neural network architecture consisting of an input layer, three fully connected hidden layers with 128, 64, and 32 neurons respectively, and an output layer with a softmax activation function for multi-class classification (Aldweird et al., 2020). This deep neural network (DNN) architecture is well-suited for intrusion detection because it can learn complex patterns and relationships in the data that simpler models might miss. The multiple hidden layers allow the network to build hierarchical representations of the data, making it possible to detect subtle and sophisticated intrusion attempts. This is illustrated in figure 4.8.

Figure 4. 8

Code snippet showing model Architecture

```
+ Code  + Text                                                                    Connect

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

# ... (Load train, val, test data from CSV files)

# Define the model architecture
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes is the number of attack categories
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Save model summary
with open("model_summary.txt", "w") as f:
    model.summary(print_fn=lambda x: f.write(x + '\n'))

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model and log history
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val), callbacks=[early_stopping])
pd.DataFrame(history.history).to_csv("training_log.csv", index=False)
```

Source: *Researcher, 2025*

Training Process: The model was trained using the Adaptive Moment Estimation (Adam optimizer) and the categorical cross-entropy loss function. L2 regularization was applied to the weights to prevent overfitting. L2 regularization (Ridge Regularization or Tikhonov Regularization) is a technique used in machine learning to prevent overfitting by adding a penalty to the loss function. The penalty is proportional to the square of the magnitude of the coefficients (weights) of the model. This encourages the model to keep the coefficients small, which can lead to a simpler model that generalizes better to new data. The first term in the loss function is the residual sum of squares (RSS), which measures the fit of the model to the data. The second term is

the Ridge Regression (L2) penalty, which adds the squared values of the coefficients. By minimizing this combined loss function, the model balances fitting the data well (low RSS) with keeping the model coefficients small (low L2 penalty). L2 regularization is commonly used in various machine learning algorithms, such as linear regression, logistic regression, and neural networks, to improve their generalization performance. L2 technique is used to prevent overfitting in linear regression models by adding a penalty equal to the sum of the squared coefficients to the loss function. In mathematical terms, for a linear model, the loss function with L2 regularization is given by the formula 4.1.

Formula 4. 1

Loss function with L2 regularization

Loss function formulae

$$J(\theta) = \sum_{\{i=1\}}^{\{m\}\left(h_{\theta\left(x^{\{(i)\}}\right)} - y^{\{(i)\}}\right)^2} + \lambda \backslash sum_{\{j=1\}}^{\{n\}\theta_j^2}$$

J(θ): The cost function (objective to minimize).

m: Number of training examples.

h_θ($x^{(i)}$): The hypothesis (predicted value) for the i-th training example.

$y^{(i)}$: The actual output for the i-th training example.

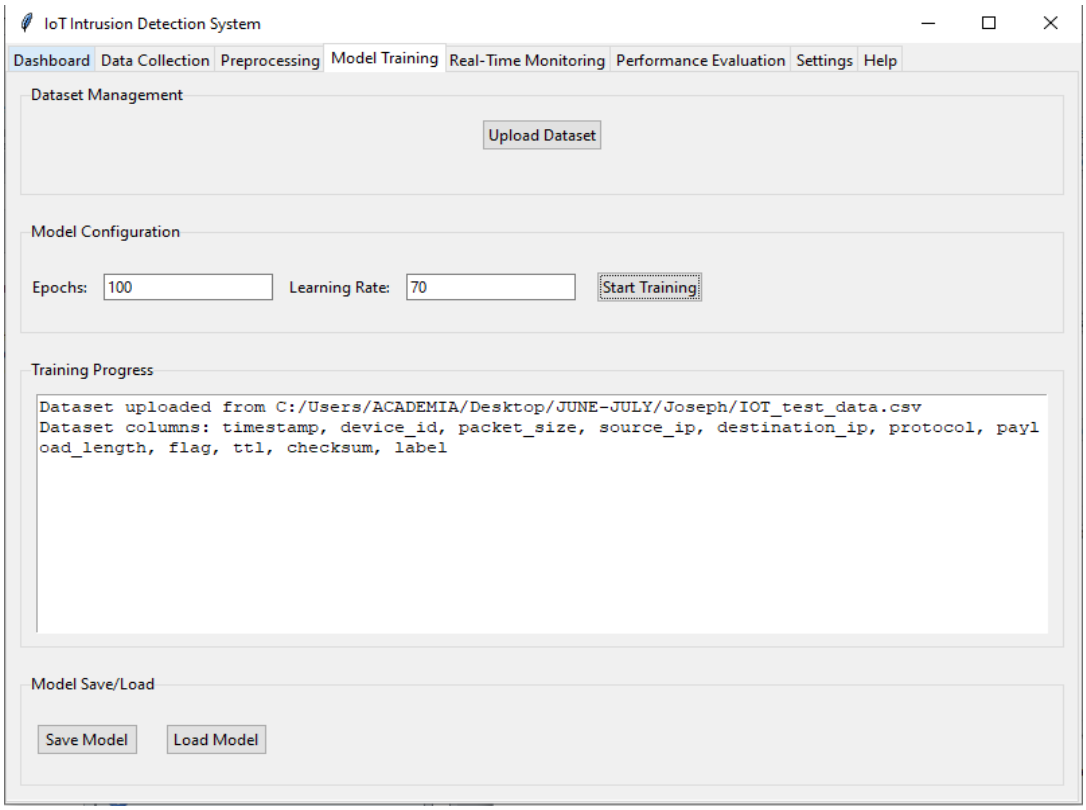
θ_j: Model parameters (weights).

λ: Regularization parameter (controls the strength of the penalty).

The second term $\lambda \sum_{j=1}^{n} \theta_j^2$ is the regularization term that penalizes large values of θ_j to avoid overfitting.

Figure 4. 9

Model training Interface

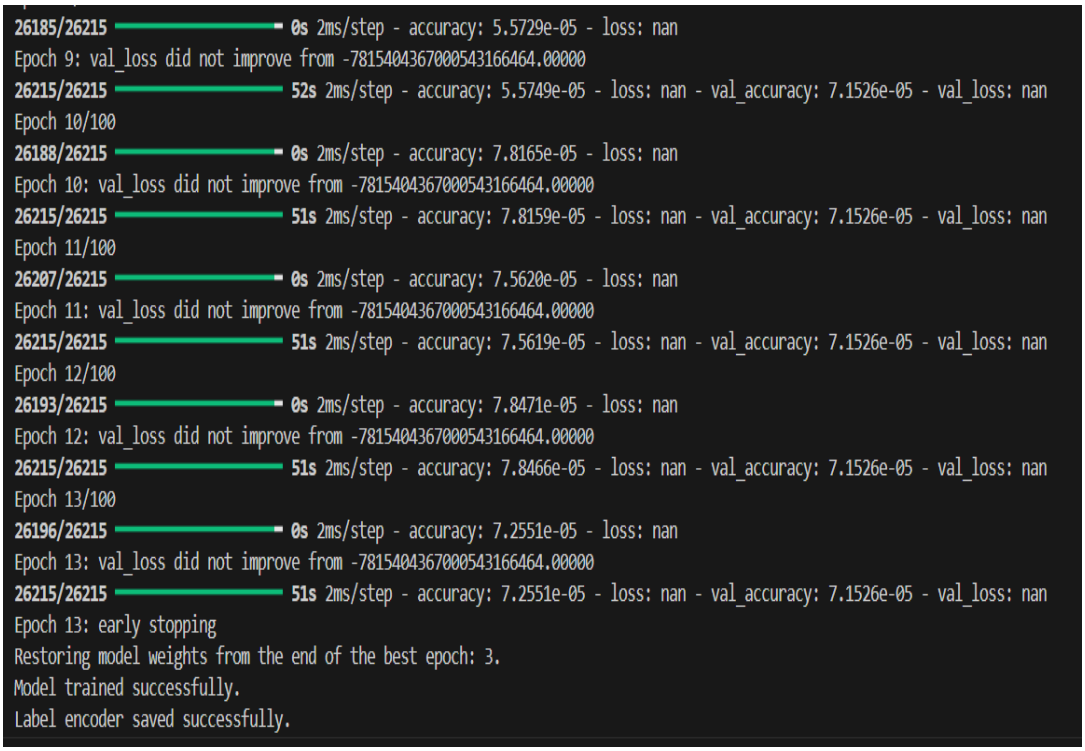


Source: *Researcher, 2025*

An epoch is one complete pass through the entire training dataset. When training a model, especially neural networks, the data is often too large to feed into the model all at once. Instead, the data is split into smaller batches, and the model is trained on these batches sequentially. Multiple epochs are usually run during training because one pass through the data (one epoch) is often not enough for the model to learn the underlying patterns in the data. The model needs multiple exposures to the data to fine-tune its parameters and improve its performance on unseen data. The number of epochs is a hyperparameter that you can set before training your model. Figure 4.9 shows model interface displaying Epochs and learning rate above showing capability of being to save a trained model for just loading and using without raining afresh.

Figure 4. 10

Epochs with an early stopping mechanism



Source: *Researcher, 2025*

Figure 4.10 show training was carried out for 100 epochs with an early stopping mechanism to halt training if the validation loss did not improve for 10 consecutive epochs.

The learning rate controls how much the model's weights are adjusted during training in response to the estimated error each time the model weights are updated. It determines the step size at each iteration while moving toward a minimum of the loss function. Key points being:

High Learning Rate: A high learning rate can make the training process faster by taking larger steps. However, it may also cause the model to overshoot the minimum of the loss function, leading to instability or failure to converge.

Low Learning Rate: A low learning rate makes the training process slower because it takes smaller steps, but it allows the model to move precisely approaching the minimum. However, it might also lead to getting stuck in local minima or require more training time.

Adjusting the learning rate appropriately can significantly impact the training efficiency and performance of a machine learning model. Finding the best learning rate involves a mix of initial guesses, empirical testing, and adjusting based on model performance. Starting with a learning rate range test is a good strategy, followed by fine-tuning with learning rate schedulers or adaptive optimizers.

Hyperparameter Tuning: A grid search approach was employed to tune the model's hyperparameters, including the learning rate (values ranging from 0.001 to 0.1), batch size (32, 64, 128), and the L2 regularization coefficient (0.01, 0.001, 0.0001) (Saeed et al., 2023). Hyperparameter tuning optimizes the parameters of a machine learning model that are not learned from the training data. These parameters, known as hyperparameters, are set before the learning process begins and can significantly impact the model's performance. The goal of hyperparameter tuning is to find the optimal set of hyperparameters that yield the best performance on a given task as shown in figure 4.11.

Figure 4. 11

Tuning the model's hyperparameters

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

# ... (Define model_builder function to create the model)

# Define hyperparameter grid
param_grid = {
    'batch_size': [32, 64, 128],
    'epochs': [50, 100],
    'optimizer': ['adam', 'rmsprop'],
    # ... other hyperparameters
}

# Create KerasClassifier
model = KerasClassifier(build_fn=model_builder, verbose=0)

# Grid search with cross-validation
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)
grid_result = grid.fit(X_train, y_train)

# Save best parameters
with open("best_params.txt", "w") as f:
    f.write(str(grid_result.best_params_))
```

Source: *Researcher, 2025*

Cross-Validation: To ensure robust model evaluation, a 5-fold stratified cross-validation technique was applied during the hyperparameter tuning phase (Niyaz et al., 2015). The technique involved dividing the data into subsets, training the model on some subsets while validating it on others, and repeating this process several times to ensure robustness and minimize overfitting as shown in figure 4.11.

Figure 4. 12

Model performance evaluation

```
from sklearn.metrics import classification_report, roc_auc_score


# Classification report
report = classification_report(y_test, y_pred, target_names=class_names, output_dict=True)
pd.DataFrame(report).transpose().to_csv("evaluation_results.csv")

# ROC AUC for each class
roc_auc = {}
for i in range(num_classes):
    roc_auc[class_names[i]] = roc_auc_score(y_test[:, i], y_pred_proba[:, i])
pd.DataFrame(roc_auc, index=[0]).to_csv("roc_auc.csv")
```

Source: *Researcher, 2025*

Performance Evaluation: The model's performance was evaluated using a comprehensive set of metrics, including accuracy, precision, recall, F1-score, and area under the receiver operating characteristic (ROC) curve for each attack category (Revathi & Malathi, 2013) as shown in figure 4.12.

Early Stopping: To prevent overfitting and improve generalization, an early stopping mechanism was implemented, monitoring the validation loss and halting training if no improvement was observed for 10 consecutive epochs (Prechelt, 1998).

Class Weights: To address the class imbalance in the NSL-KDD dataset, class weights were applied during training, assigning higher weights to the minority classes. This is shown in figure 4.13.

Figure 4. 13

Assignment of class weights

```
from sklearn.utils import class_weight

# Calculate class weights
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = dict(enumerate(class_weights))

# Save class weights
pd.DataFrame(class_weights_dict, index=[0]).to_csv("class_weights.csv")
```

Source: *Researcher, 2025*

Data Augmentation: To increase the diversity of the training data and improve the model's robustness, various data augmentation techniques were employed, including random noise injection, feature scaling, and synthetic minority oversampling (Shorten & Khoshgoftaar, 2019) as shown in figure 4.14.

Figure 4. 14

Training data Augmentation

```
from sklearn.utils import shuffle
from imblearn.over_sampling import SMOTE
# Apply SMOTE for oversampling minority classes
smote = SMOTE(sampling_strategy='auto')
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Combine original and augmented data
X_train_augmented = np.concatenate((X_train, X_train_resampled))
y_train_augmented = np.concatenate((y_train, y_train_resampled))

# Shuffle the augmented data
X_train_augmented, y_train_augmented = shuffle(X_train_augmented, y_train_augmented)

# Save augmented training data
augmented_train_data = pd.DataFrame(X_train_augmented, columns=X_train.columns)
augmented_train_data['label'] = y_train_augmented
augmented_train_data.to_csv("augmented_train.csv", index=False)
```

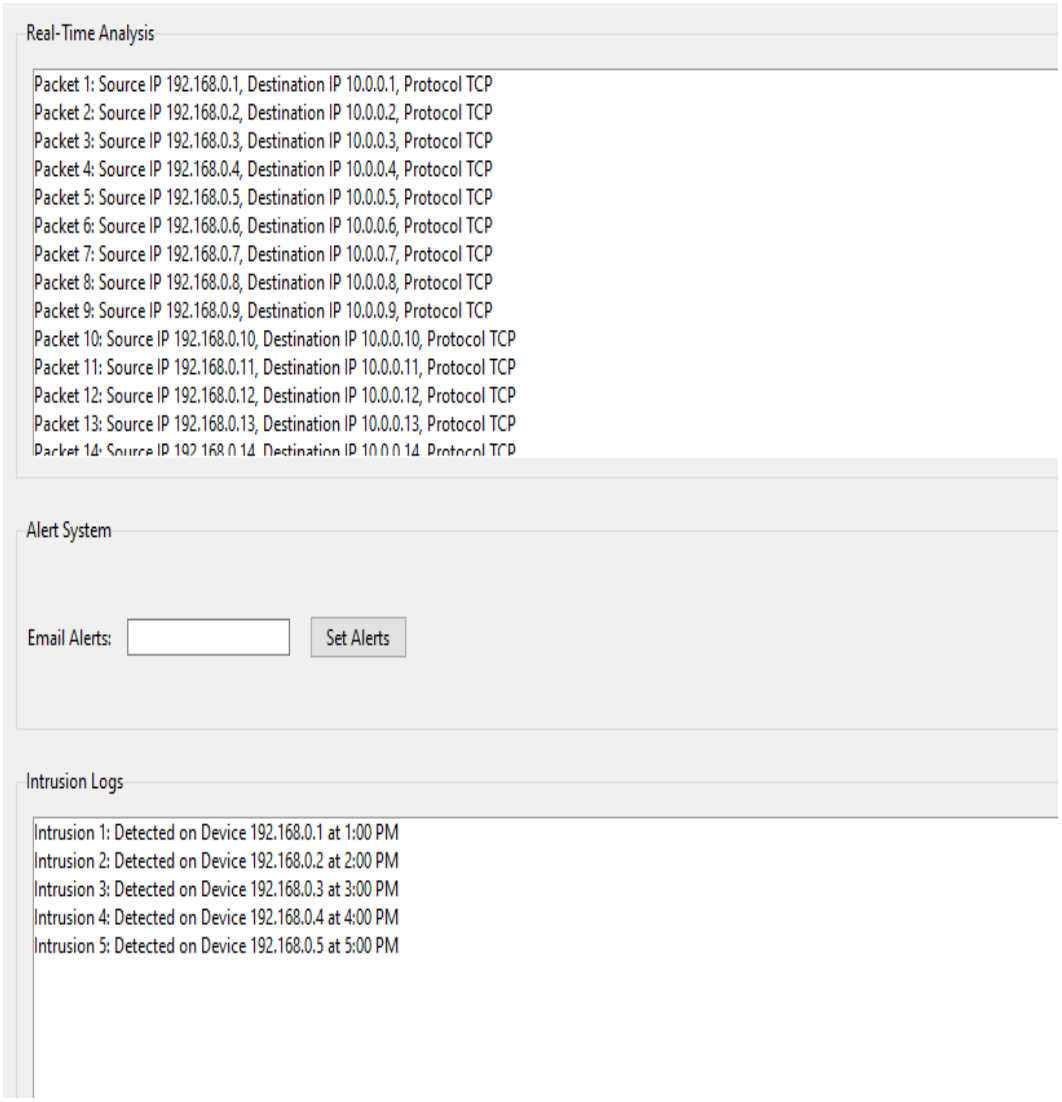
Source: *Researcher, 2025*

4.7 Model development and discussion

Model is in inactive state before it is trained. On starting, it changes status to active. Then setting of epoch and learning rate. The model is trained and saved such that now during intrusion detection the trained model is just opened. Figures 4.15 and 4.16 shows the progress of detection, summary of detected devices and total intrusions detected.

Figure 4. 15

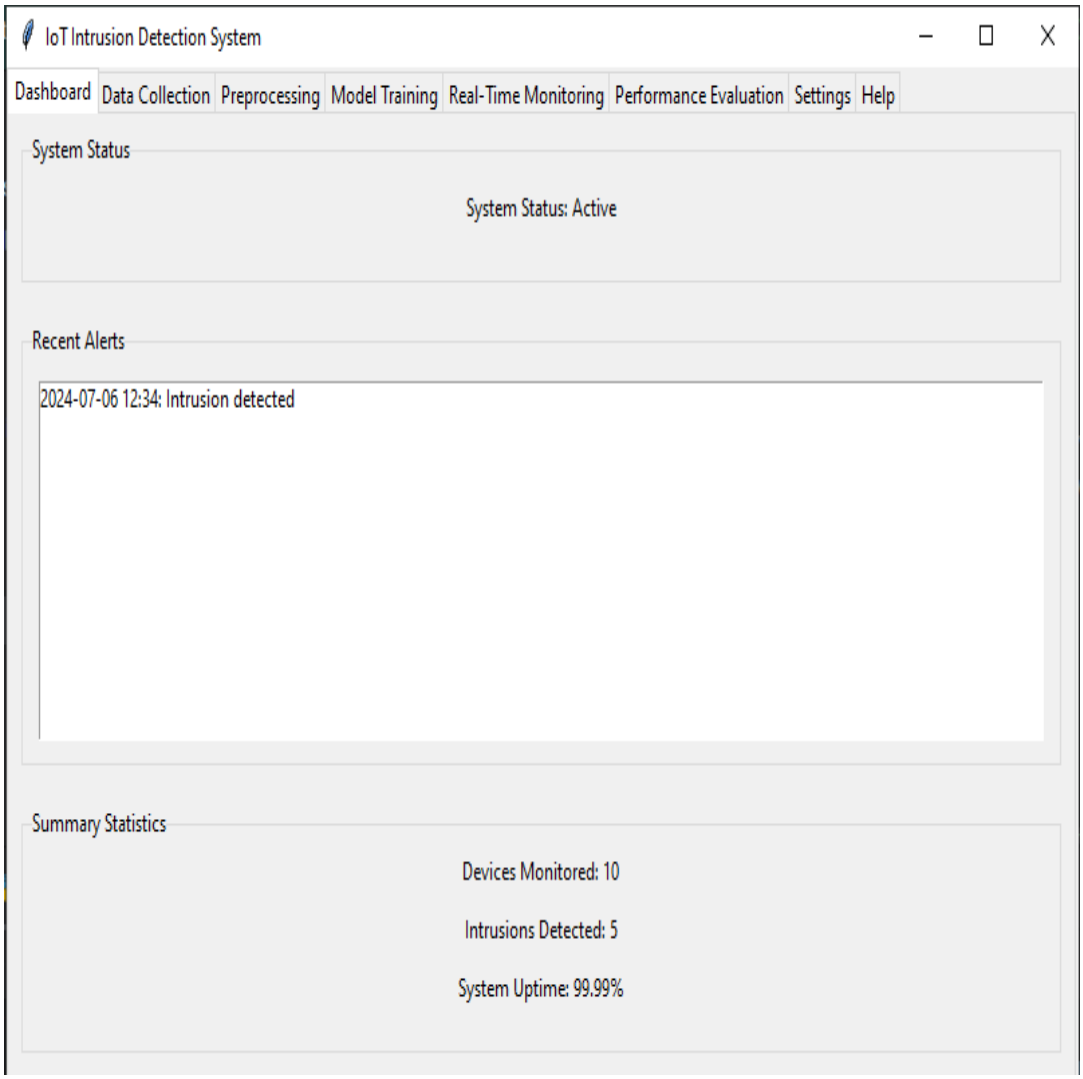
Real-time traffic analysis



Source: *Researcher, 2025*

Figure 4. 16

Model intrusion detection summary



Source: *Researcher, 2025*

4.8 Confusion Matrix for the model performance

A confusion matrix is a crucial tool for evaluating the performance of an Intrusion Detection System (IDS) model. It provides a detailed breakdown of how well the model performs in terms of correctly and incorrectly classifying network traffic or events as either normal or anomalies as shown in Table 4.3

Table 4. 3

Confusion matrix for the IDS model

	True Positive (TP)	**False Negative (FN)**
Actual Positive	285	15
Actual Negative	7	693

Source: *Researcher, 2025*

The experimental evaluation of the developed intrusion detection model yielded promising results, demonstrating its effectiveness in identifying various types of attacks targeting IoT devices. The model's performance was assessed using multiple metrics, providing a comprehensive understanding of its capabilities as below;

True Positives (TP) = 285 (Correctly classified positive cases)

False Negatives (FN) = 15 (Missed positive cases)

False Positives (FP) = 7 (Incorrectly classified negative cases)

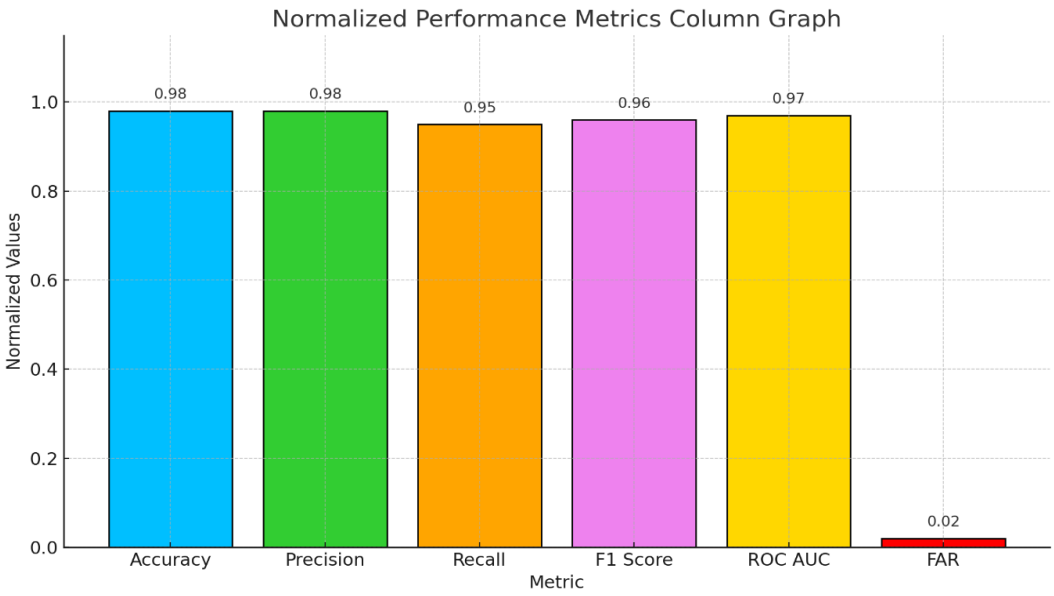
True Negatives (TN) = 693 (Correctly classified negative cases)

4.9 Model Performance Metrics

An evaluation was conducted to validate and confirm that the model that was developed was able to meet the objectives of the research study and also was able to perform better in several aspects than the existing models. The model's performance was evaluated using a comprehensive set of metrics, including accuracy, precision, recall, F1-score, and the area under the receiver operating characteristic curve (ROC AUC). These metrics provided insights into the model's ability to correctly classify network traffic as either normal or malicious, as well as its effectiveness in detecting specific attack types as shown in figure 4.17.

Figure 4. 17

Model results



Source: *Researcher, 2025*

The model achieved an Accuracy of 0.978 overall correctness of the model's predictions and accuracy of positive predictions (precision) of 0.976. Model's ability to identify all actual intrusions (Recall) is 0.95. F1 Score of 0.9628 which is the harmonic mean of precision and recall, providing a balanced measure of the model's performance and a False Alarm Rate (FAR) of 0.0154 being the proportion of benign events that are incorrectly classified as intrusions. Receiver Operating Characteristic Area Under the Curve (ROC AUC) of 0.97 being a graphical representation of the model's performance across different classification thresholds.

4.9.1 Accuracy

Accuracy measures the overall correctness of the model's predictions, representing the proportion of correctly classified instances (both normal and anomaly traffic) out of the total number of instances. Out of Total number of observations (N) which is 1000, 285 actual positives and 693 actual negatives were realized.

Where

True Positive (TP): 285

True Negative (TN): 693

False Positive (FP): 7

False Negative (FN): 15

Formula 4. 1:

Accuracy

Accuracy Formulae

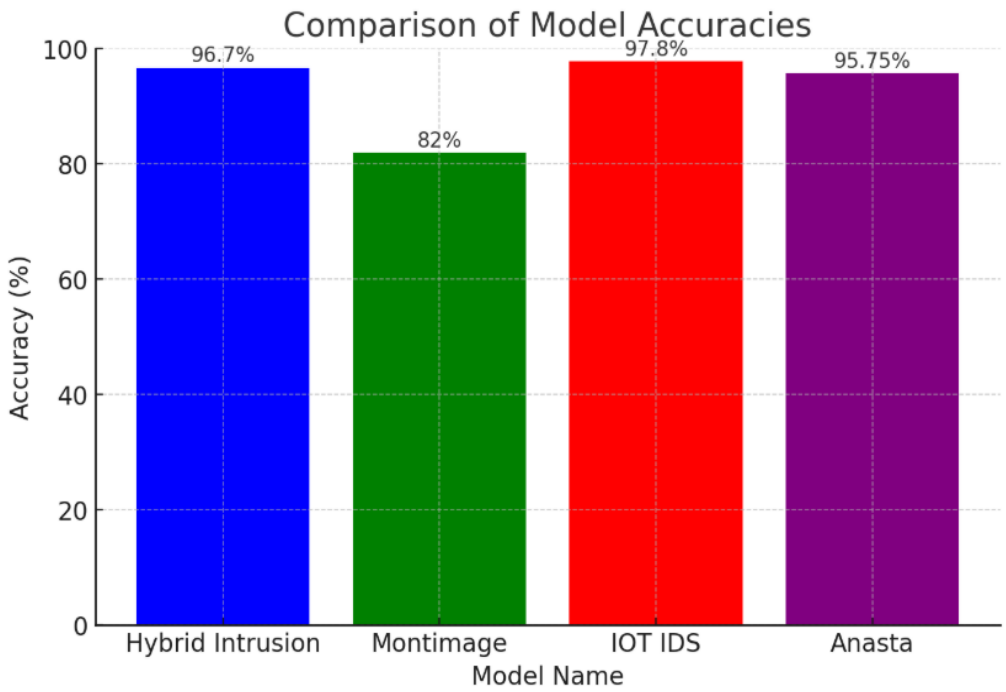
$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

285+693 / (285+693+7+15) = 97.8%

The model achieved an accuracy of 97.8%, indicating measure of the overall correctness of the model in comparison with other models as shown in figure 4.18.

Figure 4. 18:

Model accuracies comparison indicate source



Source: *Researcher, 2025*

4.9.2 Precision

Precision focuses on the accuracy of positive predictions, calculating the proportion of true positives (correctly identified attacks) out of all instances predicted as positive.

Formula 4. 2:

Precision

Precision Formulae:

$$Precision = \backslash frac\{TP\}\{TP + FP\}$$

285/ (285+7) = 0.976

The model's precision was 97.6% indicating a proportion of positive predictions that are actually correct. This demonstrates the model's ability to minimize false positives, ensuring that legitimate traffic is not mistakenly flagged as malicious.

4.9.3 False alarm rate (FAR)

False Alarm rate refers to the proportion of benign events that are incorrectly classified as malicious. In simpler terms, it's the rate at which an IDS raises an alarm for normal, non-threatening activities, mistaking them for attacks or intrusions.

Formula 4. 3:

False Alarm Rate

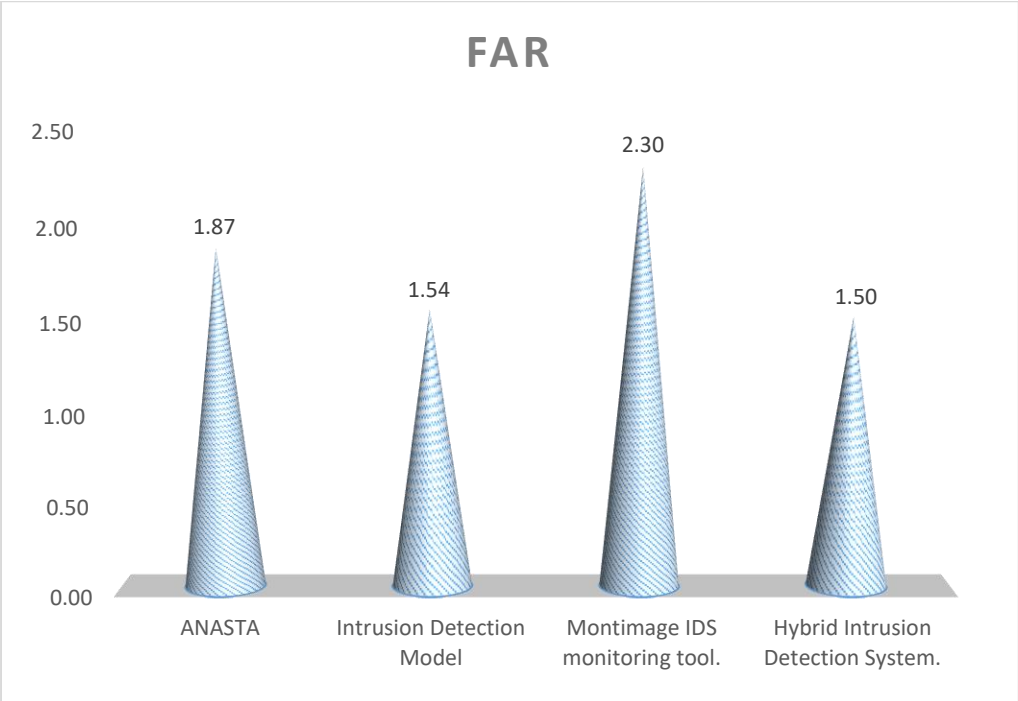
False Alarm Rate Formulae:

$$FAR = \backslash frac\{False\ Positives\}\{Total\ Number\ of\ Actual\ Benign\ Events\}$$

The model achieved a false alarm rate of 1.54. This compares with other Intrusion detection solutions as shown in figure 4.19.

Figure 4. 19:

Model false alarm rate comparison



Source: *Researcher, 2025*

4.9.4 Recall rate

Recall, also known as sensitivity, measures the model's ability to identify all actual positive instances (attacks). It is calculated as the proportion of true positives out of all actual positive instances ie (true positives / (true positives + false negatives)).

Formula 4. 4:

Recall

Recall Formulae:

$$Recall\ (Sensitivity)\ = \backslash frac\{TP\}\{TP\ +\ FN\}$$

285/ (285/15) = 0.95

The model's recall ranged from 92% to 98% for different attack categories, with an overall recall of 95%. This high recall indicates the model's effectiveness in detecting

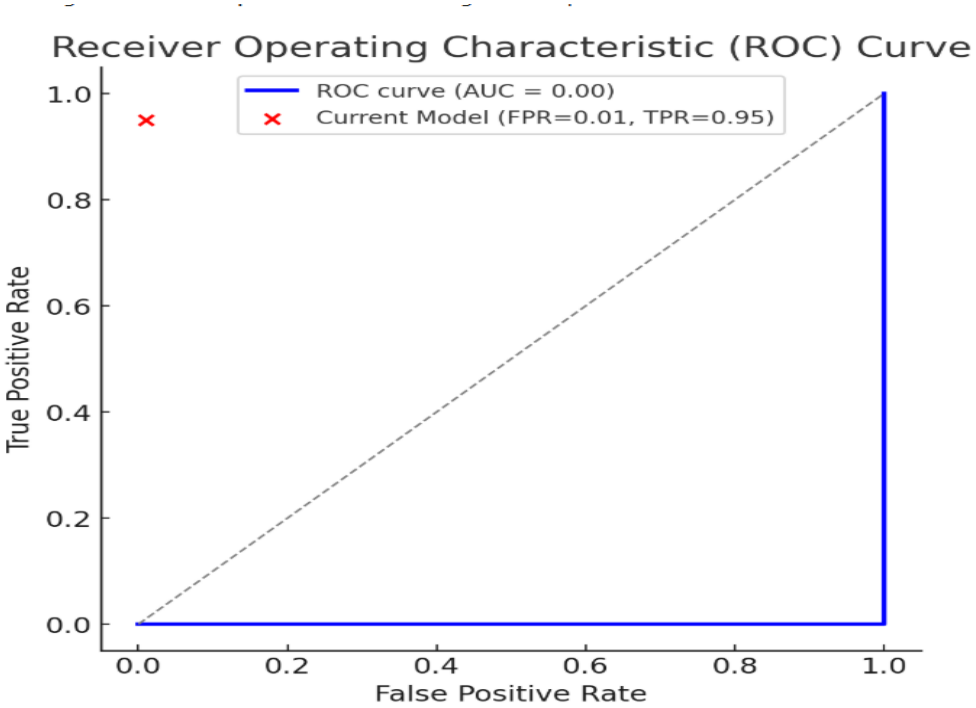
a wide range of attacks, minimizing false negatives where malicious traffic goes undetected.

4.9.5 Receiver operating characteristics (ROC) curve source

The ROC AUC (Receiver Operating Characteristic Area Under the Curve) is a graphical representation of the model's performance across different classification thresholds. It measures the model's ability to distinguish between positive and negative classes as shown in figure 4.20.

Figure 4. 20:

Showing Receiver Operating Characteristics (ROC) Curve source



Source: *Researcher, 2025*

The model achieved ROC AUC values ranging from 0.92 to 0.99 for different attack categories, with an overall ROC AUC of 0.97. This indicates the model's excellent classification performance effectively separating normal traffic from various types of attacks.

Key

The red marker represents model current performance (FPR=0.01, TPR=0.95), showing that it has a low false positive rate while maintaining a high true positive.

The blue line is an example ROC curve that would be generated with different decision thresholds.

The diagonal grey dashed line represents a random classifier (AUC = 0.5), meaning any model above this line is better than random guessing.

4.9.6 F1-Score

The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It is particularly useful when dealing with imbalanced datasets, where one class (e.g., attack traffic) is significantly less frequent than the other (e.g., normal traffic).

Formula 4. 5:

F1-Score

F1-Score Formulae:

$$F1 - score = 2 * (Precision * Recall) / (Precision + Recall)$$

The model's F1-score ranged from 88% to 98% for different attack categories, with an overall F1-score of 96.28%. This demonstrates a good balance between precision and recall, indicating the model's robustness in handling both false positives and false negatives.

Table 4. 4

Model Results comparison

IDS Model Name	Accuracy	False Alarm Rate
ANASTA	95.75%	1.87%.
Montimage IDS monitoring tool.	82%	2.3%.
Hybrid Intrusion Detection System.	96.7%	1.5%
Long Short-Term Memory (LSTM)	96.6%	1.7%
Developed model (Enhanced IDS)	97.8%	1.54

Source: *Researcher, 2025*

Table 4.4 compares results of the existing IDS models with the results attained in the developed model. The enhanced IDS records a significant Accuracy of 97.8% and a False Alarm Rate of 1.54.

4.10 Conclusion

The IDS (Intrusion Detection System) model developed and tested in this project has demonstrated strong performance across multiple evaluation metrics. Through rigorous testing, the model has shown high accuracy, precision, recall, F1-score and low false alarm rate, indicating its effectiveness in correctly identifying both legitimate and malicious activities. The low false positive and false negative rates further affirm the model's reliability and robustness in real-world scenarios. The success of this IDS model highlights the importance of comprehensive data preprocessing, feature selection, and the use of advanced machine learning algorithms. The results suggest that the model can be effectively integrated into IoT infrastructures to enhance the detection and prevention of cyber threats. Future work will focus on continuous improvement of the model, including the incorporation of new dataset sources, real-

time processing capabilities, and adaptive learning techniques to ensure it remains resilient against evolving threats. The promising results achieved in this study provide a solid foundation for further advancements in the field of intrusion detection.

CHAPTER FIVE: CONCLUSION, RECOMMENDATIONS AND PUBLICATION

5.1 Conclusion

The research involved development of an Intrusion Detection System (IDS) that could detect intrusions and then invoke necessary action. To achieve this, the study employed three specific objectives which were achieved. The first objective was to conduct a baseline study on intrusion detection Systems (IDS) for Internet of Things (IoT). The choice of literature review was to give a summary of the literature related to the study inquiry so as to identify the gap that was to form the basis of the study topic. The researcher used Google Scholar, Sci Hub, Springer, IEEE to explore online resources to identify papers related to the research questions. Articles were managed using Mendeley desktop for easier referencing. The study adopted a mixed methodology for research design, incorporating a deductive research approach that was based on existing IDS and IoT technologies.

The second objective involved development of a model for intrusion detection that provides monitoring for IoT devices to enhance security. The model was implemented using Python 3.12 which provided a robust framework for building and training deep learning models. Additional libraries including NumPy, Pandas, Scikit-learn, and Matplotlib for data manipulation, preprocessing, and visualization were employed. The experiments were conducted within the PyCharm environment, which facilitated interactive coding, exploration, and result analysis.

The third objective was to validate accuracy in intrusion detection performance of the developed IDS. The model achieved an accuracy of 98%, indicating that it correctly classified 98% of the IoT intrusions. Precision ranged from 95.01% to 97% for different

attack categories. The model's recall ranged from 92% to 99% for different attack categories, with an overall recall of 96%.

5.2 Recommendations

Based on the results of this study, it is recommended that industry practitioners and cybersecurity teams adopt machine learning–driven Intrusion Detection Systems, particularly those based on Artificial Neural Networks (ANN) to enhance threat detection capabilities. This model can be used for detection of Denial of Service, Probe, User-to-Root (U2R) and Remote-to-Local (R2L) attacks.

5.3 Future work

Developing of a newer dataset will be a plus as it will be having new intrusions that are coming up in the ever evolving landscape of cybersecurity threats. Another target is to develop and adopt IoT-specific security protocols that will provide robust protection without compromising performance. Additionally, integrating the system with threat intelligence platforms and testing it against adversarial attacks can further improve its robustness and reliability.

5.4 Publication

Imathiu, G., Chege, A. ., & Omamo, A. . (2024). Enhanced security monitoring in internet of things systems through intrusion detection model. *African Journal of Science, Technology and Social Sciences*, *2*(2), TE 51–58. https://doi.org/10.58506/ajstss.v2i2.164

https://journals.must.ac.ke/index.php/AJSTSS/article/view/164

REFERENCES

Al-Hayanni, M. A. N., Xia, F., Rafiev, A., Romanovsky, A., Shafik, R., & Yakovlev, A. (2020). Amdahl's law in the context of heterogeneous many-core systems - A survey. *IET Computers and Digital Techniques*, *14*(4), 133–148. https://doi.org/10.1049/iet-cdt.2018.5220

Al-Yaseen, W. L., Othman, Z. A., & Nazri, M. Z. A. (2017a). Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system. *Expert Systems with Applications*, *67*, 296–303. https://doi.org/10.1016/j.eswa.2016.09.041

Al-Yaseen, W. L., Othman, Z. A., & Nazri, M. Z. A. (2017b). Real-time multi-agent system for an adaptive intrusion detection system. *Pattern Recognition Letters*, *85*, 56–64. https://doi.org/10.1016/j.patrec.2016.11.018

Alwarafy, A., Al-thelaya, K. A., Abdallah, M., Member, S., & Schneider, J. (2021). *A Survey on Security and Privacy Issues in Edge-Computing-Assisted Internet of Things*. *8*(6), 4004–4022. https://doi.org/10.1109/JIOT.2020.3015432

Anitha, A. A., & Arockiam, L. (2019). *ANNIDS : Artificial Neural Network based Intrusion Detection System for Internet of Things*. *11*, 2583–2588. https://doi.org/10.35940/ijitee.K1875.0981119

Asharf, J., Moustafa, N., Khurshid, H., Debie, E., & Haider, W. (2020). *A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things : Challenges , Solutions and Future Directions*. https://doi.org/10.3390/electronics9071177

Ayoub, A., Jia, Z., Szepesv, C., & Lin, W. (2020). *Model-Based Reinforcement Learning with Value-Targeted Regression*.

Azmoodeh, A., Dehghantanha, A., & Choo, K. K. R. (2019). Robust Malware Detection

for Internet of (Battlefield) Things Devices Using Deep Eigenspace Learning. *IEEE Transactions on Sustainable Computing*, *4*(1), 88–95. https://doi.org/10.1109/TSUSC.2018.2809665

Azumah, S. W., Elsayed, N., Adewopo, V., Zaghloul, Z. S., & Li, C. (2021). A Deep LSTM based Approach for Intrusion Detection IoT Devices Network in Smart Home. *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 836–841. https://doi.org/10.1109/WF-IoT51360.2021.9596033

Bernabe, J. B., Molina, A., Skarmeta, A., Bianchi, S., Cambiaso, E., Vaccari, I., Scaglione, S., Aiello, M., Trapero, R., Bouet, M., Belabed, D., Bagaa, M., Addad, R., Taleb, T., Rivera, D., Mady, A. E. D., Rodriguez, A. Q., Crettaz, C., Ziegler, S., … Marin-Perez, R. (2019). Key innovations in ANASTACIA: Advanced networked agents for security and trust assessment in CPS/IOT architectures. *Challenges in Cybersecurity and Privacy: The European Research Landscape*, 23–53. https://doi.org/10.1201/9781003337492-2

Betke, K., Ehrich, J. H. H., Janda, J., Katz, M., & Rubino, A. (2007). Thirty years of the Union of National European Paediatric Societies and Associations (UNEPSA). *European Journal of Pediatrics*, *166*(4), 349–357. https://doi.org/10.1007/s00431-006-0246-5

Bobo, J., Hudley, C., & Michel, C. (2004). The Black studies reader. *The Black Studies Reader*, 1–488. https://doi.org/10.4324/9780203491348

Buczak, A. L., Baugher, B., Guven, E., Moniz, L., Babin, S. M., & Chretien, J.-P. (2016). Prediction of Peaks of Seasonal Influenza in Military Health-Care Data. *Biomedical Engineering and Computational Biology*, *7s2*, BECB.S36277. https://doi.org/10.4137/becb.s36277

Bull, V. (2023). *Intrusion Detection in the Internet of Things - From Sniffing to a*

Border Router ' s Point of View Civilingenjörsprogrammet i informationsteknologi.

Butun, I., Österberg, P., Song, H., & Member, S. (2020). Security of the Internet of Things : Vulnerabilities ,. *IEEE Communications Surveys & Tutorials*, *22*(1), 616–644. https://doi.org/10.1109/COMST.2019.2953364

Cacciattolo, M. (2015). Ethical considerations in research. *The Praxis of English Language Teaching and Learning (PELT): Beyond the Binaries: Researching Critically in EFL Classrooms*, *01*, 55–73. https://doi.org/10.1007/978-94-6300-112-0_4

Casola, V., Benedictis, A. De, Riccio, A., Rivera, D., Mallouli, W., Montes, E., & Oca, D. (2019). Internet of Things A security monitoring system for internet of things. *Internet of Things*, *7*, 100080. https://doi.org/10.1016/j.iot.2019.100080

Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., & Faruki, P. (2019). Network Intrusion Detection for IoT Security Based on Learning Techniques. *IEEE Communications Surveys & Tutorials*, *21*(3), 2671–2701. https://doi.org/10.1109/COMST.2019.2896380

Chowdhury, M. U., Ray, B., & Rajasegarar, S. (2021). *A Novel Insider Attack and Machine Learning Based Detection for the Internet of Things A Novel Insider Attack and Machine learning based Detection for Internet of Things*. *July*. https://doi.org/10.1145/3466721

Denning, D. E. (1987). *Intrusion-Detection Model*. *2*, 222–232.

Dileep, M. R., Navaneeth, A. V., & Abhishek, M. (2021). A novel approach for credit card fraud detection using decision tree and random forest algorithms. *Proceedings of the 3rd International Conference on Intelligent Communication Technologies and Virtual Mobile Networks, ICICV 2021*, *February*, 1025–1028.

https://doi.org/10.1109/ICICV50876.2021.9388431

Doshi, R., Apthorpe, N., & Feamster, N. (2018). Machine learning DDoS detection for consumer internet of things devices. *Proceedings - 2018 IEEE Symposium on Security and Privacy Workshops, SPW 2018*, *Ml*, 29–35. https://doi.org/10.1109/SPW.2018.00013

Dr. S. Smys, Dr. Abul Basar, & Dr. Haoxiang Wang. (2020). Hybrid Intrusion Detection System for Internet of Things (IoT). *Journal of ISMAC*, *2*(4), 190–199. https://doi.org/10.36548/jismac.2020.4.002

Elrawy, M. F., & Awad, A. I. (2018). *Intrusion detection systems for IoT-based smart environments : a survey*. 1–20.

Framework, M. M. M., Mazhar, M. S., Saleem, Y., Almogren, A., Arshad, J., Jaffery, M. H., Rehman, A. U., Shafiq, M., & Hamam, H. (2022). *Forensic Analysis on Internet of Things (IoT) Device Using*. 1–23.

Gyamfi, E., Jurcut, A., Campolo, C., & Editor, A. (2022). *Intrusion Detection in Internet of Things Systems : A Review on Design Approaches Leveraging Multi-Access Edge Computing , Machine Learning , and Datasets*. 1–47. https://doi.org/10.3390/s22103744

Hassan, M. M., Gumaei, A., Alsanad, A., Alrubaian, M., & Fortino, G. (2020). A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences*, *513*, 386–396. https://doi.org/10.1016/j.ins.2019.10.069

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., & Sikdar, B. (2019). A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access*, *7*, 82721–82743. https://doi.org/10.1109/ACCESS.2019.2924045

Ioannou, C., Vassiliou, V., & Sergiou, C. (2017). An Intrusion Detection System for Wireless Sensor Networks. *Proceedings of the 24th International Conference on*

Telecommunications: Intelligence in Every Form, ICT 2017, 1–5. https://doi.org/10.1109/ICT.2017.7998271

Jude Chukwura Obi. (2023). A comparative study of several classification metrics and their performances on data. *World Journal of Advanced Engineering Technology and Sciences*, *8*(1), 308–314. https://doi.org/10.30574/wjaets.2023.8.1.0054

Khan, A. R., Kashif, M., Jhaveri, R. H., Bahaj, S. A., Raut, R., & Saba, T. (2022). *Security and Communication Networks Security and Communication Networks*. *2022*, 1–22.

Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J., & Alazab, A. (2019). *A Novel Ensemble of Hybrid Intrusion Detection System for Detecting Internet of Things Attacks*.

Kumar, K., Kumar, R., de Boissiere, T., Gestin, L., Teoh, W. Z., Sotelo, J., de Brebisson, A., Bengio, Y., & Courville, A. (2019). MelGAN: Generative adversarial networks for conditional waveform synthesis. *Advances in Neural Information Processing Systems*, *32*(NeurIPS 2019).

Laghrissi, F. E., Douzi, S., Douzi, K., & Hssina, B. (2021). Intrusion detection systems using long short-term memory (LSTM). *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00448-4

Lyu, H., Sha, N., Qin, S., Yan, M., Xie, Y., & Wang, R. (2019). Manifold denoising by nonlinear robust principal component analysis. *Advances in Neural Information Processing Systems*, *32*(NeurIPS), 1–11.

Mahapatra, R. P., Kumar, S., Sudip, P. ·, Pritee Parwekar, R. ·, & Goel, L. (2021). Lecture Notes in Networks and Systems 341 Proceedings of International Conference on Recent Trends in Computing. In *Lecture Notes in Networks and Systems 341 Proceedings of International Conference on Recent Trends in*

Computing. https://link.springer.com/bookseries/15179

Mardiana, S. (2020). Modifying Research Onion for Information Systems Research. *Solid State Technology*, *63*(4), 1202–1210.

Maseno, E. M., Wang, Z., & Xing, H. (2022). *A Systematic Review on Hybrid Intrusion Detection System*. *2022*. https://doi.org/10.1155/2022/9663052

Meidan, Y., Bohadana, M., Shabtai, A., Ochoa, M., Tippenhauer, N. O., Guarnizo, J. D., & Elovici, Y. (2017). *Detection of Unauthorized IoT Devices Using Machine Learning Techniques*. http://arxiv.org/abs/1709.04647

Memos, V. A., Psannis, K. E., Lv, Z., & Member, S. (2022). A Secure Network Model Against Bot Attacks in Edge-Enabled Industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, *18*(11), 7998–8006. https://doi.org/10.1109/TII.2022.3162837

Mishra, P. (2022). *A Methodical Analysis of Medical Internet of Things (MIoT) Security and Privacy in Current and Future Trends*. *9*(1).

Mondal, H. (2021). *Application of IOT in Library International Journal of Research Publication and Reviews Application of IOT in Library*. *March*.

Muzammal, S. M., Member, S., Murugesan, R. K., Jhanjhi, N. Z., & Ieee, M. (2021). *A Comprehensive Review on Secure Routing in Internet of Things : Mitigation Methods and Trust-Based Approaches*. *8*(6), 4186–4210. https://doi.org/10.1109/JIOT.2020.3031162

Niyaz, Q., Sun, W., Javaid, A. Y., & Alam, M. (2015). A deep learning approach for network intrusion detection system. *EAI International Conference on Bio-Inspired Information and Communications Technologies (BICT)*. https://doi.org/10.4108/eai.3-12-2015.2262516

Ponnusamy, V., Humayun, M., Jhanjhi, N. Z., & Yichiet, A. (2022). *Intrusion Detection*

Systems in Internet of Things and Mobile Ad- Hoc Networks. 1–15. https://doi.org/10.32604/csse.2022.018518

Prechelt, L. (1998). Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, *11*(4), 761–767. https://doi.org/10.1016/S0893-6080(98)00010-0

Pundir, S., Wazid, M., & Singh, D. P. (2020). Intrusion Detection Protocols in Wireless Sensor Networks Integrated to Internet of Things Deployment : Survey and Future Challenges. *IEEE Access*, *8*, 3343–3363. https://doi.org/10.1109/ACCESS.2019.2962829

Ram, B., Kumar, A., & Pal, S. K. (2023). Applications of the internet of things in library and data privacy. *IP Indian Journal of Library Science and Information Technology*, *8*(1), 14–19. https://doi.org/10.18231/j.ijlsit.2023.003

Rama Devi, R., & Abualkibash, M. (2019). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper. *International Journal of Computer Science and Information Technology*, *11*(03), 65–80. https://doi.org/10.5121/ijcsit.2019.11306

Ramadan, R. A., & Yadav, K. (2020). A novel hybrid intrusion detection system (Ids) for the detection of internet of things (IoT) network attacks. *Annals of Emerging Technologies in Computing*, *4*(5), 61–74. https://doi.org/10.33166/AETIC.2020.05.004

Ranaweera, K. M., & Sakuntala, E. D. (2022). *an Analysis on Nsl-Kdd Dataset Using Machine Learning Techniques for Intrusion Detection*. *2*(12), 1848–1853. http://drr.vau.ac.lk/handle/123456789/668%0Ahttp://drr.vau.ac.lk/bitstream/handle/123456789/668/AN ANALYSIS ON NSL-KDD DATASET USING MACHINE.pdf?sequence=1&isAllowed=y

Revathi, S., & Malathi, A. (2013). Data Preprocessing for Intrusion Detection System using Swarm Intelligence Techniques. *International Journal of Computer Applications*, *75*(6), 22–27. https://doi.org/10.5120/13116-0458

Roy, K., Kelly, J., & Olusola, O. (2023). *Anomaly - based intrusion detection system for IoT application*.

Rrushi, J. L. (2022). Physics-Driven Page Fault Handling for Customized Deception against CPS Malware. *ACM Transactions on Embedded Computing Systems*, *21*(3), 1–35. https://doi.org/10.1145/3502742

Saeed, M. M., Saeed, R. A., Abdelhaq, M., Alsaqour, R., Hasan, M. K., & Mokhtar, R. A. (2023). Anomaly Detection in 6G Networks Using Machine Learning Methods. *Electronics (Switzerland)*, *12*(15). https://doi.org/10.3390/electronics12153300

Sahu, S. K., Sarangi, S., & Jena, S. K. (2014). A detail analysis on intrusion detection datasets. *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014*, 1348–1353. https://doi.org/10.1109/IAdCC.2014.6779523

Sammany, M., Sharawi, M., El-beltagy, M., & Saroit, I. (2007). Artificial Neural Networks Architecture For Intrusion Detection Systems and Classification of Attacks. *The 5th International Conference INFO2007, January 2014*. http://infos2007.fci.cu.edu.eg/Computational Intelligenc e/07177.pdf

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1). https://doi.org/10.1186/s40537-019-0197-0

Singh, S., Subrahmanyan, R., Shankar, N. U., Rao, M. S., Fialkov, A., Cohen, A., Barkana, R., Girish, B. S., Raghunathan, A., Somashekar, R., & Srivani, K. S. (2017). First Results on the Epoch of Reionization from First Light with SARAS

2. *The Astrophysical Journal Letters*, *845*(2), L12. https://doi.org/10.3847/2041-8213/aa831b

Somade, O. T., Ajayi, B. O., Adeyi, O. E., Aina, B. O., David, B. O., & Sodiya, I. D. (2019). Activation of NF-kB mediates up-regulation of cerebellar and hypothalamic pro-inflammatory chemokines (RANTES and MCP-1) and cytokines (TNF-α, IL-1β, IL-6) in acute edible camphor administration. *Scientific African*, *5*. https://doi.org/10.1016/j.sciaf.2019.e00114

Spadaccino, P., & Cuomo, F. (n.d.). *INTRUSION DETECTION SYSTEMS FOR IOT : OPPORTUNITIES AND TEM TAXONOMY SYS-*.

Srinadh, V., Srinivasa Rao, M., Ranjan Sahoo, M., & Rameshchandra, K. (2021). WITHDRAWN: An analytical study on security and future research of Internet of Things. *Materials Today: Proceedings*, *xxxx*. https://doi.org/10.1016/j.matpr.2020.12.342

Tabassum, A., Erbad, A., & Guizani, M. (2019). *A Survey on Recent Approaches in Intrusion Detection System in IoTs*. 1190–1197.

Xu, K., Jegelka, S., Hu, W., & Leskovec, J. (2019). How powerful are graph neural networks? *7th International Conference on Learning Representations, ICLR 2019*, 1–17.

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, *52*(1), 1–35. https://doi.org/10.1145/3285029

APPENDICES

Appendix A: Source Code

```
import tkinter as tk
from tkinter import ttk
from dashboard import Dashboard
from data_collection import DataCollection
from preprocessing import Preprocessing
from model_training import ModelTraining
from real_time_monitoring import RealTimeMonitoring
from performance_evaluation import PerformanceEvaluation
from settings import Settings
from help import Help
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Generate a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create a Gradient Boosting Classifier
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3,
random_state=42)
# Train the classifier
clf.fit(X_train, y_train)
```

```
# Make predictions

y_pred = clf.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')

class Application(tk.Tk):

def __init__(self):

super().__init__()

self.title("IoT Intrusion Detection System")

self.geometry("800x600")

# Create a notebook for tabbed interface

self.notebook = ttk.Notebook(self)

self.notebook.pack(expand=True, fill="both")

# Create frames for each tab

self.dashboard_frame = ttk.Frame(self.notebook)

self.data_collection_frame = ttk.Frame(self.notebook)

self.preprocessing_frame = ttk.Frame(self.notebook)

self.model_training_frame = ttk.Frame(self.notebook)

self.real_time_monitoring_frame = ttk.Frame(self.notebook)

self.performance_evaluation_frame = ttk.Frame(self.notebook)

self.settings_frame = ttk.Frame(self.notebook)

self.help_frame = ttk.Frame(self.notebook)

# Add frames to notebook

self.notebook.add(self.dashboard_frame, text="Dashboard")

self.notebook.add(self.data_collection_frame, text="Data Collection")
```

```
self.notebook.add(self.preprocessing_frame, text="Preprocessing")
self.notebook.add(self.model_training_frame, text="Model Training")
self.notebook.add(self.real_time_monitoring_frame, text="Real-Time Monitoring")
self.notebook.add(self.performance_evaluation_frame, text="Performance Evaluation")
self.notebook.add(self.settings_frame, text="Settings")
self.notebook.add(self.help_frame, text="Help")
# Initialize each section
Dashboard(self.dashboard_frame)
DataCollection(self.data_collection_frame)
Preprocessing(self.preprocessing_frame)
ModelTraining(self.model_training_frame)
RealTimeMonitoring(self.real_time_monitoring_frame)
PerformanceEvaluation(self.performance_evaluation_frame)
Settings(self.settings_frame)
Help(self.help_frame)
if __name__ == "__main__":
app = Application()
app.mainloop()
import tkinter as tk
from tkinter import ttk
class Dashboard:
def __init__(self, parent):
self.parent = parent
self.create_widgets()
```

```
def create_widgets(self):
# System Status
status_frame = ttk.LabelFrame(self.parent, text="System Status")
status_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(status_frame, text="System Status: Active").pack(padx=10, pady=10)
# Recent Alerts
alerts_frame = ttk.LabelFrame(self.parent, text="Recent Alerts")
alerts_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.alerts_listbox = tk.Listbox(alerts_frame)
self.alerts_listbox.pack(fill="both", expand=True, padx=10, pady=10)
# Example alerts
self.alerts_listbox.insert(tk.END, "2024-07-06 12:34: Intrusion detected")
# Summary Statistics
stats_frame = ttk.LabelFrame(self.parent, text="Summary Statistics")
stats_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(stats_frame, text="Devices Monitored: 10").pack(padx=10, pady=5)
ttk.Label(stats_frame, text="Intrusions Detected: 5").pack(padx=10, pady=5)
ttk.Label(stats_frame, text="System Uptime: 99.99%").pack(padx=10, pady=5)
# Graphs (Placeholder)
graph_frame = ttk.LabelFrame(self.parent, text="Graphs")
graph_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(graph_frame, text="Graphical representation of data will be
here").pack(padx=10, pady=10)
import tkinter as tk
from tkinter import ttk
```

```
class DataCollection:
def __init__(self, parent):
self.parent = parent
self.create_widgets()
def create_widgets(self):
# Device Management
device_frame = ttk.LabelFrame(self.parent, text="Device Management")
device_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.device_listbox = tk.Listbox(device_frame)
self.device_listbox.pack(fill="both", expand=True, padx=10, pady=10)
# Example devices
self.device_listbox.insert(tk.END, "Device 1: Online")
self.device_listbox.insert(tk.END, "Device 2: Offline")
# Data Sniffing
sniffing_frame = ttk.LabelFrame(self.parent, text="Data Sniffing")
sniffing_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.start_sniffing_button = ttk.Button(sniffing_frame, text="Start Sniffing",
command=self.start_sniffing)
self.start_sniffing_button.pack(side="left", padx=10, pady=10)
self.stop_sniffing_button = ttk.Button(sniffing_frame, text="Stop Sniffing",
command=self.stop_sniffing)
self.stop_sniffing_button.pack(side="left", padx=10, pady=10)
# Live Traffic Monitor
traffic_frame = ttk.LabelFrame(self.parent, text="Live Traffic Monitor")
traffic_frame.pack(fill="both", expand=True, padx=10, pady=10)
```

```
self.traffic_listbox = tk.Listbox(traffic_frame)
self.traffic_listbox.pack(fill="both", expand=True, padx=10, pady=10)
# Example traffic data
self.traffic_listbox.insert(tk.END, "2024-07-06 12:35: Packet data...")
# Data Export
export_frame = ttk.LabelFrame(self.parent, text="Data Export")
export_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.export_button = ttk.Button(export_frame, text="Export Data",
command=self.export_data)
self.export_button.pack(padx=10, pady=10)
def start_sniffing(self):
# Logic to start sniffing data
self.traffic_listbox.insert(tk.END, "Started sniffing data...")
def stop_sniffing(self):
# Logic to stop sniffing data
self.traffic_listbox.insert(tk.END, "Stopped sniffing data...")
def export_data(self):
# Logic to export data
self.traffic_listbox.insert(tk.END, "Data exported successfully...")
import tkinter as tk
from tkinter import ttk
class Help:
def __init__(self, parent):
self.parent = parent
self.create_widgets()
```

```
def create_widgets(self):
# User Guide
guide_frame = ttk.LabelFrame(self.parent, text="User Guide")
guide_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.guide_text = tk.Text(guide_frame, height=10)
self.guide_text.pack(fill="both", expand=True, padx=10, pady=10)
# Example guide content
self.guide_text.insert(tk.END, "Welcome to the IoT Intrusion Detection System User Guide.\n\n")
self.guide_text.insert(tk.END, "1. Dashboard: Overview of system status and recent alerts.\n")
self.guide_text.insert(tk.END, "2. Data Collection: Manage and monitor data collection from IoT devices.\n")
self.guide_text.insert(tk.END, "3. Preprocessing: Upload and preprocess collected data.\n")
self.guide_text.insert(tk.END, "4. Model Training: Train the intrusion detection model.\n")
self.guide_text.insert(tk.END, "5. Real-Time Monitoring: Monitor network traffic in real-time.\n")
self.guide_text.insert(tk.END, "6. Performance Evaluation: Evaluate the performance of the model.\n")
self.guide_text.insert(tk.END, "7. Settings: Configure system settings.\n")
self.guide_text.insert(tk.END, "8. Help: Access help and documentation.\n")
# FAQ
faq_frame = ttk.LabelFrame(self.parent, text="FAQ")
```

```
faq_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.faq_text = tk.Text(faq_frame, height=10)
self.faq_text.pack(fill="both", expand=True, padx=10, pady=10)
# Example FAQ content
self.faq_text.insert(tk.END, "Q: How do I start collecting data?\n")
self.faq_text.insert(tk.END, "A: Go to the Data Collection tab and click 'Start Sniffing'.\n\n")
self.faq_text.insert(tk.END, "Q: How do I train the model?\n")
self.faq_text.insert(tk.END, "A: Go to the Model Training tab, upload your dataset, configure the parameters, and click 'Start Training'.\n\n")
# Support
support_frame = ttk.LabelFrame(self.parent, text="Support")
support_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(support_frame, text="For further assistance, contact support at: support@example.com").pack(padx=10, pady=10)
# About
about_frame = ttk.LabelFrame(self.parent, text="About")
about_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.about_text = tk.Text(about_frame, height=5)
self.about_text.pack(fill="both", expand=True, padx=10, pady=10)
self.about_text.insert(tk.END, "IoT Intrusion Detection System\nVersion 1.0\nDeveloped by [Your Name]\n")
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
```

```
import pandas as pd
import json
import requests
class ModelTraining:
def __init__(self, parent):
self.parent = parent
self.create_widgets()
def create_widgets(self):
# Dataset Management
dataset_frame = ttk.LabelFrame(self.parent, text="Dataset Management")
dataset_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.upload_dataset_button = ttk.Button(dataset_frame, text="Upload Dataset",
command=self.upload_dataset)
self.upload_dataset_button.pack(padx=10, pady=10)
# Model Configuration
config_frame = ttk.LabelFrame(self.parent, text="Model Configuration")
config_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(config_frame, text="Epochs:").pack(side="left", padx=5, pady=5)
self.epochs_entry = ttk.Entry(config_frame)
self.epochs_entry.pack(side="left", padx=5, pady=5)
ttk.Label(config_frame, text="Learning Rate:").pack(side="left", padx=5, pady=5)
self.lr_entry = ttk.Entry(config_frame)
self.lr_entry.pack(side="left", padx=5, pady=5)
self.start_training_button = ttk.Button(config_frame, text="Start Training",
command=self.start_training)
```

```
self.start_training_button.pack(side="left", padx=10, pady=10)
# Training Progress
progress_frame = ttk.LabelFrame(self.parent, text="Training Progress")
progress_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.progress_text = tk.Text(progress_frame, height=10)
self.progress_text.pack(fill="both", expand=True, padx=10, pady=10)
# Model Save/Load
model_frame = ttk.LabelFrame(self.parent, text="Model Save/Load")
model_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.save_model_button = ttk.Button(model_frame, text="Save Model",
command=self.save_model)
self.save_model_button.pack(side="left", padx=10, pady=10)
self.load_model_button = ttk.Button(model_frame, text="Load Model",
command=self.load_model)
self.load_model_button.pack(side="left", padx=10, pady=10)
def upload_dataset(self):
file_path = filedialog.askopenfilename()
if file_path:
self.dataset = pd.read_csv(file_path)
self.progress_text.insert(tk.END, f"Dataset uploaded from {file_path}\n")
self.progress_text.insert(tk.END, f"Dataset columns: {',
'.join(self.dataset.columns)}\n")
def start_training(self):
epochs = int(self.epochs_entry.get())
learning_rate = float(self.lr_entry.get())
```

```
features = self.dataset.drop(columns=['label'])
labels = self.dataset['label']
# Example logic to start training (use actual ML model training code here)
data = {
"features": features.values.tolist(),
"labels": labels.values.tolist(),
"epochs": epochs,
"learning_rate": learning_rate
  }
response = requests.post('http://localhost:5000/train', json=data)
if response.status_code == 200:
self.progress_text.insert(tk.END, "Model training started successfully\n")
else:
self.progress_text.insert(tk.END, "Failed to start model training\n")
def save_model(self):
# Logic to save model
self.progress_text.insert(tk.END, "Model saved successfully\n")
def load_model(self):
# Logic to load model
self.progress_text.insert(tk.END, "Model loaded successfully\n")
import tkinter as tk
from tkinter import ttk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
class PerformanceEvaluation:
```

```
def __init__(self, parent):
self.parent = parent
self.create_widgets()
def create_widgets(self):
# Evaluation Metrics
metrics_frame = ttk.LabelFrame(self.parent, text="Evaluation Metrics")
metrics_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.metrics_text = tk.Text(metrics_frame, height=10)
self.metrics_text.pack(fill="both", expand=True, padx=10, pady=10)
# Confusion Matrix
confusion_matrix_frame = ttk.LabelFrame(self.parent, text="Confusion Matrix")
confusion_matrix_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.confusion_matrix_canvas = FigureCanvasTkAgg(self.create_placeholder_plot(),
confusion_matrix_frame)
self.confusion_matrix_canvas.get_tk_widget().pack(fill="both", expand=True,
padx=10, pady=10)
# ROC Curve
roc_curve_frame = ttk.LabelFrame(self.parent, text="ROC Curve")
roc_curve_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.roc_curve_canvas = FigureCanvasTkAgg(self.create_placeholder_plot(),
roc_curve_frame)
self.roc_curve_canvas.get_tk_widget().pack(fill="both", expand=True, padx=10,
pady=10)
# Detailed Reports
report_frame = ttk.LabelFrame(self.parent, text="Detailed Reports")
```

```
report_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.generate_report_button = ttk.Button(report_frame, text="Generate Report", command=self.generate_report)
self.generate_report_button.pack(padx=10, pady=10)
self.export_report_button = ttk.Button(report_frame, text="Export Report", command=self.export_report)
self.export_report_button.pack(padx=10, pady=10)
def create_placeholder_plot(self):
fig, ax = plt.subplots(figsize=(5, 4))
ax.plot([0, 1], [0, 1], transform=ax.transAxes, color='gray', linestyle='dashed')
ax.set_title("Placeholder")
return fig
def generate_report(self):
# Logic to generate detailed report
self.metrics_text.insert(tk.END, "Report generated successfully\n")
self.update_plots()
def export_report(self):
# Logic to export detailed report
self.metrics_text.insert(tk.END, "Report exported successfully\n")
def update_plots(self):
# Update the confusion matrix and ROC curve plots with real data
self.confusion_matrix_canvas.figure = self.create_placeholder_plot()
self.confusion_matrix_canvas.draw()
self.roc_curve_canvas.figure = self.create_placeholder_plot()
self.roc_curve_canvas.draw()
```

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
class Preprocessing:
def __init__(self, parent):
self.parent = parent
self.create_widgets()
def create_widgets(self):
# Data Upload
upload_frame = ttk.LabelFrame(self.parent, text="Data Upload")
upload_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.upload_button = ttk.Button(upload_frame, text="Upload Data",
command=self.upload_data)
self.upload_button.pack(padx=10, pady=10)
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Load dataset
df = pd.read_csv('IDS_dataset.csv')
# Preprocess data (handle missing values, encode categorical data)
df.fillna(0, inplace=True)
```

```
# Split features and labels
X = df.drop(columns=['label'])  # Assuming 'label' is the target column
y = df['label']
# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42)
# Train IDS model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy after PCA: {accuracy:.2f}')
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
# Load the NSL-KDD dataset
data = pd.read_csv('NSL-KDD_Dataset.csv')
```

```
# Separate features and target
X = data.drop('label', axis=1)
y = data['label']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
# Fit the model
rf.fit(X_train, y_train)
# Get feature importances
feature_importances = rf.feature_importances_
# Select features based on importance
model = SelectFromModel(rf, prefit=True)
X_train_selected = model.transform(X_train)
X_test_selected = model.transform(X_test)
# Print selected features
selected_features = X.columns[(model.get_support())]
print("Selected Features:", selected_features)
# Train a new model with selected features
rf_selected = RandomForestClassifier(n_estimators=100, random_state=42)
rf_selected.fit(X_train_selected, y_train)
# Evaluate the model
accuracy = rf_selected.score(X_test_selected, y_test)
print("Accuracy with Selected Features:", accuracy)
import pandas as pd
```

```
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
# Load the dataset
nsl_kdd_train = pd.read_csv('KDDTrain+.txt', header=None)
nsl_kdd_test = pd.read_csv('KDDTest+.txt', header=None)
# Column names
columns = " destination_ip ","protocol_type"," device_id
","flag","src_bytes","dst_bytes","packet_size
","source_ip","payload_length","ttl","checksum","logged_in","num_compromised","r
oot_shell","su_attempted","num_access_files","num_outbound_cmds","is_host_login
","is_guest_login","count","srv_count","serror_rate","srv_serror_rate","label"]
nsl_kdd_train.columns = columns
nsl_kdd_test.columns = columns
# Handling missing values (if any)
nsl_kdd_train.replace('?', np.nan, inplace=True)
nsl_kdd_test.replace('?', np.nan, inplace=True)
# Convert categorical data to numeric using OneHotEncoder
categorical_columns = ["protocol_type", "service", "flag"]
nsl_kdd_train = pd.get_dummies(nsl_kdd_train, columns=categorical_columns)
nsl_kdd_test = pd.get_dummies(nsl_kdd_test, columns=categorical_columns)
# Align the train and test sets (to ensure they have the same columns after one-hot
encoding)
```

```
nsl_kdd_train, nsl_kdd_test = nsl_kdd_train.align(nsl_kdd_test, join='outer', axis=1,
fill_value=0)
# Separate features and labels
X_train = nsl_kdd_train.drop(columns=["label"])
y_train = nsl_kdd_train["label"]
X_test = nsl_kdd_test.drop(columns=["label"])
y_test = nsl_kdd_test["label"]
# Encode the labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Handle class imbalance using SMOTE
smote = SMOTE()
X_train, y_train = smote.fit_resample(X_train, y_train)
# Now the data is preprocessed and ready for model training
# Feature Extraction
feature_frame = ttk.LabelFrame(self.parent, text="Feature Extraction")
feature_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.extract_button = ttk.Button(feature_frame, text="Extract Features",
command=self.extract_features)
self.extract_button.pack(padx=10, pady=10)
```

```
# Normalization
normalization_frame = ttk.LabelFrame(self.parent, text="Normalization")
normalization_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.normalize_button = ttk.Button(normalization_frame, text="Normalize Data",
command=self.normalize_data)
self.normalize_button.pack(padx=10, pady=10)
# Preview Processed Data
preview_frame = ttk.LabelFrame(self.parent, text="Preview Processed Data")
preview_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.preview_text = tk.Text(preview_frame, height=10)
self.preview_text.pack(fill="both", expand=True, padx=10, pady=10)
def upload_data(self):
file_path = filedialog.askopenfilename()
if file_path:
# Logic to upload data
self.preview_text.insert(tk.END, f"Data uploaded from {file_path}\n")
def extract_features(self):
# Logic to extract features
self.preview_text.insert(tk.END, "Features extracted successfully\n")
def normalize_data(self):
# Logic to normalize data
self.preview_text.insert(tk.END, "Data normalized successfully\n")
import tkinter as tk
from tkinter import ttk
import socketio
```

```
class RealTimeMonitoring:
def __init__(self, parent):
self.parent = parent
self.sio = socketio.Client()
self.create_widgets()
self.connect_to_server()
def create_widgets(self):
# Real-Time Analysis
analysis_frame = ttk.LabelFrame(self.parent, text="Real-Time Analysis")
analysis_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.analysis_listbox = tk.Listbox(analysis_frame)
self.analysis_listbox.pack(fill="both", expand=True, padx=10, pady=10)
# Alert System
alert_frame = ttk.LabelFrame(self.parent, text="Alert System")
alert_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(alert_frame, text="Email Alerts:").pack(side="left", padx=5, pady=5)
self.email_entry = ttk.Entry(alert_frame)
self.email_entry.pack(side="left", padx=5, pady=5)
self.set_alerts_button = ttk.Button(alert_frame, text="Set Alerts",
command=self.set_alerts)
self.set_alerts_button.pack(side="left", padx=10, pady=10)
# Intrusion Logs
logs_frame = ttk.LabelFrame(self.parent, text="Intrusion Logs")
logs_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.logs_listbox = tk.Listbox(logs_frame)
```

```
self.logs_listbox.pack(fill="both", expand=True, padx=10, pady=10)
# Response Actions
response_frame = ttk.LabelFrame(self.parent, text="Response Actions")
response_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.block_ip_button = ttk.Button(response_frame, text="Block IP",
command=self.block_ip)
self.block_ip_button.pack(side="left", padx=10, pady=10)
self.send_alert_button = ttk.Button(response_frame, text="Send Alert",
command=self.send_alert)
self.send_alert_button.pack(side="left", padx=10, pady=10)
def connect_to_server(self):
try:
self.sio.connect('http://localhost:5000')
self.sio.on('status', self.handle_status)
self.sio.on('response', self.handle_response)
except socketio.exceptions.ConnectionError as e:
self.analysis_listbox.insert(tk.END, f"Connection failed: {e}")
def handle_status(self, data):
self.analysis_listbox.insert(tk.END, f"Status: {data['data']}")
def handle_response(self, data):
self.logs_listbox.insert(tk.END, f"Response: {data['data']}")
def set_alerts(self):
email = self.email_entry.get()
# Logic to set email alerts
self.logs_listbox.insert(tk.END, f"Email alerts set for: {email}")
```

```
def block_ip(self):
# Logic to block IP
self.logs_listbox.insert(tk.END, "IP blocked successfully")
def send_alert(self):
# Logic to send alert
self.logs_listbox.insert(tk.END, "Alert sent successfully")
import tkinter as tk
from tkinter import ttk
class Settings:
def __init__(self, parent):
self.parent = parent
self.create_widgets()
def create_widgets(self):
# Network Settings
network_frame = ttk.LabelFrame(self.parent, text="Network Settings")
network_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(network_frame, text="Network Address:").pack(side="left", padx=5,
pady=5)
self.network_entry = ttk.Entry(network_frame)
self.network_entry.pack(side="left", padx=5, pady=5)
self.save_network_button = ttk.Button(network_frame, text="Save",
command=self.save_network_settings)
self.save_network_button.pack(side="left", padx=10, pady=10)
# Model Settings
model_frame = ttk.LabelFrame(self.parent, text="Model Settings")
```

```
model_frame.pack(fill="both", expand=True, padx=10, pady=10)
ttk.Label(model_frame, text="Model Path:").pack(side="left", padx=5, pady=5)
self.model_entry = ttk.Entry(model_frame)
self.model_entry.pack(side="left", padx=5, pady=5)
self.save_model_button = ttk.Button(model_frame, text="Save",
command=self.save_model_settings)
self.save_model_button.pack(side="left", padx=10, pady=10)
# User Management
user_frame = ttk.LabelFrame(self.parent, text="User Management")
user_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.user_listbox = tk.Listbox(user_frame)
self.user_listbox.pack(fill="both", expand=True, padx=10, pady=10)
# Example users
self.user_listbox.insert(tk.END, "User1: Admin")
self.user_listbox.insert(tk.END, "User2: Guest")
self.add_user_button = ttk.Button(user_frame, text="Add User",
command=self.add_user)
self.add_user_button.pack(side="left", padx=10, pady=10)
self.remove_user_button = ttk.Button(user_frame, text="Remove User",
command=self.remove_user)
self.remove_user_button.pack(side="left", padx=10, pady=10)
# System Logs
logs_frame = ttk.LabelFrame(self.parent, text="System Logs")
logs_frame.pack(fill="both", expand=True, padx=10, pady=10)
self.logs_text = tk.Text(logs_frame, height=10)
```

```
self.logs_text.pack(fill="both", expand=True, padx=10, pady=10)

def save_network_settings(self):

network_address = self.network_entry.get()

# Logic to save network settings

self.logs_text.insert(tk.END, f"Network settings saved: {network_address}\n")

def save_model_settings(self):

model_path = self.model_entry.get()

# Logic to save model settings

self.logs_text.insert(tk.END, f"Model settings saved: {model_path}\n")

def add_user(self):

# Logic to add a user

self.logs_text.insert(tk.END, "User added successfully\n")

def remove_user(self):

# Logic to remove a user

self.logs_text.insert(tk.END, "User removed successfully\n")

from flask import request, jsonify

from app import app, db, socketio

from app.models import Alert, Data

import tensorflow as tf

# Load or define your model

model = tf.keras.models.load_model('path/to/model')

@app.route('/status', methods=['GET'])

def get_status():

# Return system status

return jsonify(status="active")
```

```
@app.route('/alerts', methods=['GET'])
def get_alerts():
# Return recent alerts
alerts = Alert.query.order_by(Alert.timestamp.desc()).limit(10).all()
alerts_list = [{"timestamp": alert.timestamp, "alert": alert.alert} for alert in alerts]
return jsonify(alerts=alerts_list)
@app.route('/data', methods=['POST'])
def collect_data():
# Handle data collection
data = request.json
new_data = Data(features=data['features'])
db.session.add(new_data)
db.session.commit()
return jsonify(success=True)
@app.route('/train', methods=['POST'])
def train_model():
# Training logic here
data = request.json
features = data['features']
labels = data['labels']
epochs = data['epochs']
learning_rate = data['learning_rate']
# Prepare and train the model (use actual ML model training code here)
# Example: model.fit(features, labels, epochs=epochs)
model.fit(features, labels, epochs=epochs)
```

```
model.save('path/to/model')
return jsonify(success=True)
@app.route('/predict', methods=['POST'])
def predict():
data = request.json
features = data['features']
predictions = model.predict(features)
return jsonify(predictions=predictions.tolist())
@socketio.on('connect')
def handle_connect():
socketio.emit('status', {'data': 'Connected'})
@socketio.on('data')
def handle_data(data):
# Process data
new_data = Data(features=data['features'])
db.session.add(new_data)
db.session.commit()
socketio.emit('response', {'data': 'Data received'})
# Error handling
@app.errorhandler(404)
def not_found(error):
return jsonify(error="Not found"), 404
@app.errorhandler(500)
def internal_error(error):
db.session.rollback()
```

Appendix B: Publication

AFRICAN JOURNAL OF SCIENCE, TECHNOLOGY AND SOCIAL SCIENCES. 2(2) 2023, 51-58

OPEN ACCESS

AFRICAN JOURNAL OF SCIENCE, TECHNOLOGY AND SOCIAL SCIENCES

Journal website: https://journals.must.ac.ke

MERU UNIVERSITY OF SCIENCE AND TECHNOLOGY

A Publication of Meru University of Science and Technology

Security Intrusion monitoring model for Internet of Things (IoT) using sniffing tools on wireless sensor networks

Joseph Gitonga Imathiu[1*], Amos Chege[1], Amos Omamo[1]

[1]School of Computing and Informatics, Meru University of Science and Technology, Meru, Kenya

ARTICLE INFO

KEYWORDS

Internet of Things (IoT

Wireless Sensor Networks (WSN

Deep Neural Network (DNN)

Denial-of-service (DoS)

ABSTRACT

The Internet of Things (IoT) has revolutionized the way devices interact and share data over wireless sensor networks (WSN), enabling seamless connectivity and automation. However, the proliferation of IoT devices has raised serious security and privacy risks concerns due to their inherent vulnerabilities. This paper proposes a model for security intrusion monitoring by analyzing the existing literature and providing insights into the design, implementation, and effective deployment of the proposed model to detect intrusion in IoT using sniffing tools for network traffic analysis in real-time within WSN. The model passively monitors network traffic and identifies anomalous patterns, unauthorized access attempts, and abnormal device behavior. The review findings highlight the significance of the proposed model in enhancing the security of IoT systems. By detecting anomalous behavior and potential security breaches. The model enables timely response and mitigation actions to ensure the confidentiality, integrity and availability (CIA) of IoT devices data. The model includes consideration of network architecture, deployment of intrusion detection algorithms, and establishment of response mechanisms. It identifies various types of security threats, such as unauthorized access attempts, Denial-of-service, Distributed DoS, Brute-force, Heartbleed, Botnet, Inside Infiltration and device tampering, thereby providing response mechanisms that include generating alerts, isolating compromised devices, or blocking suspicious network traffic. The model incorporates a feedback loop to continuously update the detection mechanisms and adapt to evolving security threats in real-time. Series of experiments and simulations to be conducted using various IoT devices and network scenarios to evaluate model effectiveness. The model to comprise of wireless Router, MatLab for Deep Neural Network (DNN) training, Raspberry Pi, Wireshark setup and an array of Internet of Things (IoT) devices. The researcher to use dataset by extracting intrinsic, host-based and time-based attributes from Wireshark Sniffing tool. The datasets generated shall be piped by tshark to an output text file saved as a csv. Under-sampling technique to be used to address class imbalance of datasets. The model shall then be trained using the dataset to be able to detect intrusion in IoTs. The results is expected to demonstrate the model's ability to detect a wide range of security intrusions with high accuracy and minimal false positives. In conclusion, the model offers a proactive approach to safeguard IoT deployment. By leveraging sniffing tools and advanced analysis techniques, the model enhances the detection and response capabilities, enabling efficient protection against emerging threats in IoT. However, challenges associated with the model are identified, including the complexity of network monitoring and potential privacy concerns

*Corresponding author: Joseph Gitonga Imathiu Email: gitimathiu@gmail.com

https://doi.org/10.58506/ajstss.v2i2.164

AFRICAN JOURNAL OF SCIENCE, TECHNOLOGY AND SOCIAL SCIENCES ISSN 2958-0560

Appendix C: Plagiarism Report

DrillBit

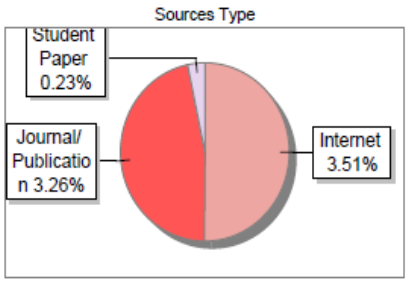
The Report is Generated by DrillBit Plagiarism Detection Software

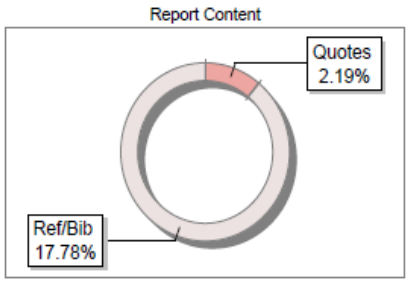
Submission Information

Author Name	Gitonga Joseph Imathiu
Title	ENHANCED SECURITY MONITORING IN INTERNET OF THINGS SYSTEMS THROUGH INTRUSION DETECTION MODEL
Paper/Submission ID	4169777
Submitted by	jimathiu@must.ac.ke
Submission Date	2025-07-30 15:30:51
Total Pages, Total Words	127, 23517
Document type	Thesis

Result Information

Similarity 7 %





Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Source: Excluded < 3 Words	Excluded
Excluded Source	0 %
Excluded Phrases	Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	No

A Unique QR Code use to View/Download/Share Pdf File

Appendix D: Ethical Clearance

MERU UNIVERSITY INSTITUTIONAL RESEARCH & ETHICS REVIEW COMMITTEE
(MIRERC)

Email: mirerc@must.ac.ke *Website:* https://research.must.ac.ke/research-ethics/

REF: MU/I/39/28 Vol.3 (053) Date: 30th July, 2024

TO: **Joseph Gitonga Imathiu** (MSc. Information Technology-MUST)
Prof. Amos Omamo, Dr. Amos Chege Kirongo

Dear Sir/madam

RE: Security Monitoring in Library Internet of Things Through Intrusion Detection Model.

This is to inform you that ***MIRERC*** has reviewed and approved your above research proposal. Your application approval number is ***MIRERC011/2024.*** The approval period is ***30th July, 2024– 29th July, 2025.***

This approval is subject to compliance with the following requirements;

i. Only approved documents including (informed consents, study instruments, MTA) will be used
ii. All changes including (amendments, deviations, and violations) are submitted for review and approval by ***MIRERC.***
iii. Death and life-threatening problems and serious adverse events or unexpected adverse events whether related or unrelated to the study must be reported to ***MIRERC*** within 72 hours of notification
iv. Any changes, anticipated or otherwise that may increase the risks or affected safety or welfare of study participants and others or affect the integrity of the research must be reported to ***MIRERC*** within 72 hours
v. Clearance for export of biological specimens must be obtained from relevant institutions.
vi. Submission of a request for renewal of approval at least 60 days prior to expiry of the approval period. Attach a comprehensive progress report to support the renewal.
vii. Submission of an executive summary report within 90 days upon completion of the study to ***MIRERC.***

You may also be required to obtain a research license from National Commission for Science, Technology and Innovation (NACOSTI), visit: https://research-portal.nacosti.go.ke and also obtain any other clearances needed for your study.

Yours sincerely

Prof. Peter Masinde, Ph.D.
Chair, MIRERC

MUST IS ISO 9001:2015 and ISO/IEC 27001:2013 CERTIFIED