

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes



Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the
Requirements for the Degree of

MASTER

In Electronic

Option: Computer Engineering

Entitled

**Edge-ML based Network Intrusion
Detection System for IoT devices**

Presented by

- **BERKANI Lina**
- **KHELIFI Cylia**

Supervisor

- **Dr. TOUZOUT Walid**

Registration Number:...../2024

*To Hind, To Youcef, To Ahmad al-Najjar, To Kamal and his Brother, To
the Soul of the Soul,
To the Children, Youths and Elderly, To the Women and Men,
To the Mountains of Gaza, Who redefined the rules, Who are writing the
history and breaking myths, Who are liberating an Ummah,
To the free souls behind bars,
To Those seeking freedom,
I give you my life.*

—LB

*To my beloved parents, whose unwavering support and encouragement have
been my guiding light, may Allah fill their lives with happiness and joy,
To my dear brothers, Fares and Marwane, for their boundless
love and belief in me,
To Nora, my lovely niece coming soon to illuminate this world, whom I cannot
wait to hold in my arms,
To my cherished family and wonderful friends, for their endless support,
Thank you for being my pillars of strength.*

—CK

Abstract

In recent years, there has been a substantial proliferation in the use of the Internet of Things in a wide variety of domains, from providing new services and options in smart home applications to industrial IoT, automating healthcare, power grids and more. However, IoT networks are prone to security breaches due to the limited computational power and constrained resources of these devices, which cannot support traditional security mechanisms. This security concern is increasingly becoming a relevant research issue, for which a number of Network Intrusion Detection Systems (NIDSs) have been proposed. In this report, we develop and implement a practical machine learning based IoT network intrusion detection system that operates on low-end microcontrollers. The proposed system is deployed on edge which ensures a fast response to attacks targeting IoT devices, thanks to the decentralized data processing. Privacy of network users is also preserved as data is kept locally at the edge of the network. Two prototypes were proposed for the realization of this project, which are based on the Raspberry Pi and the ESP32 Microcontroller. The ESP32 based prototype is composed of three sub-systems, each utilizing an ESP32 MCU. Four distinct Machine Learning algorithms were explored to detect malicious from benign traffic, and recognize the type of attack, reaching up to an accuracy of 99.76% and an F1-score of 94.25% for tree-based models. A detailed evaluation and comparison of the models was conducted. The most accurate model was selected and then optimized to obtain a lightweight and faster executable version, for an easier deployment on edge. The models were additionally tested on real-world data, by predicting the class label of previously unseen data from new Pcap files.

Keywords: Cybersecurity, NIDS, IoT, Intrusion Detection, Edge Computing, Machine Learning, System-on-chip, FreeRTOS, Communication Protocols

Acknowledgments

In the Name of Allah, the Most Gracious, the Most Merciful.

All the praises and thanks be to Allah Almighty, the Giver of blessings and gifts. Prayers and peace of Allah be upon the noble Prophet Muhammad SAW. We are grateful to Allah SWT for granting us the grace and strength to complete this thesis and attain this stage of achievement in life.

We are deeply thankful to our parents, for their unwavering love and support throughout the years. Words cannot express our gratitude for the sacrifices you have made on our behalf.

We would like to express our sincere gratitude to our supervisor **Dr. Touzout Walid** for his guidance and support throughout this project.

We extend our heartfelt appreciation to our siblings and friends, who believed in us and encouraged us through life's hardships. Thank you for all the fun times we shared together, you made our lives happier.

May Allah bless each and every one of you abundantly, and may this project be beneficial and inspiring for future generations.

Contents

Abstract	iii
Acknowledgments	iv
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Objectives	2
1.4 Methodology	2
1.5 Overview of the Startup	3
1.6 Structure of the Report	3
2 Theoretical Background	4
2.1 Overview of Cybersecurity	4
2.1.1 Types of Cybersecurity Threats	4
2.1.2 Artificial Intelligence in Cybersecurity	5
2.2 Intrusion Detection Systems	5
2.2.1 Classification of an IDS based on monitored activity	6
2.2.2 Classification of an IDS based on detection method	6
2.2.2.1 Anomaly-based network IDS	7
2.2.3 Classification of an IDS based on response upon detection	7
2.3 Internet of Things	7
2.4 Overview of Artificial Intelligence	8
2.5 Machine Learning	9
2.5.1 Machine Learning Types	9
2.5.1.1 Supervised Learning	9
2.5.1.2 Unsupervised Learning	10
2.5.1.3 Reinforcement Learning	10

2.5.2	Machine Learning Algorithms	11
2.5.2.1	Naive Bayes:	12
2.5.2.2	Decision Tree	13
2.5.2.3	Random Forest	15
2.5.2.4	Neural Networks	17
2.5.3	Evaluation Metrics	20
2.6	Implementation Tools and Technologies	21
2.6.1	Raspberry PI	21
2.6.2	ESP32 Microcontroller and ESP-IDF	21
2.6.3	Communication protocols	22
2.6.3.1	Ethernet Protocol	22
2.6.3.2	SPI Communication Protocol	22
2.6.3.3	I2C Communication Protocol	22
2.6.4	ENC28J60 Module	23
2.6.5	PCAP File Format	23
3	Hardware System Design	24
3.1	ESP32 based Prototype	24
3.1.1	System Architecture	24
3.1.2	PCAP Generator Component	25
3.1.3	Packet Feature Extractor Component	26
3.1.4	Model Inference Component	28
3.2	Raspberry Pi based Prototype	30
4	Software System Design	31
4.1	Firmware Design and Implementation	31
4.1.1	Creating and Transmitting the PCAP files	31
4.1.1.1	Default Ethernet Initialization	32
4.1.1.2	SNTP Client	33
4.1.1.3	Custom Ethernet Initialization	34
4.1.1.4	Tasks Synchronization and SD Card Access	34
4.1.1.5	PCAP Creator Task	34
4.1.1.6	PCAP Reader/Sender Task	36
4.1.2	Extracting Features from PCAP Files	37
4.1.2.1	PCAP Receiver/Writer Task	38
4.1.2.2	Feature Extractor Task	38
4.1.3	Model Inference	42

4.2	ML Model Development	43
4.2.1	Dataset Selection and Preprocessing	44
4.2.1.1	NIDS datasets	44
4.2.1.2	CIC-IoT-2023 Dataset	45
4.2.2	ML Model Building	48
4.2.2.1	Naive Bayes	48
4.2.2.2	Decision Tree	49
4.2.2.3	Random Forest	49
4.2.2.4	Neural Network	49
4.3	Raspberry Pi Implementation	51
5	Results and Analysis	54
5.1	Hypothesis on the Performance of ML Models	54
5.2	ML Models Evaluation	54
5.2.1	Naive Bayes	55
5.2.2	Neural Network	56
5.2.3	Decision Tree	57
5.2.4	Random Forest	58
5.3	ML Models Comparison	59
5.3.0.1	Explaining the Bias of the Models	60
5.4	Testing with External Pcap Files	61
6	Conclusion and Future Work	63
	Bibliography	64

List of Figures

2.1	Artificial Intelligence Subsets [17]	8
2.2	Machine learning Types and Algorithms [19]	10
2.3	Decision Trees Algorithm [25]	13
2.4	Random Forest Algorithms [27]	16
2.5	Graphical Representation of a Neuron [29]	17
2.6	Neural Networks Layers [30]	18
3.1	System Architecture Design	25
3.2	PCAP Generator Hardware Architecture	26
3.3	PCAP Generator Hardware Implementation	27
3.4	Feature Extractor Hardware Architecture	28
3.5	Feature Extractor Hardware Implementation	28
3.6	Model Inference Component Hardware Architecture	29
3.7	Model Inference Component Hardware Implementation	29
3.8	System Architecture Design	30
3.9	System Hardware Implementation	30
4.1	PCAP Generator Initialization Flowchart	32
4.2	PCAP File Packet Record Fields [36]	33
4.3	PCAP File Global Header Fields [36]	35
4.4	Flowchart of the PCAP Creator Task	36
4.5	Flowchart of the PCAP Reader/Sender Task	37
4.6	Component Initialization Flowchart	38
4.7	Flowchart of the PCAP Receiver/Writer Task	39
4.8	Flowchart of Feature Extractor Task	41
4.9	Model Inference Component Flowchart	43
4.10	One-hot Encoding Example of Categorical Features	47
4.11	Distribution of Attack Categories in CIC IoT 2023 Dataset	47

4.12 Dataset Pre-processing Steps	48
4.13 Model Layers Structure	50
4.14 Implementation on Raspberry Pi Flowchart	52
4.15 Generating the Predict() Function Flowchart	53
5.1 Comparison of F1 Scores for Different Models on Various Attacks	61
5.2 Prediction on Real-Time Traffic	61
5.3 Prediction on "DoS" Pcap File	62
5.4 Prediction on "DDoS" Pcap File	62
5.5 Prediction on Benign Traffic Pcap File	62

List of Tables

2.1	Comparison between Regression and Classification	11
2.2	Comparison between Gini index and Information Gain	15
2.3	Comparison between Decision Tree and Random Forest	17
2.4	Confusion Matrix	20
4.1	List of Features in the CIC IoT 2023 Dataset	45
5.1	Confusion Matrix Results of Naive Bayes	55
5.2	Performance Metrics for Naive Bayes	55
5.3	Confusion Matrix Results of Neural Network	56
5.4	Performance Metrics of Neural Network	56
5.5	Confusion Matrix Results of Decision Tree	57
5.6	Performance Metrics for Decision Tree	57
5.7	Confusion Matrix Results of Random Forest	58
5.8	Performance Metrics of Random Forest	58
5.9	Average Accuracy of Each Model	59
5.10	F1 Scores of ML Models Found by Enchun Shao	60

Chapter 1

Introduction

The integration of the Internet of Things (IoT) has revolutionized many industries by enabling seamless connectivity and data exchange between devices. However, it has also introduced serious network security challenges. In the context of IoT networks security, Network Intrusion Detection Systems (NIDS) offer an ideal solution, by overcoming the resources constraints of traditional security mechanisms in IoT devices. In this work, an Edge based NIDS, which leverages Machine Learning algorithms and targets IoT devices, is developed. The proposed system aims to enhance the security posture of IoT Ethernet networks by detecting potential threats in real-time.

1.1 Motivation

The rapid expansion of IoT devices, which are often not designed to handle protection against security attacks, has made them prime targets for cyber attacks. One example of a notably damaging attack is the Mirai attack, which compromised over 600,000 IoT edge devices. This attack exploited the weak security of millions of IoT devices to launch a massive Distributed Denial-of-Service (DDoS) attack against high-profile targets [1]. In addition, many other cases of compromise of edge IoT devices have been documented. In response to these threats, significant research has been conducted in this area, and many IoT compatible NIDSs have been proposed.

Machine Learning (ML), a sub-field of Artificial Intelligence, has become increasingly prominent in the cybersecurity world. Many recent Intrusion detection Systems (IDSs) are based on ML algorithms. This integration leverages the ‘learning’ capability of ML models in detecting anomalies and malicious activities in network traffic, thus enhancing the effectiveness of IDSs which are traditionally signature-based.

1.2 Problem Statement

This project aims to provide IoT networks with a security layer that doesn't introduce any overhead to the devices or network. Due to the constrained computational and memory resources of IoT devices, traditional security mechanisms are challenging to implement on such platforms. Additionally, many IoT devices that are deployed and in-use lack basic security measures, such as encrypted communications [2]. This security concern highlights the need to develop solutions that are reliable and accessible to average users. The problem addressed in this project is the creation of a network monitoring system for attacks on IoT networks, that leverages Machine Learning algorithms and is deployed on edge. The processing must be decentralized and data should remain at the network's edge. Additionally, the system should alert network users upon the detection of an intrusion.

1.3 Objectives

The objectives of this work are the following:

- Create an IoT network monitoring system that operates on low-end edge microcontrollers.
- Ensure privacy of network users by decentralizing the data processing and keeping all data locally at the edge of the network.
- Create a Machine Learning model capable of detecting anomalies and malicious activities in network traffic.
- Ensure real-time detection of intrusions in IoT networks, and real-time alerting mechanism.

1.4 Methodology

The methodology of this project involves several key steps. First, a suitable dataset is selected to align with our project requirements. Then, Naive Bayes, Decision Trees, Random Forest and Neural Network ML algorithms are used in training network intrusion detection models. Prototypes, using Raspberry Pi and ESP32, are developed to deploy the trained models on edge. Next, the performance of these models is evaluated through experiments to ensure accurate detection of network intrusions. Finally, the results are analyzed to determine the most effective approach for real-time IoT network security.

1.5 Overview of the Startup

Qareeb is a startup deep expertised in low-level firmware and advanced AI techniques to offer cutting-edge Edge Computing and IoT solutions. All their products utilize a unified, advanced edge computing protocol, ensuring seamless decentralized computing and comprehensive control over every edge box, regardless of location. Some of their popular projects are:

- **Qvision:** Uses state-of-the-art streaming technologies and AI acceleration for real-time event detection in construction sites, isolated places, commercial centers, and any location requiring real-time surveillance. Compatible with CPUs, GPUs, and any type of camera.
- **Qfarming:** Employs LoRa technology for offline data transmission on top of their edge protocol, transforming agriculture with advanced sensors and seamless data management.
- **QAccess:** Enhances security with state-of-the-art face recognition, license plate recognition, RFID, and biometric devices, all powered by their unified edge computing protocol.

Their cost-effective, scalable solutions ensure efficiency and reliability. Qareeb's end-to-end approach maximizes performance, providing an unparalleled experience in any environment. For more information, the link to their website is provided: [Qareeb Website](#). Our Network Intrusion Detection System is integrated as an additional security layer in the Advanced Healthcare System in Ambulance project.

1.6 Structure of the Report

This report is organized into six chapters. The Introduction provides an overview and motivation for the project. The Theoretical Background covers essential concepts and used technologies. The Hardware System Design details the hardware components and architecture of the proposed system. The Software System Design discusses the development of the firmware and software components, including machine learning algorithms. The Results and Analysis presents the results of the experiments with various machine learning algorithms and their evaluation. Finally, the Conclusion and Future Work summarizes the findings and suggests directions for further work.

Chapter 2

Theoretical Background

2.1 Overview of Cybersecurity

Cybersecurity is the process of protecting networks, systems, data, and programs from unauthorized access or criminal exploitation. Threat actors, who represent a security risk, typically attack sensitive information of individuals and organizations such as credit card details, personal data, and trade secrets, for financial gain and various other motives. Cybersecurity aims at ensuring the safety of digital and physical assets, and managing cyber risks that may arise [3]. The field of cybersecurity is built upon three principles which represent the fundamental objectives for information security. These three concepts form what is often referred to as the CIA triad [4]:

- **Confidentiality:** Ensuring that information is accessible only to those authorized to access it.
- **Integrity:** Ensuring that the data is not tampered with, correct and reliable.
- **Availability:** Ensuring timely and reliable access to data for those who are authorized to access it.

2.1.1 Types of Cybersecurity Threats

Cybersecurity threats are diverse and continuously evolving. Some of the most common threats include [3]:

- **Malware:** is software that is designed to harm and cause damage to a network or computer. Common types of malware are: viruses, ransomware and spyware.

- **Phishing:** is the use of digital communication, such as emails and voice communication, to steal sensitive information by tricking users into believing their sources to be reputable.
- **Social Engineering:** is a manipulation technique that adversaries use to trick users into revealing their sensitive information.

Additionally, cyber threats can be categorized based on their threat consequences [4]. For instance, Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are threats to availability of systems, whereas Masquerading attacks such as: ARP Spoofing, Man in the Middle (MitM) are threats to both the confidentiality and integrity of information. Reconnaissance attacks, such as Port Scan and Vulnerability Scan attacks, are primarily threats to confidentiality. Other Web-based attacks, such as SQL Injection and Cross-Site Scripting (XSS), threaten all mentioned aspects of cybersecurity: confidentiality, integrity and availability.

2.1.2 Artificial Intelligence in Cybersecurity

Artificial Intelligence (AI) has become an innovative technology in many industries, including cybersecurity. This technology offers both opportunities and challenges in this field. While AI can be used to enhance network protection and ensure user safety, it can be exploited by adversaries to cause harm and disruption [5]. For more than twenty years, the cybersecurity industry has been leveraging Artificial Intelligence (AI) in different areas like spam filtering, malware detection, and intrusion detection. This integration has significantly enhanced performance by automating cybersecurity tasks, ensuring efficient resource utilization, and discovering new threats and attacks [6]. Sections 2.4 and 2.5 below provide detailed information about AI and explore its sub-fields and techniques.

2.2 Intrusion Detection Systems

As network attacks have increased in number and severity over the past few years, intrusion detection systems have become an integral part to the security infrastructure of most organizations. Intrusion detection systems (IDSs) are devices or software programs that monitor network traffic or system events and analyze them for any signs of intrusions. Intrusions are defined as any attempts to compromise the confidentiality, integrity, availability (CIA) of a computer or network, or to bypass its security mechanisms [7]. IDSs are essential for protecting digital assets and facilitating the handling of security breaches. They can be

categorized into two main types: Network-based Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS), both designed to identify any malicious attempt to penetrate a system's defenses [8].

2.2.1 Classification of an IDS based on monitored activity

Intrusion detection systems can be grouped into two main types: NIDS (Network-based Intrusion Detection Systems) and HIDS (Host-based Intrusion Detection Systems). This classification takes into consideration the kind of activity the IDS monitors.

Network-based intrusion detection systems are devices placed within a network to analyze traffic moving through its hosts. Network-based IDSs exist in the form of software or hardware-based systems [9]. A computer network interface card (NIC) typically operates in non-promiscuous mode, meaning it only receives packets whose destination Media Access Control (MAC) addresses are either the NIC's MAC address or a broadcast address. However, the network interface card of a NIDS operates in promiscuous mode, receiving all network traffic regardless of the destination MAC addresses [10]. Typical problems with NIDS are high false positive and false negative rates, and the inability to detect attacks within encrypted traffic and during periods of high traffic in busy networks [11] [7].

Host-based intrusion detection systems are deployed on single computer systems to monitor local events and protect against internal and external cyberattacks [12]. Host-based IDS data sources are typically of two types: operating system audit trails and system logs. The first are generated at the kernel level, which makes them more protected than system logs but also very long and detailed. System logs on the other hand are smaller, less cryptic, and considerably simpler to comprehend [7]. These data help the HIDS track changes made to registry settings, critical system configuration, and log and content files, alerting to any unauthorised or anomalous activity [13]. The major drawback of a HIDS is its high consumption of host resources [11].

2.2.2 Classification of an IDS based on detection method

The detection method lies at the heart of the IDS, as it specifies how the system detects intrusions. The IDS employs two types of detection methods: signature-based and anomaly-based detection methods [9]. A signature-based IDS works by comparing monitored activities against a database of pre-defined pattern of events known as "attack signatures".

Conversely, an anomaly-based IDS establishes a baseline of normal (legitimate) activities and identifies deviations from this baseline as potential intrusions [7].

2.2.2.1 Anomaly-based network IDS

Intrusion detection systems usually leverage anomaly detection capabilities to model normal network traffic behavior and detect deviations as potential signs of attacks. This approach is especially effective against obvious attacks which generate significant change in network traffic, such as Port and Service Scanning or Denial-of-Service (DoS) attempts. Anomaly detection is much less effective at detecting stealthy attacks, which are more subtle and resemble normal communications. Detecting subtle attacks would require tuning anomaly detection to be more sensitive, which would lead to many false positive alarms. To address this type of threat, Classification methods, which provide better accuracy, are used. Binary classification help distinguishing benign from malicious traffic, while multi-class classification is used to identify different types of attacks and malicious behaviors in a fine-grained manner. As network traffic is diverse with abundant communication protocols, the modeling of network environments and behaviors is becoming increasingly difficult. This is why *“AI applications to network security are the most successful in monitoring simple, stable, and predictable network environments composed of low-end devices with simple behaviors, such as Internet-of-Things (IoT) networks. . . ”* [6].

2.2.3 Classification of an IDS based on response upon detection

After the analysis of collected events or network information for attack indicators, intrusion detection systems proceed to generate appropriate responses. These responses can range from reporting the findings to a designated location, to implementing more proactive automated actions. Commercial IDSs typically offer a broad spectrum of response options, which are commonly classified as active responses, passive responses, or a combination of both. An active IDS triggers an automated countermeasure action to prevent further escalation of certain types of attacks. A passive IDS only generates alerts to inform users when malicious activity is detected [7].

2.3 Internet of Things

The Internet of Things (IoT) refers to the interconnected network of physical devices embedded with sensors, software, and other technologies to collect and exchange data over the internet. This network includes a vast range of devices, from household appliances to

industrial machinery, enabling smarter environments and more efficient operations. However, the rapid proliferation of IoT devices has also introduced serious security challenges. IoT networks are particularly vulnerable to cyberattacks due to their limited computing resources, lack of established security standards, and the sheer number of interconnected devices, which provide multiple entry points for attackers [14].

2.4 Overview of Artificial Intelligence

The term ‘Artificial Intelligence’ broadly refers to applications of technology able to perform tasks that resemble human cognitive function, and the capability of a machine to imitate intelligent human behavior. It involves the development of computer systems that are able to perform tasks which normally require human intelligence, such as visual perception, speech recognition, and decision-making [15]. While the presented definition provides a general outline of the meaning of the term, there is not a universally accepted definition for AI. The concept and parameters of AI are subject to debate and continual evolution, “A lot of cutting-edge AI has filtered into general applications, often without being called AI because once something becomes useful enough and common enough it is not labeled AI anymore” [16].

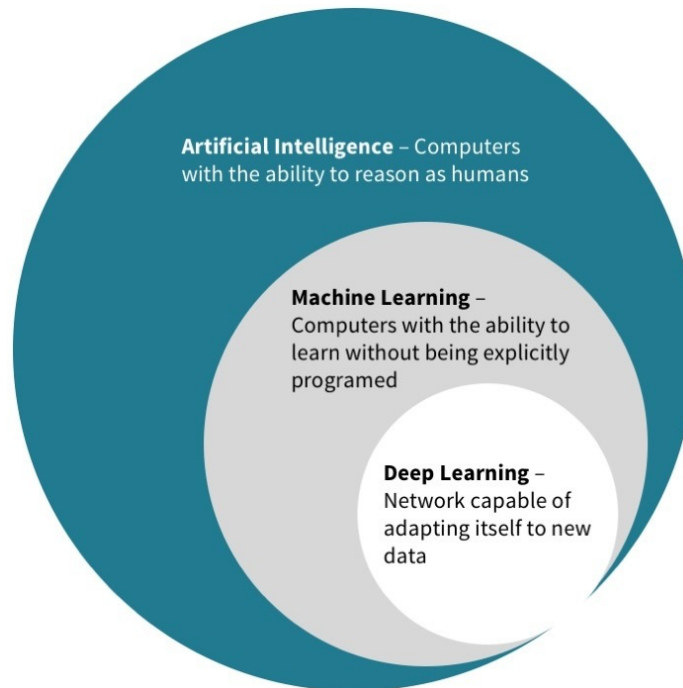


Figure 2.1: Artificial Intelligence Subsets [17]

As seen in Figure 2.1, Artificial Intelligence is the concept of creating smart intelligent machines, while Machine Learning is a subset of it which enables the development of AI-driven systems that automatically learn and improve from experience. Furthermore, Deep Learning is a subset of Machine Learning that uses vast volumes of data with complex algorithms, such as Neural Networks, in training AI models.

2.5 Machine Learning

Machine Learning, a subset of Artificial Intelligence, focuses on the creation of algorithms that allow a computer to learn from data and previous experiences, without the need for explicit programming. Artificial Intelligence, particularly Machine Learning, has grown rapidly in recent years in the context of data analysis and computing, enabling applications to function in an intelligent manner. It was defined in the 1950s by AI pioneer Arthur Samuel, who popularized the term and is best known for his program that played championship level checkers, as *“the field of study that gives computers the ability to learn without explicitly being programmed.”* [18].

Machine learning contains a set of algorithms that operate on a vast amount of data. These algorithms are fed with data to be trained on, and through this training process, they construct models to perform specific tasks. Hence, Machine Learning primarily relies on two essential elements: a large collection of data, and the algorithm responsible for its processing.

2.5.1 Machine Learning Types

Machine Learning can be broadly classified into three main types, which are the most used, based on their methods and learning approaches. These types include: Supervised Learning, Unsupervised Learning, and Reinforcement Learning, all shown in Figure 2.2. Each type is best suited for specific problems and datasets, and employs specific algorithms in building its output model.

2.5.1.1 Supervised Learning

Supervised Learning is a method in machine learning used to understand how inputs and outputs are connected in a system. It works by using pairs of input-output examples that are already known. The output is seen as the ‘label’ or the ‘dependent variable’ of the input data. In supervised learning, the aim is to create a computer system that understands how inputs relate to outputs. This system learns from examples where both the input and the

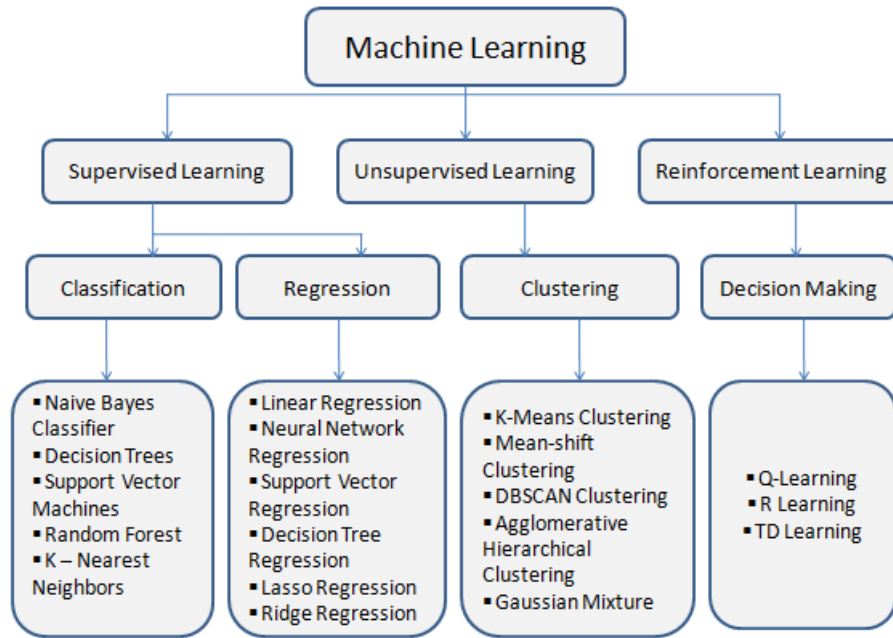


Figure 2.2: Machine learning Types and Algorithms [19]

correct output are known. Once trained, the system can then predict the output for new inputs. If the output takes a finite set of discrete values indicating the class labels of the input, the learned mapping leads to the classification of the input data. If the output takes continuous values, it leads to a regression of the input [20].

2.5.1.2 Unsupervised Learning

Unsupervised Learning operates differently from supervised learning. In unsupervised learning, there are no clear labels for the data. Instead, the focus is on allowing the model to find patterns and make predictions independently. The machine is given data and is expected to discover hidden features based on the similarities it identifies.

This approach, which uses input vectors without corresponding target values, continues to evolve. Unsupervised learning has seen remarkable developments in its ability to group and interpret information based on similarities, patterns, and differences [21]. However, unsupervised learning is computationally complex and requires a large amount of data.

2.5.1.3 Reinforcement Learning

Reinforcement Learning involves the process of learning optimal behaviors within an environment to maximize rewards. Unlike supervised or unsupervised learning, where data

is provided as input, in reinforcement learning, data is accumulated through trial-and-error methods. In supervised learning, the training data includes the answer key, which allows the model to learn from correct answers directly. However, in reinforcement learning, the agent determines its actions to achieve a given task. In the absence of a training dataset, it is bound to learn from its experience by receiving feedback from the environment in the form of rewards or punishments. This method mirrors the way children learn, by performing actions and observing the outcomes, using feedback to refine their understanding of the environment [22].

2.5.2 Machine Learning Algorithms

Supervised learning is divided up into two fundamental categories: Regression and Classification. They are both employed in machine learning to generate predictions based on labeled data. The key distinction between the two lies in the nature of predictions they output. Regression algorithms predict continuous values, such as prices, salaries, or ages. Conversely, classification algorithms predict discrete values, such as gender (male/female), truth value (true/false), spam detection (spam/not spam), and more.

Table 2.1: Comparison between Regression and Classification

Regression	Classification
Output variable is continuous	Output variable is discrete
It finds the best fit for accurately predicting the output	It finds the decision boundary that classifies the datasets into different groups
It aims to forecast or predict	It aims to find the right category
The Regression algorithm can be further divided into Linear and Non-linear Regression.	The Classification algorithms can be divided into Binary Classifiers and Multi-class Classifiers.

Classification: Classification is used to predict a categorical output, assigning data instances to predefined groups or classes depending on their features. It encompasses three subcategories: Binary classification, multi-class classification, and multi-label classification. Some of the popular algorithms used in classification are described below:

- **Logistic Regression:** is a method used to generate the probability of a discrete outcome given an input variable. It is useful when the dependent variable is binary

but the independent variables are continuous. The term "regression" in its name is derived from its relation to linear regression.

- **Decision Tree Classification:** as mentioned above, this technique is suitable for both regression and classification.
- **Random Forest Classification:** a method that aggregates the predictions of multiple decision trees to make a final classification. It employs a majority vote, where the class predicted by the majority of individual trees is considered the final prediction. This approach improves the accuracy and robustness of the classification.
- **Support Vector Classifier:** this method aims to find the optimal fit line (or surface in higher dimensions) that separates data points into different classes with the maximum margin. The margin represents the distance between the hyperplane and the nearest data points from each class, known as support vectors.

2.5.2.1 Naive Bayes:

Naive Bayes is a simple and effective probabilistic classifier based on Bayes' Theorem, that is well-suited for high-dimensional datasets. Naive Bayes classifiers assume that the features in a dataset are independent given the class label. This assumption of independence, termed 'naive', simplifies the computation, although it may not always hold true in practice as features may be correlated. Despite this simplification, Naive Bayes performs well in practice [23].

Bayes' theorem is stated mathematically in the following equation:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad (2.1)$$

where:

$P(A|B)$ = the probability of event A happening given that B is true. For naive bayes, A is the class, and B is the set of features.

$P(A)$ = Prior Probability of event A. It is the overall probability of each class.

$P(B|A)$ = Likelihood. It is the probability of each feature given each class.

During training, Naive Bayes calculates the prior probabilities $P(A)$ for each class and the likelihoods $P(B|A)$ for each feature given the class. These probabilities are estimated from the training data.

Naive Bayes assumes that all features are independent given the class label. This means that the joint probability $P(A|B)$ can be written as the product of the individual probabilities of each feature:

$$P(A|x_1, x_2, \dots, x_n) = P(A) \cdot P(x_1|A) \cdot P(x_2|A) \cdot \dots \cdot P(x_n|A) \quad (2.2)$$

For a new instance, Naive Bayes calculates the posterior probability for each class using the previously computed priors and likelihoods. The class with the highest posterior probability is chosen as the prediction (the correct class).

2.5.2.2 Decision Tree

A Decision Tree is a supervised learning method used for both classification and regression tasks. “A *decision tree* is a classifier expressed as a recursive partition of the instance space.” [24]. It has a hierarchical structure resembling a tree which consists of a root node, branches, internal nodes, and leaf nodes (also known as terminal or decision nodes). The model is built by making decisions based on features, where each internal node represents a feature test, the branches indicate decision rules, and the leaf nodes represent the predicted class or value.

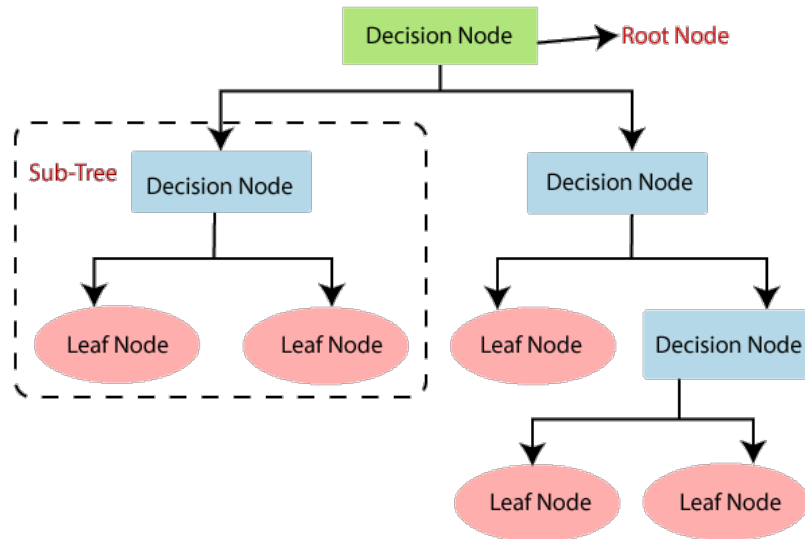


Figure 2.3: Decision Trees Algorithm [25]

Creating a decision tree involves iteratively dividing the data into smaller groups based on various attributes. At each step, the algorithm selects the most suitable attribute for further division, typically using criteria like Information Gain or Gini Impurity. This division process continues until certain conditions are met, such as reaching a maximum depth or

having a minimum number of instances in a group. One the key challenges in decision tree construction is the identification of the attribute for the root node at each level. This process, known as attribute selection, is addressed using impurity-based criteria. These criteria help in deciding which feature to choose for splitting the data.

The impurity-based criteria that are commonly used are:

Information gain: Information gain tells how effectively a feature can split the data into different classes or groups. It quantifies the amount of uncertainty or randomness reduced by splitting the data based on a particular feature. Higher information gain implies that splitting the data based on that feature leads to more accurate predictions or classifications. The mathematical equation for Information Gain is denoted below:

$$IG(S, A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \cdot Entropy(S_v) \quad (2.3)$$

where:

S = a set of instances

A = an attribute

S_v = the subset of S

v = represents an individual value that the attribute A can take

Entropy is the measure of uncertainty in a dataset. It calculates the impurity of a collection of examples. A lower entropy indicates less disorder and higher predictability. It represents how much the data is random. It is given by the following equation:

$$H(S) = - \sum_{i=1}^n p_i \log_2(p_i) \quad (2.4)$$

where:

p_i = the proportion of examples in class i to the total number of examples in dataset S
with n classes

Gini Index: Gini index represents the likelihood of misclassification for a randomly chosen element within the dataset. Lower Gini index values indicate a lower impurity and better separation of classes in the dataset. Its formula is given by:

$$Gini(S) = 1 - \sum_{i=1}^n p_i^2 \quad (2.5)$$

where:

p_i = the proportion of examples in class i to the total number of examples in dataset S with n classes

Table 2.2: Comparison between Gini index and Information Gain

Information Gain	Gini Index
Multiplies probability of each class by log base 2 of that class probability	Subtracts the sum of square of probabilities of each class from 1
It works well with smaller partitions in the dataset	It works well with large partitions in the dataset
Used in calculating the further split of the tree by ID2, C4.5 algorithm	Used in calculating the further split of the tree by the CART algorithm

Building a decision tree in machine learning requires the following steps:

- Selecting the best split by evaluating potential splits using criteria like Gini Impurity or Information Gain for each feature.
- Creating nodes by splitting the data, based on the attribute above, into subsets and create nodes in the tree.
- Recursively repeating the process for each subset until a stopping criterion is met (e.g. maximum depth, minimum samples per leaf, or no further improvement).

2.5.2.3 Random Forest

Random Forest is a ML algorithm comprising of a large number of individual decision trees operating as an ensemble. Each tree in the forest generates a class prediction, and the class with the highest number of votes becomes the model's prediction [26]. The underlying principle is that a group of weak learners (individual trees) can collaboratively form a strong learner, improving prediction accuracy and controlling overfitting compared to a single decision tree.

Random Forests are used for both classification and regression tasks. They operate by constructing multiple decision trees during training and outputting the mode of the classes (for classification) or the mean prediction (for regression) from the individual trees [26].

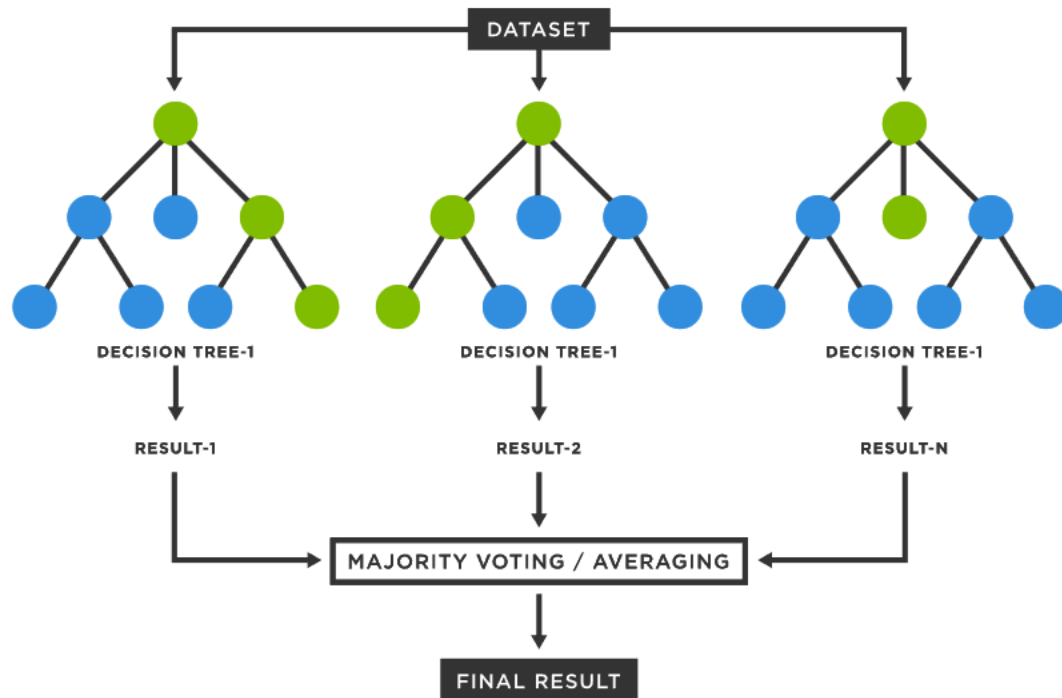


Figure 2.4: Random Forest Algorithms [27]

The following describes the process by which a random forest is trained:

- Randomly selecting multiple subsets from the original dataset, along with features to build several decision trees.
- For each bootstrap sample, or subset, a decision tree is trained. These trees are also labeled as weak learners or base learners.
- When training each decision tree, a subset of features is randomly selected at each split or node. This will help introduce variability among the trees, which reduces overfitting.
- Once all trees are trained, they are combined to form the random forest. Predictions of this random forest will depend on a majority voting of the output of each decision tree for classification tasks, and on an averaging for regression tasks. This approach helps improve accuracy and reduce overfitting and variance.

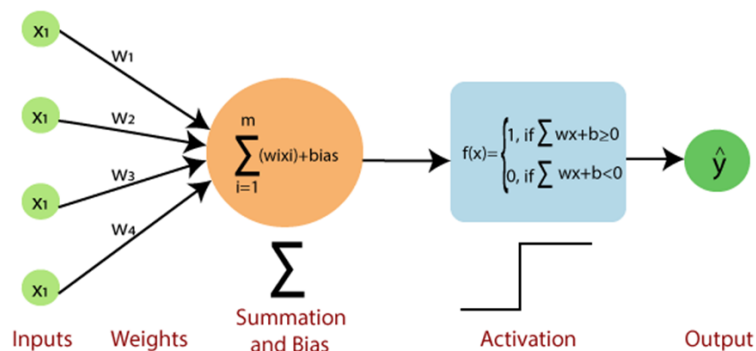
Table 2.3: Comparison between Decision Tree and Random Forest

Aspect	Decision Tree	Random Forest
Structure	Single tree with a hierarchical structure	Ensemble of multiple decision trees
Performance	Prone to overfitting, with complex data	Less prone to overfitting due to ensemble approach
Interpretability	Easy to interpret and visualize	Difficult to interpret due to the numerous trees
Training and predicting time	Faster training time	Slower training time due to multiple trees
Accuracy	Can be accurate with simpler datasets	Generally more accurate, with complex data

2.5.2.4 Neural Networks

Traditional ML algorithms rely on predefined assumptions about the data and often require feature engineering to perform well. However, as data becomes more complex, these algorithms may struggle to capture hidden patterns, leading to reduced prediction accuracy. This is where Neural Networks can address these limitations. Inspired by the structure and function of the human brain, Neural Networks are capable of modeling complex relationships and learning complicated representations and patterns.

Neural networks, a subset of ML algorithms, consist of layers of interconnected nodes, or neurons, where each connection is assigned a weight. An artificial neuron takes multiple inputs, computes a weighted sum, and then applied an activation function to produce an output [28].

**Figure 2.5:** Graphical Representation of a Neuron [29]

The mathematical expression for a perceptron is:

$$y = \phi \left(\sum_{i=1}^n w_i x_i + b \right) \quad (2.6)$$

where:

y = the output

ϕ = the activation function

w_i = the weights

x_i = the input features

b = the bias term

n = the number of input features

Weights: In a perceptron, or artificial neuron, weights are the parameters that determine the importance of each input feature, as each input is multiplied by its corresponding weight. The aim of the weights is to scale the inputs appropriately based on their relevance to the output.

Bias: The bias allows the model to fit the data better by shifting the activation function. It can be considered as the y-intercept in a linear equation. It ensures that the perceptron can model relationships that do not pass through the origin. This allows to generate an output even when all input features are zero. Like weights, the bias is also adjusted during the training process to improve the perceptron's performance.

A neural network is composed of layers: an input layer, one or more hidden layers, and an output layer as shown in figure 2.6.

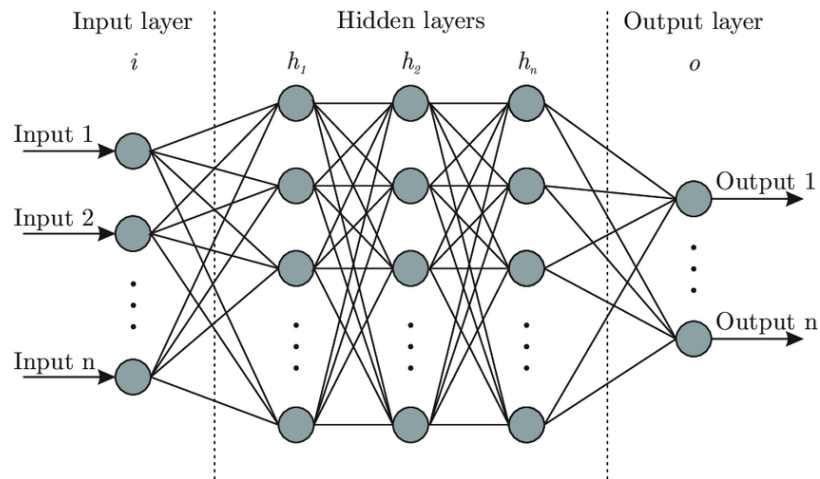


Figure 2.6: Neural Networks Layers [30]

- **Input Layer:** The input layer is the first layer in a neural network. It receives the raw data and passes it on to the next layers. Each neuron in the input layer represents a feature of the data. For example, in image recognition, each pixel of an image might be an input neuron [28].
- **Hidden Layers:** Hidden layers are intermediate layers between the input and output layers. They perform computations on the received data from the previous layer. The neurons in hidden layers apply weights to the inputs, sum them up, and pass the result through an activation function to introduce non-linearity to learn complex patterns. There can be multiple hidden layers in a neural network. Networks with more than one hidden layer are known as Deep Neural Networks [31].

For a given neuron j in the first hidden layer, the output is calculated as:

$$z_j = \sum_{i=1}^n w_{ij}x_i + b_j \quad (2.7)$$

Where w_{ij} is the weight connecting input x_i to neuron j , and b_j is the bias term.

The result z_j is then passed through an activation function ϕ to introduce non-linearity:

$$a_j = \phi(z_j) \quad (2.8)$$

This process is called forward propagation.

- **Output Layer:** The output layer is the final layer of the neural network. It produces the network's predictions. The number of neurons in the output layer depends on the type of problem being solved. For binary classification, there is one output neuron, while for multi-class classification, there are multiple output neurons, each representing a different class [28].

The weights and biases must be updated regularly in order to achieve the correct output. The obtained output is compared to the desired result and the parameters are adjusted accordingly. This procedure is called Back Propagation, which involves these steps:

- The first step is to calculate the loss between the predicted output and the actual target. A common loss function is the Mean Squared Error (MSE):

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (h_w(x^{(i)}) - y^{(i)})^2 \quad (2.9)$$

where:

- m = the number of training examples
- $x^{(i)}$ = the training example input
- $y^{(i)}$ = the actual output
- $h_w(x^{(i)})$ = the predicted output, given the weights w and biases b .

- The weights and biases are then updated to minimize the loss:

$$w_{\text{new}} = w_{\text{old}} - \alpha \frac{\partial J}{\partial w} \quad (2.10)$$

$$b_{\text{new}} = b_{\text{old}} - \alpha \frac{\partial J}{\partial b} \quad (2.11)$$

α : the learning rate or the step size at which the weights are updated during the training process.

2.5.3 Evaluation Metrics

Evaluation metrics are tools used to measure the performance of a ML model in various aspects. A fundamental component in evaluating the model is Confusion Matrix. It calculates the performance of a ML model on the test data samples. It is used to display the number of accurate and inaccurate instances based on the model's predictions. Confusion matrix analyses true positive, true negative, false positive, and false negative predictions. When there is an uneven class distribution in a dataset, this matrix is useful in evaluating a model's performance beyond the usual basic accuracy metrics.

Table 2.4: Confusion Matrix

Term	Definition
True Positives (TP)	Correctly predicted positive cases
True Negatives (TN)	Correctly predicted negative cases
False Positives (FP)	Incorrectly predicted positive cases
False Negatives (FN)	Incorrectly predicted negative cases

- **Accuracy:** it measures the correct predictions out of the complete test dataset.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.12)$$

- **Precision:** the ratio of correctly predicted positives to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2.13)$$

- **Recall (Sensitivity):** the ratio of correctly predicted positives to all the observations in the actual class.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.14)$$

- **F1-Score:** it is the harmonic mean of Precision and Recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.15)$$

2.6 Implementation Tools and Technologies

2.6.1 Raspberry PI

The Raspberry Pi is a single-board computer (SBC) that is interoperable with any input and output hardware device like a monitor, a mouse, or a keyboard. It was developed by the Raspberry Pi Foundation with the goal of making computing accessible to more people. Raspberry Pi has evolved through several models, each offering improvements in processing power, memory, and connectivity. The latest models, such as Raspberry Pi 4, come equipped with a quad-core ARM Cortex-A72 processor, up to 8GB of RAM, USB 3.0 ports, and dual 4K HDMI outputs [32].

2.6.2 ESP32 Microcontroller and ESP-IDF

The ESP32 [33] is a low-power system on a chip (SoC) with integrated Wi-Fi and Bluetooth capabilities, developed by Espressif Systems. The ESP32 features a dual-core (or single-core) Xtensa 32-bit LX6 microprocessor, a range of peripheral interfaces including multiple GPIO pins, ADCs, DACs, UART, SPI, I2C, and PWM interfaces, and a powerful wireless transceiver, making it an ideal choice for Internet of Things (IoT) projects. Espressif IoT Development Framework (ESP-IDF) [34], is the official development framework for the ESP32 chip. It provides an open-source platform for building applications for the ESP32 family of devices. ESP-IDF includes all the necessary tools, libraries, and APIs to develop, debug, and deploy applications, making it a powerful resource for developers working on Internet of Things (IoT) projects which use the ESP32 SoC microcontroller.

2.6.3 Communication protocols

2.6.3.1 Ethernet Protocol

Ethernet is a communication protocol that links computers and ethernet-capable devices in a network via a wired connection. It is used in both local area network (LAN) and wide area network (WAN) systems. Normal IEEE 802.3 compliant Ethernet frames are between 64 and 1518 bytes in length. They are made up of five or six different fields: a destination MAC address (DA), a source MAC address (SA), a type/length field, data payload, an optional padding field and a Cyclic Redundancy Check (CRC). Additionally, when transmitted on the Ethernet medium, a 7-byte preamble field and Start-of-Frame (SOF) delimiter byte are appended to the beginning of the Ethernet packet [34].

2.6.3.2 SPI Communication Protocol

Serial Peripheral Interface (SPI) is a synchronous serial communication protocol that is generally used for short distances. This protocol allows a full duplex communication between the master and the slaves, as well as the use of more than one slave device with only one master. A slave is addressed using an input pin on its interface: CS (Chip Select). The addressing is taken care of exclusively by the hardware. A simple SPI interface uses 4 lines for communication [34], which are:

- CLK (Clock): Oscillating signal generated by a Host that keeps the transmission of data bits in.
- MOSI: Master Output Slave Input.
- MISO: Master Output Slave Input.
- CS (Chip Select): Allows a Host to select individual Device(s) connected to the bus in order to send or receive data.

2.6.3.3 I2C Communication Protocol

The Inter-Integrated Circuit (I2C) is a synchronous, half-duplex, serial communication protocol that allows co-existence of multiple masters on the same bus, as well as multiple slaves [34]. Only two lines are needed for communication, which are:

- SCL (Serial clock): Provides the clock signal for synchronous communication.
- SDA (Serial Data): data lines for all the devices.

2.6.4 ENC28J60 Module

The ENC28J60 is a stand-alone Ethernet controller with an industry standard Serial Peripheral Interface (SPI). It is designed to serve as an Ethernet network interface for any controller equipped with SPI. The ENC28J60 meets all of the IEEE 802.3 specifications. It incorporates a number of packet filtering schemes to limit incoming packets. It also provides an internal DMA module for fast data throughput and hardware assisted checksum calculation, which is used in various network protocols. Communication with the host controller is implemented via an interrupt pin and the SPI, with clock rates of up to 20 MHz. Two dedicated pins are used for LED link and network activity indication [35].

2.6.5 PCAP File Format

PCAP files are a common format for storing packet captures. A PCAP file includes an exact copy of every byte of every packet as seen on the network, including OSI layers 2-7. The PCAP file format was developed alongside the development of the tcpdump and libpcap libraries in the 1990s. As these libraries became the de facto standard of network capturing on Unix systems, this format became the most common format in the open source world for capture files from any copper, fibre, or wireless network [36] [37].

Chapter 3

Hardware System Design

In this chapter, a general overview of the hardware components and architectures of two proposed prototypes for this project is presented. The prototypes are based on different hardware platforms: Espressif's ESP32 and the Raspberry Pi.

3.1 ESP32 based Prototype

3.1.1 System Architecture

Our ESP32-based network intrusion detection system is designed to inspect the contents and header information of Ethernet network packets that are captured using the ENC28J60 Ethernet module. Relevant features are then extracted from these packets and fed to a machine learning model. As shown in Figure 3.1, the system consists of 3 main components:

- Ethernet Packet Sniffer and PCAP Generator
- Packet Feature Extractor
- Model Inference

Each component in the system is responsible of communicating the necessary data that the next component relies on. This establishes the desired data flow through the system, which ensures its proper functioning.

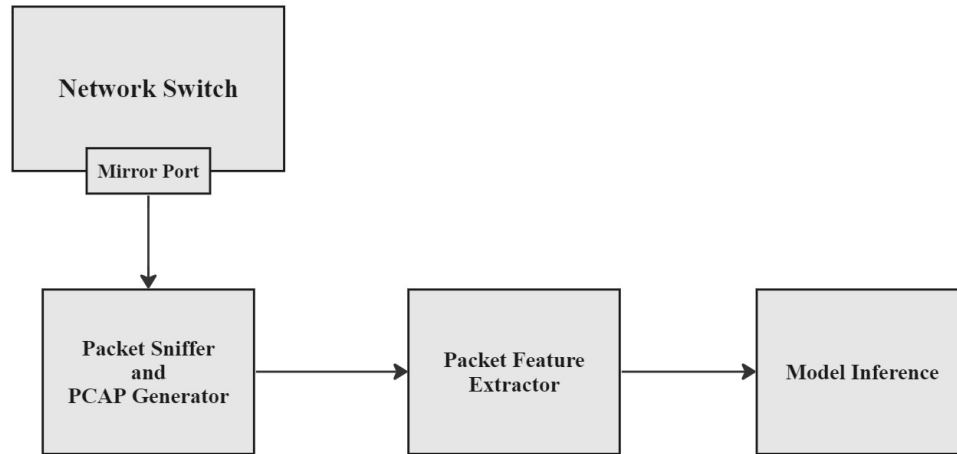


Figure 3.1: System Architecture Design

3.1.2 PCAP Generator Component

The Ethernet Packet Sniffer and PCAP Generator component, built around the ESP32, provides the Packet Feature Extractor component with PCAP files that are processed to extract relevant features. The PCAP files are generated from the network's traffic that is sniffed by the ESP32. The Packet Sniffer is connected to a local network via an Ethernet Switch, to allow for the monitoring of traffic of the different IoT devices that are connected to the network.

The ESP32 microcontroller interfaces an ENC28J60, which exposes an SPI interface for communication, using the VSPI (SPI3) peripheral. The ENC28J60 enables our microcontroller to establish an Ethernet network connection by interfacing a SPAN/Mirror port on a network switch; The SPAN port replicates the network traffic, by means of the switch's operating system, thereby allowing the ENC28J60 to receive a mirrored copy of all packets passing through the source ports of the switch. On the other side of the SPAN port, the ESP32's Ethernet interface is configured to work in promiscuous mode, which leverages the mirroring effect created by the SPAN port and enables the system to capture and analyze all network packets, regardless of their destination or source physical addresses.

Storage space is a crucial requirement for the ESP32 to transform the captured Ethernet frames into PCAP files. There are various storage options for the ESP32. One option is to partition the ESP32's flash memory into new sections, and store the generated PCAP files in a free section under a mounted filesystem, such as SPIFFS. Although this approach

requires fewer hardware components, it risks wearing out the ESP32's NOR flash memory, which is typically rated for between 10,000 to 100,000 write cycles. Given our use case involves frequent data writes, a Secure Digital (SD) card, which is a NAND-based flash memory, is used. An SD card has a greater lifespan and can be safely overwritten many more times. Thus, the ESP32 interfaces with an SD card module using the HSPI (SPI2) controller, and mounts a FAT filesystem on the card to perform standard file manipulation operations.

After capturing a specified number of Ethernet frames and creating a PCAP file, the ESP32 transmits the file to the next component for feature extraction. This transmission uses I2C communication with the ESP32 as the master and the next component's microcontroller as the slave, by using the I2C0 controller in master mode.

The interconnections of the hardware components are demonstrated in Figure 3.2. Additionally, the physical implementation of this component is shown in Figure 3.3.

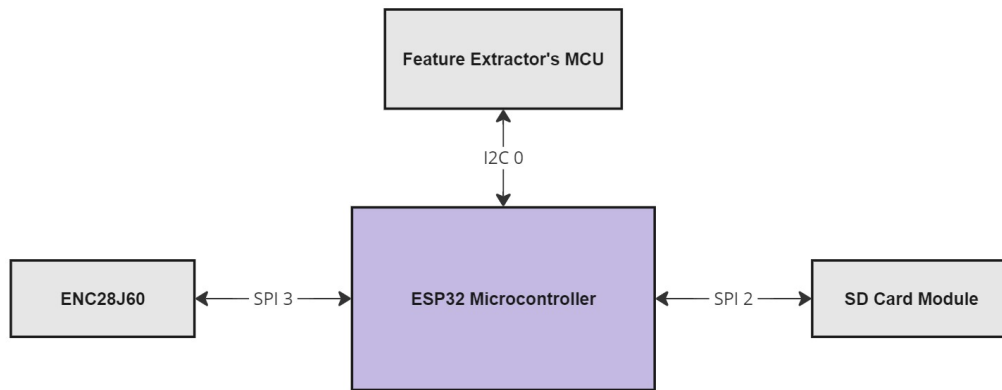
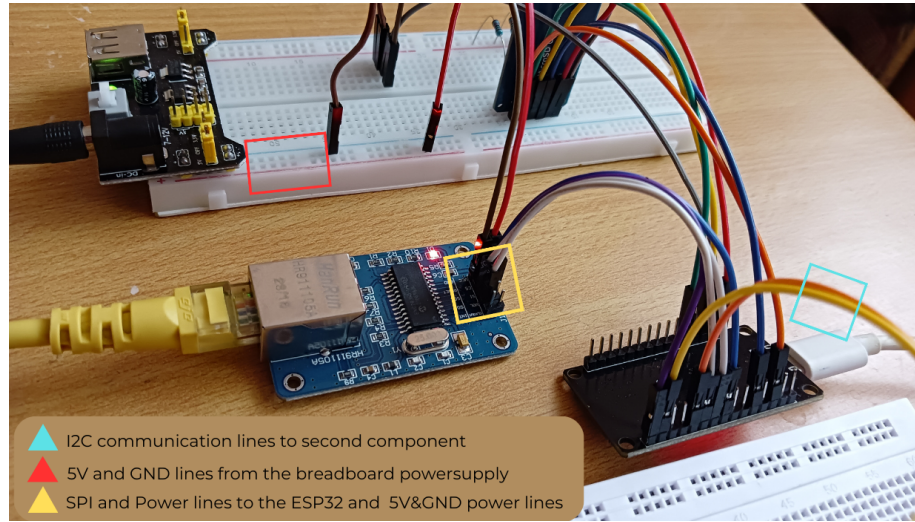


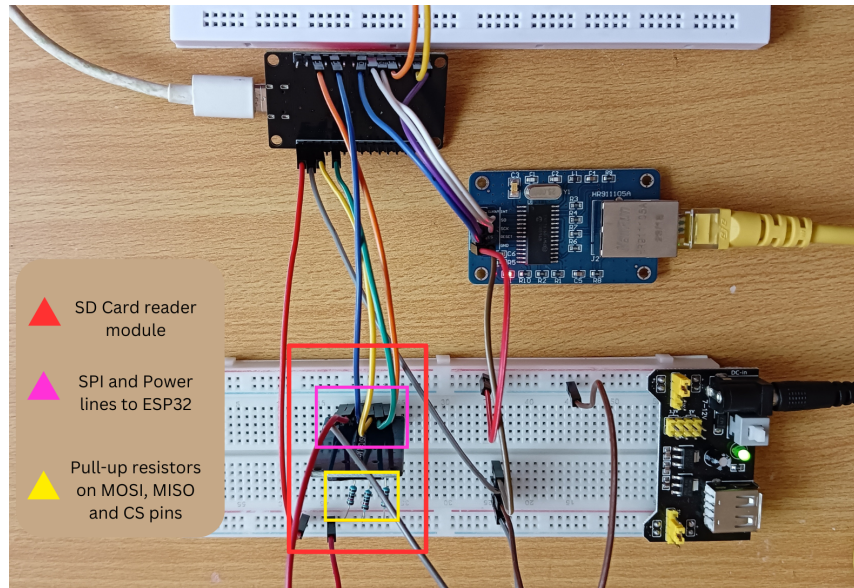
Figure 3.2: PCAP Generator Hardware Architecture

3.1.3 Packet Feature Extractor Component

The Packet Feature Extractor component functions as a pre-processing unit, responsible for extracting essential features from network PCAP files and supplying these features to the machine learning model.



(a) Side view of the implementation showing the ENC28J60 interfacing



(b) Top view of the implementation showing the SD card module interfacing

Figure 3.3: PCAP Generator Hardware Implementation

This component includes an ESP32, which processes PCAP files received from the PCAP Generator component. In this setup, the ESP32 operates as an I2C slave when communicating with the PCAP Generator MCU, using the I2C0 controller in slave mode. Due to the need for file manipulation during processing, storage space is necessary. To meet this requirement, the ESP32 interfaces with an SD card module using the VSPI (SPI3) bus.

Once the processing is complete, the extracted features are transmitted to the final component for machine learning model inference. Similar to its communication with the first

component, the MCU of this component also employs the I2C protocol to communicate with the subsequent component using its I2C1 controller in master mode.

Figure 3.4 showcases the hardware components layout and connections of this component, while Figure 3.5 illustrates its physical implementation.

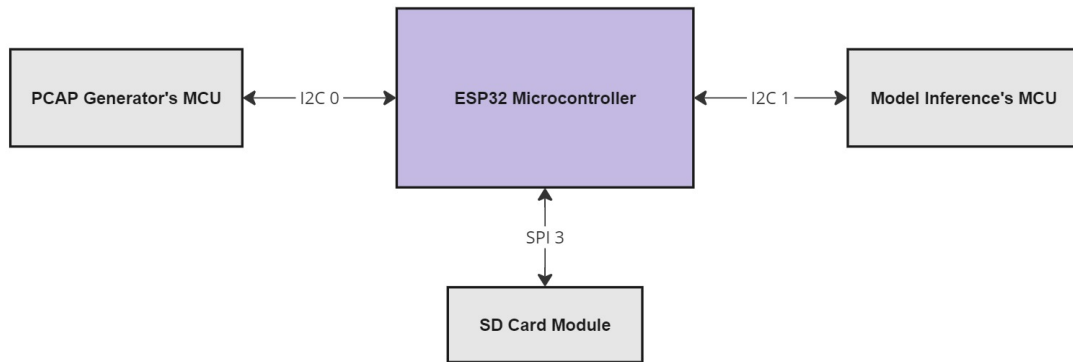


Figure 3.4: Feature Extractor Hardware Architecture

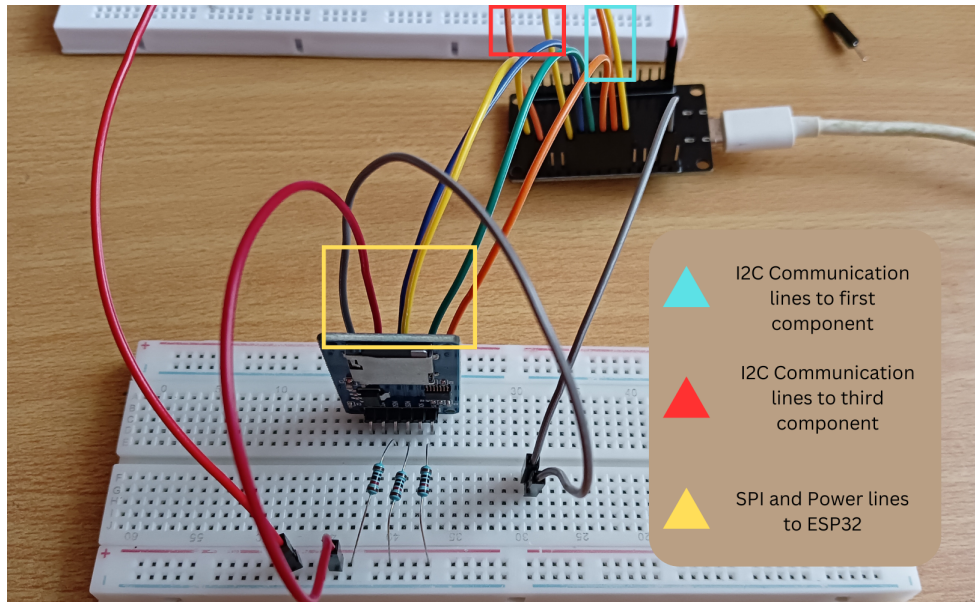


Figure 3.5: Feature Extractor Hardware Implementation

3.1.4 Model Inference Component

The Model Inference component is responsible for running the trained machine learning model using extracted features from PCAP files as input.

This component is centered around an ESP32 microcontroller which does the inference of the model. The ESP32 receives input features from the PCAP Feature Extractor's microcontroller by means of an I2C communication, in which the ESP32 is the slave. In addition to the I2C0 controller that is used in this communication, the ESP32 interfaces a buzzer and two LEDs which are used in the established alerting mechanism; The green LED is maintained in the ON state as long as the network traffic is normal. Once a malicious network activity is detected, the red LED is switched on and the buzzer is triggered.

The hardware architecture for this component is demonstrated in Figure 3.6, while Figure 3.7 showcases its physical implementation.

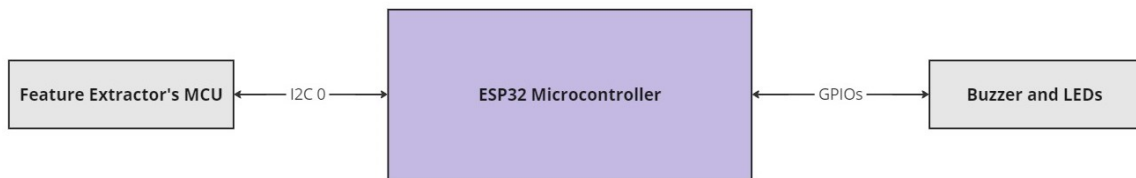


Figure 3.6: Model Inference Component Hardware Architecture

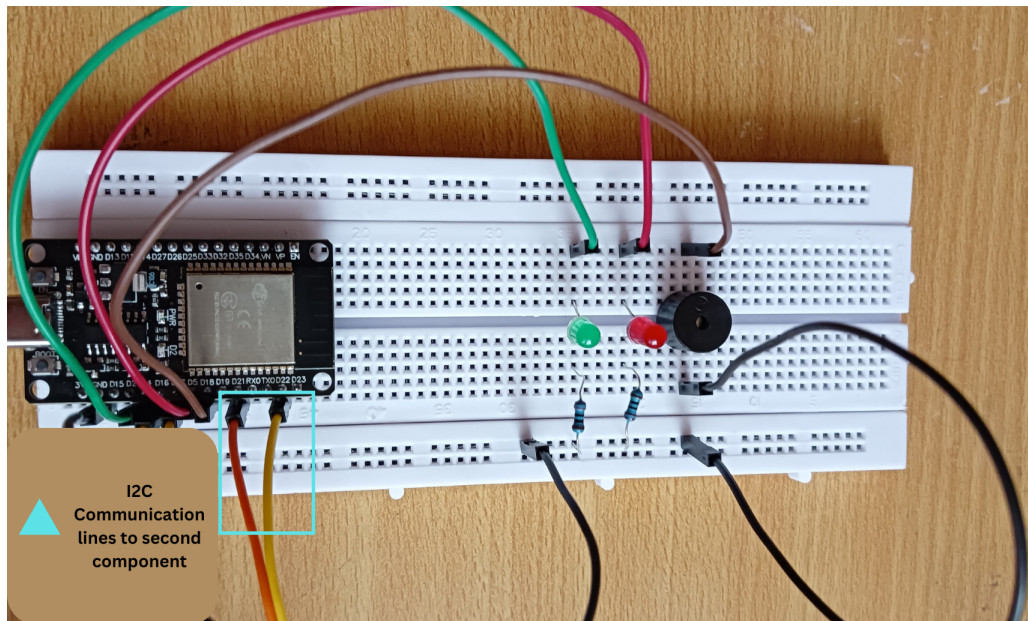


Figure 3.7: Model Inference Component Hardware Implementation

3.2 Raspberry Pi based Prototype

Our Raspberry Pi-based NIDS is designed to capture and analyze Ethernet network packets all on one board. The Raspberry Pi board is equipped with an Ethernet port that is connected, in our system, to a mirror port of a network switch. Additionally, the board interfaces a buzzer and two LEDs, by which the alerting mechanism is established. The hardware architecture for this system is demonstrated in Figure 3.8, while Figure 3.9 showcases its physical implementation.

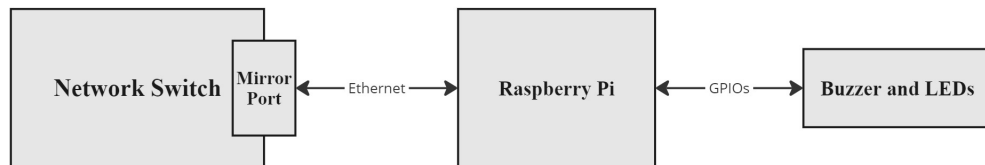


Figure 3.8: System Architecture Design

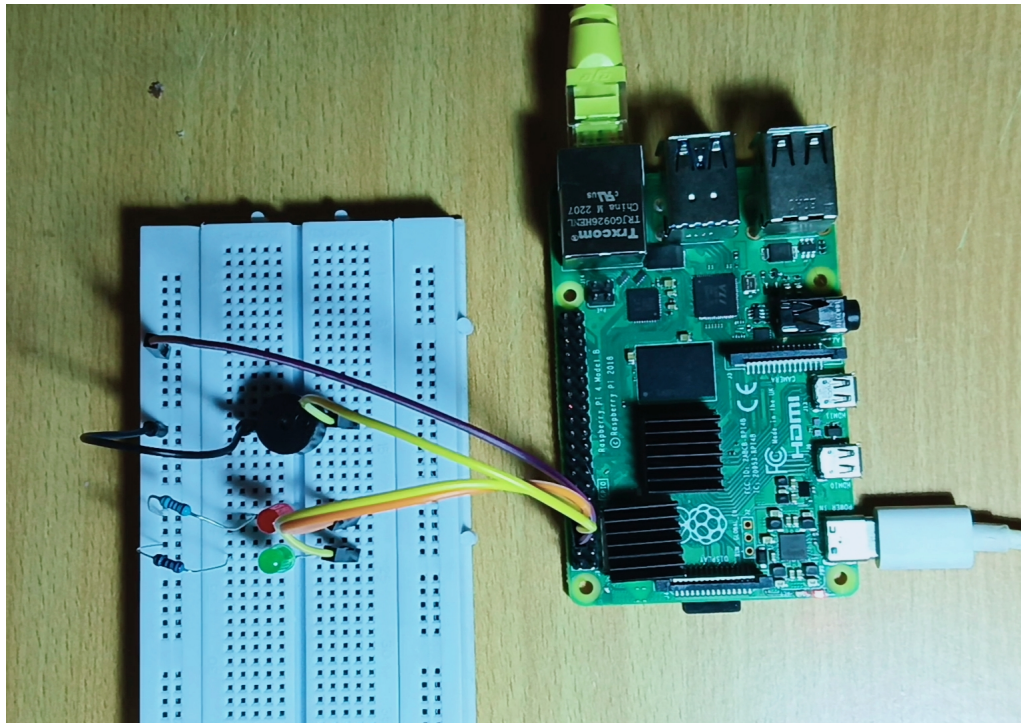


Figure 3.9: System Hardware Implementation

Chapter 4

Software System Design

4.1 Firmware Design and Implementation

In order for the proposed ESP32 prototype to be functional, firmware was developed for its three components. Each component's firmware manages the communication peripherals and manipulates data to ensure the operation of the system. All component firmware is built upon the ESP-IDF Framework, which itself is based on a modified version of the Free Real Time Operating System (FreeRTOS). The FreeRTOS provides the firmware with the foundation for multitasking, synchronization and resource management, while the ESP-IDF provides hardware abstraction, ready-to-use network stack as well as various low level drivers.

4.1.1 Creating and Transmitting the PCAP files

The first component in the proposed system deals with the Ethernet frames that are received from a mirror port of a switch. It sniffs the frames by using its Ethernet interface in promiscuous mode and creates PCAP files out of a selected number of them. In addition, the PCAP files are transmitted to the next component for further processing.

The firmware for this component relies on the pcap (file writer) ESP-IDF component¹, which allows for the tracing of captured packets in .pcap format².

As depicted in Figure 4.1, the component starts working by executing a series of initializations of peripheral drivers, event handlers as well as 2 FreeRTOS tasks, which are independent threads of execution that represent the core part of this component implementation.

¹The libraries that are part of the ESP-IDF are referred to as "components" by the ESP-IDF.

²The used component can be found at: <https://components.espressif.com/components/espressif/pcap>

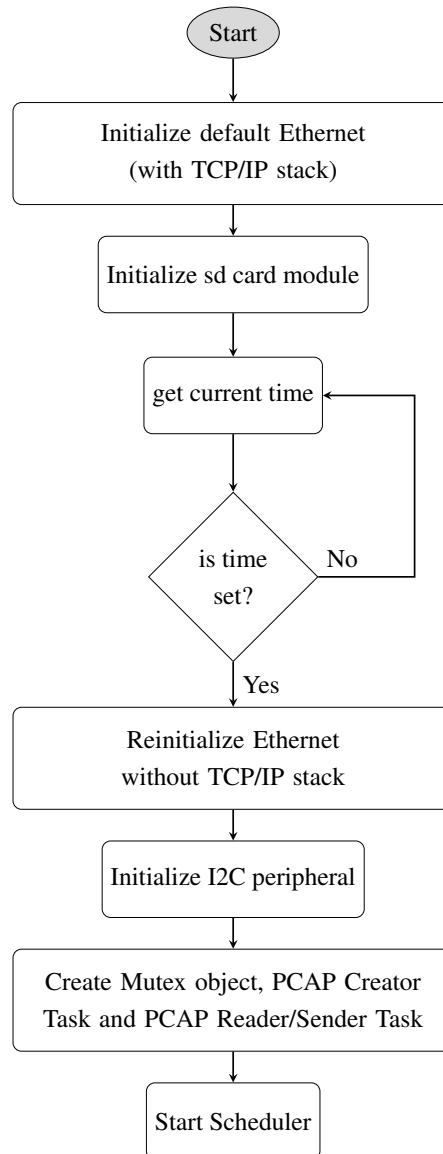


Figure 4.1: PCAP Generator Initialization Flowchart

4.1.1.1 Default Ethernet Initialization

This section of the initialization is responsible for configuring and initializing the following interfaces:

- The SPI interface for the ENC28J60 Ethernet module
- The light-weight IP (lwIP) TCP/IP stack by means of the ESP Network Interface (ESP-NETIF), which is an abstraction layer built on top of the lwIP stack.

Additionally, the Ethernet driver is installed and attached to the created network interface ESP-NETIF, and two event handlers are registered:

- Ethernet Event Handler: is used for logging useful information about Ethernet events such as: starting and stopping events, and connecting and disconnecting events.
- "got_ip" Event Handler: is called when the ESP32 successfully connects to the internet and is assigned an IP address. It initializes a Simple Network Time Protocol (SNTP) client and uses it to retrieve the current time from one of the registered NTP servers.

4.1.1.2 SNTP Client

The ESP32 connects to a Network Time Protocol (NTP) server to get the current time. This process is needed in order to fill in the timestamp fields in each Packet Record Header of the PCAP file, as demonstrated in Figure 4.2. The first Timestamp field represents the number of seconds that have elapsed since 1970-01-01 00:00:00 (Unix Seconds) when the packet was captured. The second Timestamp field represents the number of microseconds or nanoseconds that have elapsed since the last full second.

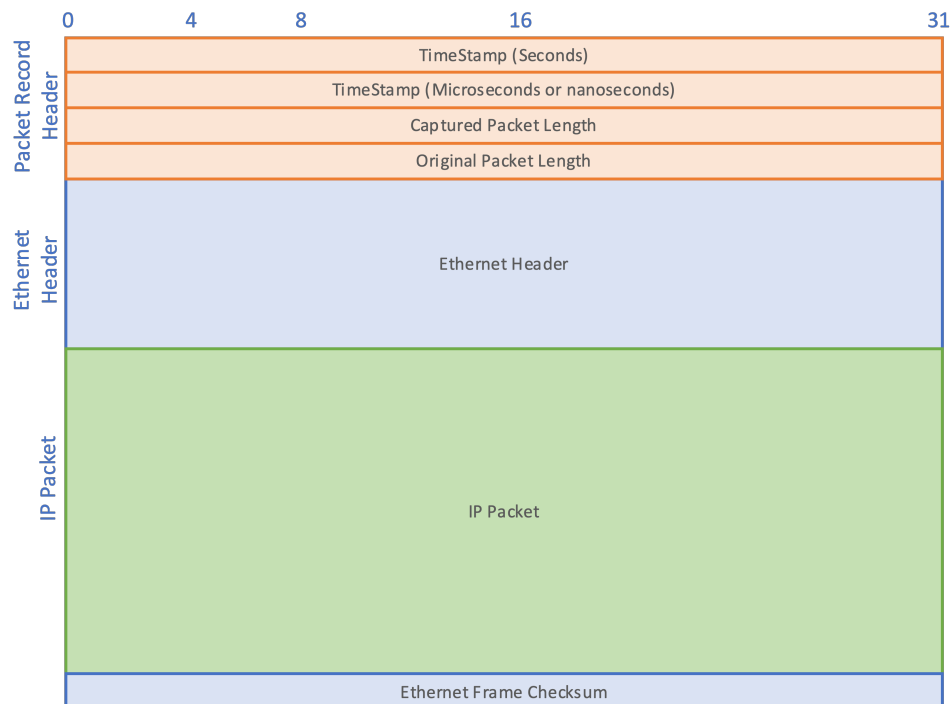


Figure 4.2: PCAP File Packet Record Fields [36]

4.1.1.3 Custom Ethernet Initialization

As the system deals with raw Ethernet packets, and since the ESP32's Real Time Clock (RTC) is now synchronized with the current time using the SNTP protocol, the TCP/IP stack can no longer be attached to the Ethernet driver. This is due to the fact that passing sniffed Ethernet frames to a custom input function, implies they will not be passed to the higher layers of the TCP/IP stack. This conflict can be resolved by re-initializing the Ethernet driver in promiscuous mode without the TCP/IP stack, and with a custom input path for the sniffed Ethernet frames. The custom input callback function is called whenever a new frame arrives, for which a structure containing its bytes pointer and length is pushed to a queue. The queue is later used by the PCAP Creator task.

4.1.1.4 Tasks Synchronization and SD Card Access

As later explained in Sections 4.1.1.5 and 4.1.1.6, the two FreeRTOS tasks share an SD card for manipulating PCAP files. To ensure synchronization when accessing the SD card between the two tasks, a Mutex is used to lock and unlock access to the SD card. This will prevent simultaneous access to the SD card, which could lead to data corruption or other issues. A Mutex is a synchronization primitive that is used in FreeRTOS [38] for inter-task communication and synchronization. It protects critical sections of code and shared resources from concurrent access.

4.1.1.5 PCAP Creator Task

The PCAP Creator task continuously checks the queue for new data. When a new packet arrives and is pushed to the queue, the task fetches the packet from the queue, and adds it to the PCAP file that is currently being created. The task keeps count of the number of packets that are recorded in the opened PCAP file, and uses that number to decide whether to close the current file or create a new one. Creating a PCAP file involves the creation of a new file in storage space. In this file, a global header, as seen in Figure 4.3, is written. The global header is then followed by a packet record for each captured packet. The structure of a PCAP file looks as follows:

- Global Header of the PCAP file
- First Packet Header
- First Packet Data
- Second Packet Header

- Second Packet Data
- ...

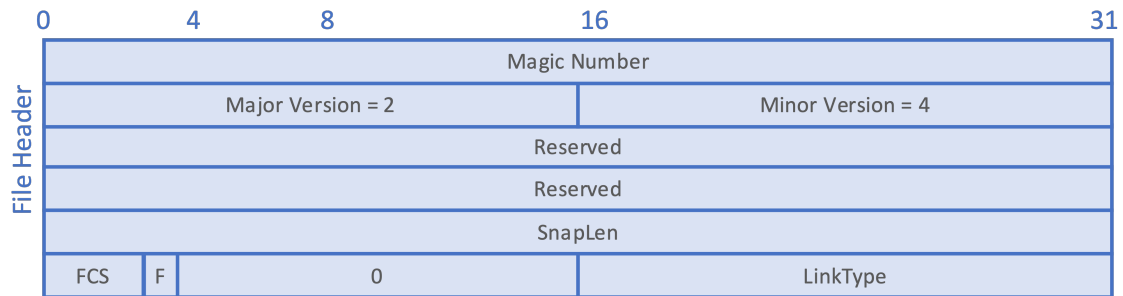


Figure 4.3: PCAP File Global Header Fields [36]

As seen above, the file header contains the following fields:

1. **Magic Number:** is used to detect the file format itself and the endianness of the data. The writing device writes the value 0xA1B2C3D4 into this field. Depending on the endianness, the reading device will read either 0xA1B2C3D4 or 0xD4C3B2A1. The read value will indicate whether the writing device is little-endian or big-endian.
2. **SnapLen (Snapshot Length):** is the maximum number of bytes that are recorded from the captured packet.
3. **LinkType:** is a value that describes the type of link from which the packets were captured. For example, a value of 1 = IEEE 802.3 Ethernet.
4. **FCS and F:** are fields for error detection.

A packet is recorded inside a PCAP file by writing its record header followed by a copy of the first *N* bytes of the Ethernet packet, *N* being the value of the *snaplen* field in the PCAP global header.

The flowchart in Figure 4.4 illustrates the PCAP Creator task working process.

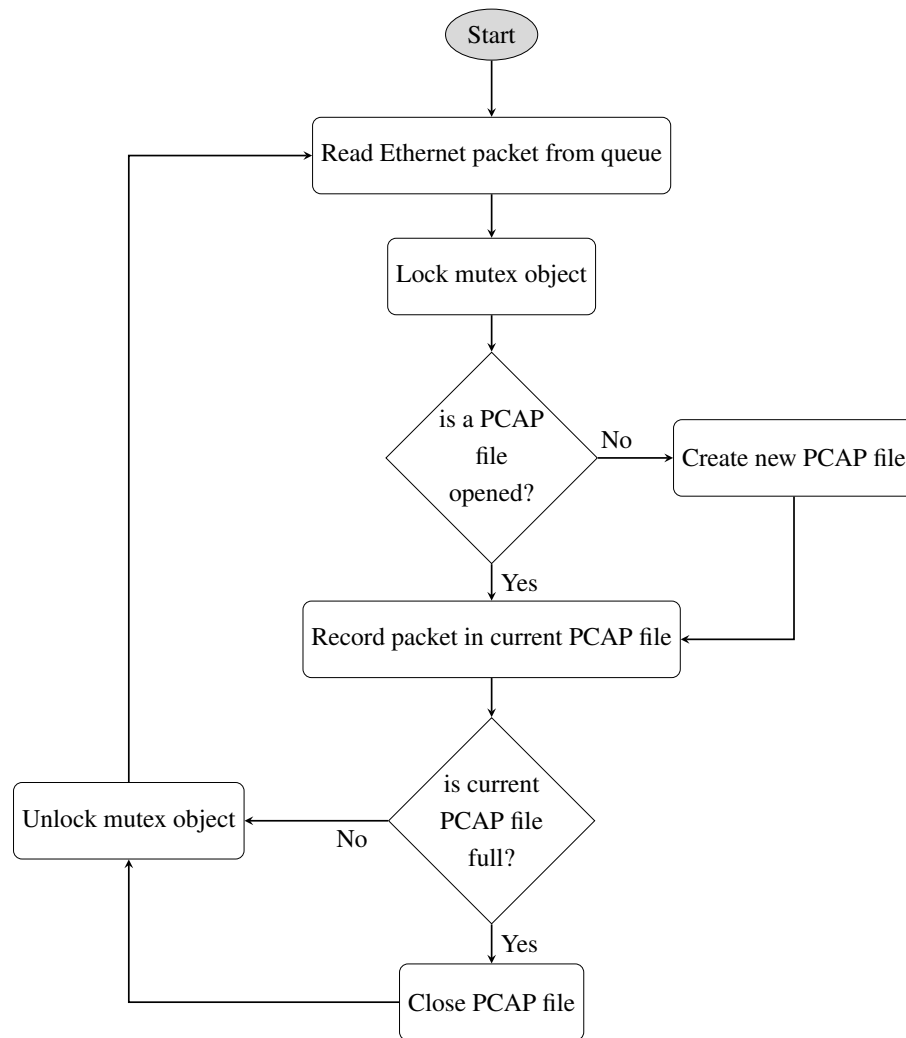


Figure 4.4: Flowchart of the PCAP Creator Task

4.1.1.6 PCAP Reader/Sender Task

The PCAP Reader/Sender task continuously checks for completed files that are ready to be sent. The complete file content is read by the task and sent to the Feature Extractor component using the I2C protocol. The PCAP file is then deleted after the transmission process. The flowchart for this task is demonstrated in Figure 4.7.

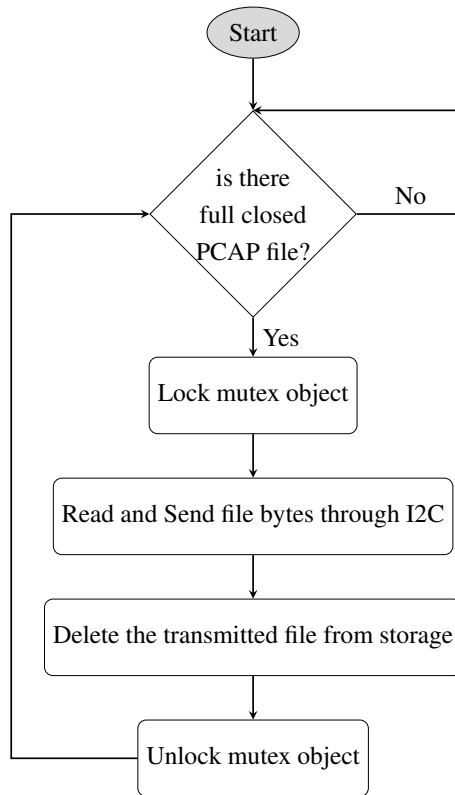


Figure 4.5: Flowchart of the PCAP Reader/Sender Task

4.1.2 Extracting Features from PCAP Files

The second component in the system extracts useful information about the PCAP files that are created from the network traffic. The PCAP files are received from the first component using I2C communication, and are saved in an SD card. Information about these files, termed ‘features’, is then transmitted to the Inference component to be used as input to the ML model. The names and descriptions of extracted features are shown in Table 4.1.

The firmware for this extraction process utilizes a modified version of the PcapPlusPlus library³, a C++ library designed for the capturing, parsing, and crafting of network packets. To reduce size and complexity of the application binary, and to ensure compatibility with ESP32 targets, the source code responsible for capturing and crafting packets, which relies on Linux and Windows network APIs, has not been compiled into the used version of the library. Hence, the modified version is only capable of parsing packets, which is sufficient for our project requirements.

The component starts by configuring and initializing the two I2C controllers, and the SPI interface for the SD card module. The main function creates two FreeRTOS tasks, and

³The library documentation can be found at: <https://pcapplusplus.github.io>

a Mutex object to synchronize access to the SD card. The tasks functionalities are discussed in sections 4.1.2.1 and 4.1.2.2, and the component initialization flowchart is shown below.

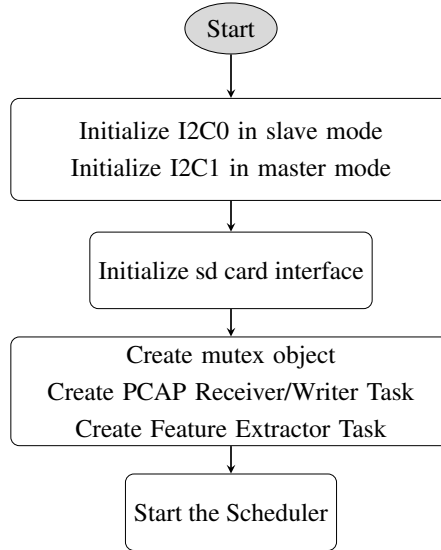


Figure 4.6: Component Initialization Flowchart

4.1.2.1 PCAP Receiver/Writer Task

The PCAP Receiver/Writer task receives file data bytes from the first component and writes these data into the sd card. The flowchart in Figure 4.7 demonstrates how it works.

4.1.2.2 Feature Extractor Task

The Feature Extractor task processes PCAP files that are received and stored in the sd card. The features that are extracted can be grouped, based on their nature and the kind of information they provide, into the following types:

- **Flow Characteristics:** these features describe the behavior of the network flow, which in our case is represented by the packets contained in the Pcap file. They include: Flow packets, Flow bytes, Number of packets in the flow, Rate, Srate, Drate.
- **Protocol Information:** these features provide information about the protocol types that are involved in the network traffic. They include: Protocol type, Transport Layer Protocols, Application Layer Protocols and other lower layers protocols.
- **Flag Counts:** these features record count of the TCP flags. They include: All TCP flag values and ACK, SYN, FIN, URG, and RST counts.

- **Dynamic Features:** these are complex features which provide insights into the nature of network traffic. They include: Magnitue, Radius, Covariance, Variance Ratio, and Weight.
- **Packets Statistics:** these include Total Sum, Minimum, Maximum, Average, and Standard Deviation of Packets.

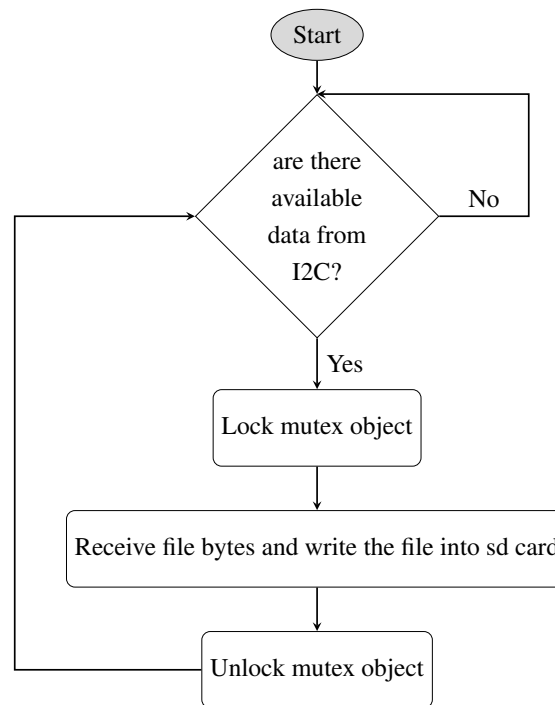


Figure 4.7: Flowchart of the PCAP Receiver/Writer Task

Metadata features, such as Protocol type and TCP flags, are extracted from the Pcap file using functions and classes provided by the library. Flow characteristics, Dynamic features, and Packets Statistics are calculated and updated iteratively during the file processing. At each iteration, a packet from the file is accessed, parsed and information about it is collected. These calculations require data objects that hold the necessary information about the packets. The mainly used data objects are the following:

- `incoming_packets` and `outgoing_packets` arrays: these arrays are used to store the sizes of incoming and outgoing Ethernet frames, respectively.
- `ethernet_sizes` array: this array is used to store the sizes of the Ethernet frames contained in the Pcap file.

- **TCPFlow and UDPFlow Classes:** these classes are used to keep track of packets that are part of a particular flow from a source pair (source IP address, source port) to a destination pair (destination IP address, destination port).

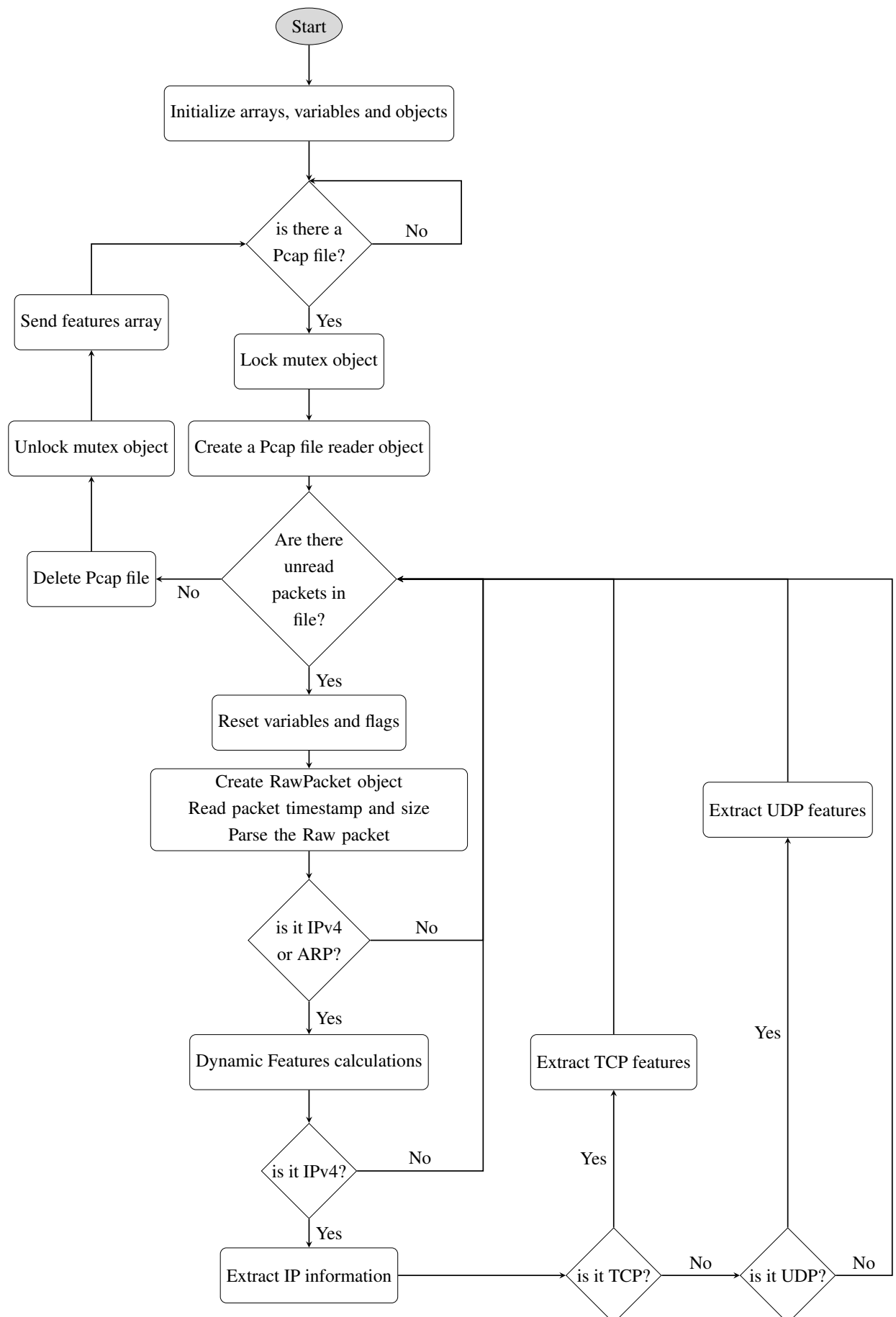
The task handles Pcap files and packets data using classes from the PcapPlusPlus library. A ‘Pcap File Reader Device’ object is created to open the Pcap file in read-mode. It is used to read all packets contained in the file. A ‘Raw Packet’ object holds the packet raw data. This data is not parsed and is kept in a byte array. In addition, this class provides methods for obtaining the timestamp and frame length of the Ethernet packet. The ‘Packet’ class represents a parsed packet. A parsed packet is a RawPacket instance with information about the protocol layers that are contained within the packet. It is implemented using a linked list of ‘Layer’ objects, where each layer points to the next higher layer that is contained in the packet. Flowchart of this task is shown in Figure 4.8.

As seen in the flowchart, features are extracted and calculated depending on the type of the packet being processed. Dynamic features are calculated from ARP and IPv4 packets, whereas flow features are extracted from TCP and UDP network flows.

Dynamic Features Calculation: Dynamic features are calculated using the incoming_packets and outgoing_packets arrays. These arrays are updated on each packet processing. Arrays of structures, that hold an IP address and its count of occurrences, are used to track IP addresses of packets. A packet’s destination IP address is checked against recorded source addresses, this checks whether this IP’s device has sourced some packets before. If it exists in the records, the frame size is added to the incoming_packets array. Otherwise, its occurrence is recorded, and the frame size is also added. Similar process is repeated for the source address of the packet.

IP Information: Information such as: Time to Live (TTL) and Protocol Type are extracted from the IP packets.

Extracting TCP and UDP Features: TCP and UDP features are flow characteristics of the TCP flows and UDP flows, respectively. A flow object defines a unique communication path between two endpoints using a pair of source and destination IP addresses and ports. TCPFlow and UDPFlow classes keep track of the existing flows within traffic, by saving

**Figure 4.8:** Flowchart of Feature Extractor Task

these flows along with the packets' lengths and timestamps that belong to them. Flow records are updated with a new flow, or new packet information if the flow already exists. Then, the total number of packets, flow byte count, flow duration, and directional (from source to destination and from destination to source) packet counts are calculated for each flow. Based on these features, Rate, Srate and Drate are also calculated.

4.1.3 Model Inference

The third component in the proposed system handles the inference, by predicting the category of the incoming traffic data. This component checks for available data in the I2C bus, then passes this data to the Predict() function which returns an array of predictions. A detailed explanation and flowchart of this function can be found in section 4.3. This function is converted to a C language format to be used with the ESP32, and is compiled with the main program. In order to manage communication and inference in parallel, two FreeRTOS tasks are created as follows:

- I2C Communication handler Task: this task manages communication over the I2C bus, by waiting for availability of data for inference, then sending it to the Prediction task via a queue.
- Prediction Task: this task is responsible for executing the prediction process, and handling the corresponding output.

The structure of the esp-idf project for this component is shown below:

```
model_inference/
├── main/
│   ├── main.c
│   ├── predict.c
│   └── predict.h
```

The combination of the self-contained and fast executable predict function, with the FreeRTOS multitasking ensures good overall performance and prediction.

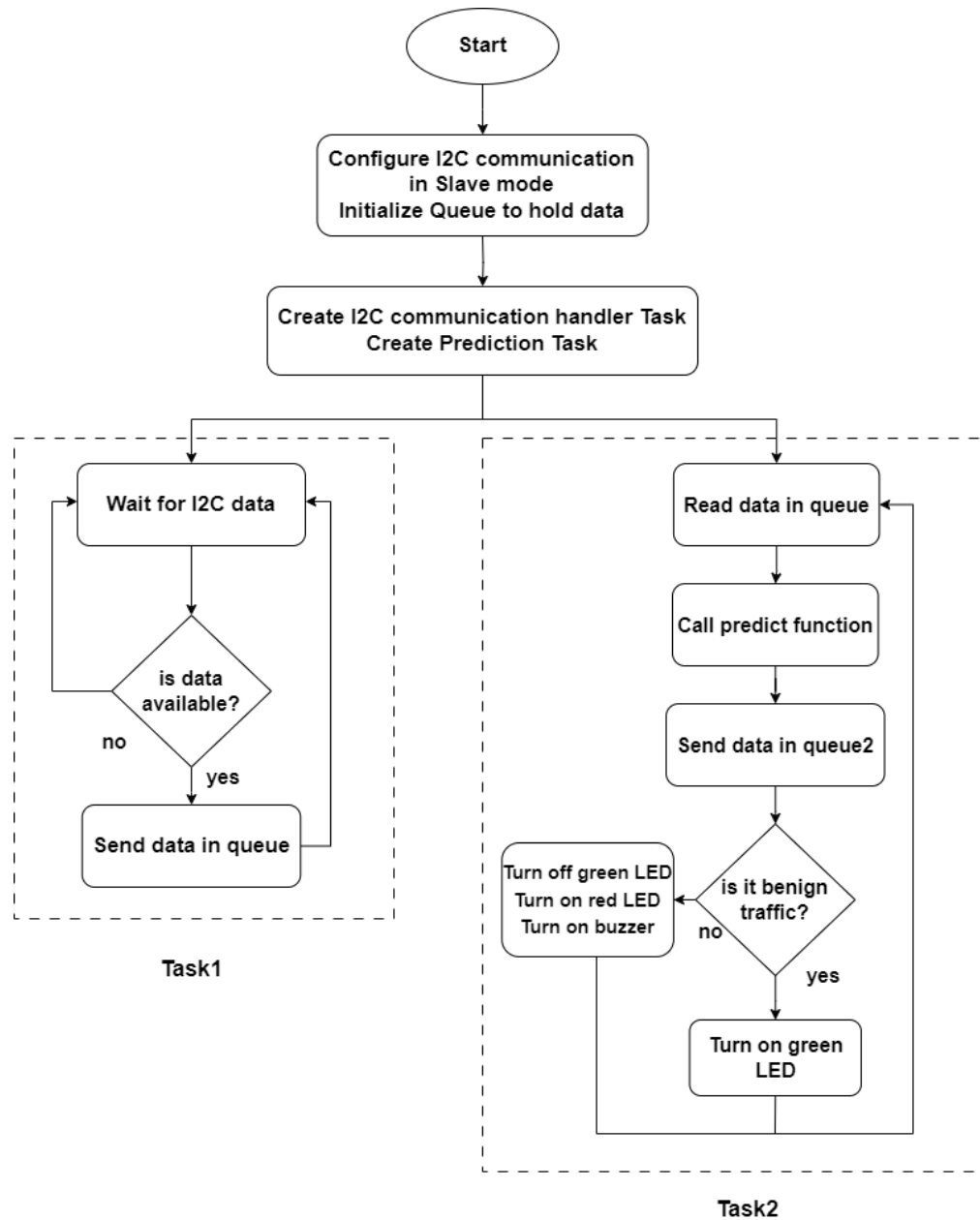


Figure 4.9: Model Inference Component Flowchart

4.2 ML Model Development

The development of the machine learning model involves selecting an appropriate dataset, preprocessing the data to align with machine learning requirements, and finally selecting the suitable algorithms and building the models.

4.2.1 Dataset Selection and Preprocessing

NIDS datasets are numerous due to the extensive research efforts in this area, with each dataset comprising distinct features and labels, and each having its own specific target.

4.2.1.1 NIDS datasets

An intrusion detection dataset can be developed by collecting information from network traffic flows. This information is required to study the attack patterns and abnormal activity of various network attacks. After collecting the incoming and the outgoing network traffic, network flow analysis is performed to study the network traffic. Flow analysis can be described as the process of analyzing the network packet information such as source IP address, destination IP address, source port number, destination port number, type of network services, etc.. [39]

The behavioral patterns of network attacks change gradually. For this reason, it is required to upgrade the used datasets in the dynamic environment. The following are some popular datasets used in this domain:

- **NSL-KDD:** The NSL-KDD dataset is a refined version of the KDD'99 dataset. Attacks are classified into four categories: Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R).
- **UNSW-NB15:** It includes nine types of attacks and normal network activities: Fuzzers, Analysis, Backdoors, DoS (Denial of Service), Exploits, Generic, Reconnaissance, Shellcode, and Worms. It contains 49 features extracted from network traffic.
- **IoT-23:** This dataset aims to reflect real-world IoT deployments and usage scenarios, including typical IoT communication protocols, such as MQTT, HTTP, and others. It detects two classes: benign traffic and malicious traffic. It is used for binary classification.
- **Bot-IoT:** The classes in this dataset include a combination of normal and botnet traffic, such as DDoS, DoS, OS and Service Scan, Keylogging, with the DDoS and DoS attacks further organized, based on the protocol used [40].
- **CIC-IDS-2017:** This dataset, provided by the Canadian Institute for Cybersecurity (CIC), consists of labeled network flows, including full packet payloads in pcap format. The dataset is publicly available and includes CSV files for machine and deep learning applications. It contains both benign and the most common attacks.

- **CIC-IoT-2023:** It reflects the current landscape of IoT devices and their associated network traffic. The dataset includes 33 attacks that are classified into seven categories: DDoS, DoS, Recon, Web-based, Brute Force, Spoofing, and Mirai, in addition to Benign Traffic.

In this project, the CIC-IoT-2023 dataset was selected due to its rich set of features extracted from the network traffic, the types of attacks and threats to which IoT devices are mostly exposed to, and because it is the newest dataset that reflects the current landscape of IoT traffic.

4.2.1.2 CIC-IoT-2023 Dataset

The CIC IoT 2023 dataset was created to help develop security software for IoT environments. It includes 33 types of attacks conducted on 105 IoT devices, classified into seven categories: DDoS, DoS, Reconnaissance, Web-based, Brute Force, Spoofing, and Mirai, in addition to Benign Traffic. This dataset includes multiple attacks that are not present in other IoT datasets.

DDoS attacks include many strategies, such as flooding attacks like UDP and ICMP Floods, and fragmentation-based attacks. These attacks cause service disruptions by overwhelming a single source with traffic. Web-based attacks, including SQL Injection and XSS, target web applications. Brute force attacks repeatedly try to gain unauthorized access. Spoofing attacks involve faking identities or altering network traffic. Mirai attacks use methods like GRE IP Flood and UDP Plain attacks, mostly targeting IoT devices [40].

Table 4.1: List of Features in the CIC IoT 2023 Dataset

Feature	Description
Destination IP Address	The IP address of the destination device.
Destination Port	The port number of the destination device.
Source IP Address	The IP address of the source device.
Source Port	The port number of the source device.
Protocol Type	UDP, TCP, IGMP, ICMP..
Flag	Flag values (e.g., FIN, SYN).
ACK count	Number of packets with ACK flag set in the same flow
SYN count	Number of packets with SYN flag set in the same flow
FIN count	Number of packets with FIN flag set in the same flow
URG count	Number of packets with URG flag set in the same flow

Continued on next page

Table 4.1 – continued from previous page

Feature	Description
RST count	Number of packets with RST flag set in the same flow
Transport Layer Protocol	Transport Layer Protocol (TCP or UDP)
Application Layer Protocol	Application layer (e.g., HTTP, DNS, Telnet)
Rate	Rate of packet transmission in a flow
Srate	Rate of outbound packets transmission in a flow
Drate	Rate of inbound packets transmission in a flow
Tot sum	Summation of packets lengths in flow
Min	Minimum packet length in the flow
Max	Maximum packet length in the flow
AVG	Average packet length in the flow
Std	Standard deviation of packet length in the flow
Tot size	Packet's length
Number	The number of packets in the flow
Flow Bytes/s	The flow rate in bytes per second.
Flow Packets/s	The flow rate in packets per second.
Magnitue	The square root of the sum of Average of lengths of incoming packets in the flow and Average of lengths of outgoing packets in the flow
Radius	The square root of the sum of Variance of lengths of incoming packets in the flow and Variance of lengths of outgoing packets in the flow
Covariance	Covariance of the lengths of incoming and outgoing packets
Variance	Ratio of Variance of the lengths of incoming packets in the flow to Variance of the lengths of outgoing packets in the flow
Weight	Product of Number of incoming packets and Number of outgoing packets

Table 4.1 presents a detailed description of a set of features present in the dataset. The 'Timestamp' records the specific time each packet is captured. 'Flow Duration' shows how long a packet has been in transit. 'Protocol Type' classifies packets by their network protocols, including common ones like IP, UDP, and TCP. The dataset also includes indicators for application layer protocols such as 'HTTP', 'HTTPS', and 'DNS' which allows to identify specific application-level behaviors. Counts of flags like 'FIN', 'SYN', 'RST',

‘ACK’, and ‘URG’ provide information on specific packet-level interactions and potential anomalies. Statistical metrics like ‘Covariance’ and ‘Variance Ratio’ measure the variability in packet lengths, helping to understand the relationship between incoming and outgoing packet sizes. ‘Weight’ represents the total count of incoming and outgoing packets and offers a comprehensive view of traffic patterns.

Several features seen above are considered categorical data; ‘Protocol Type’ feature in the dataset is a categorical one that indicates the type of network protocol: TCP or UDP. To handle categorical features, one-hot encoding is employed. It is a technique used to convert categorical variables into a numerical format suitable for ML algorithms.

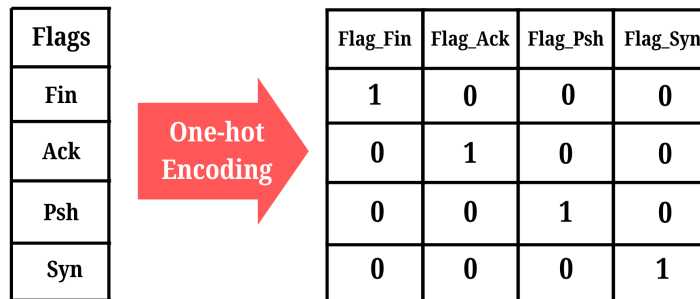


Figure 4.10: One-hot Encoding Example of Categorical Features

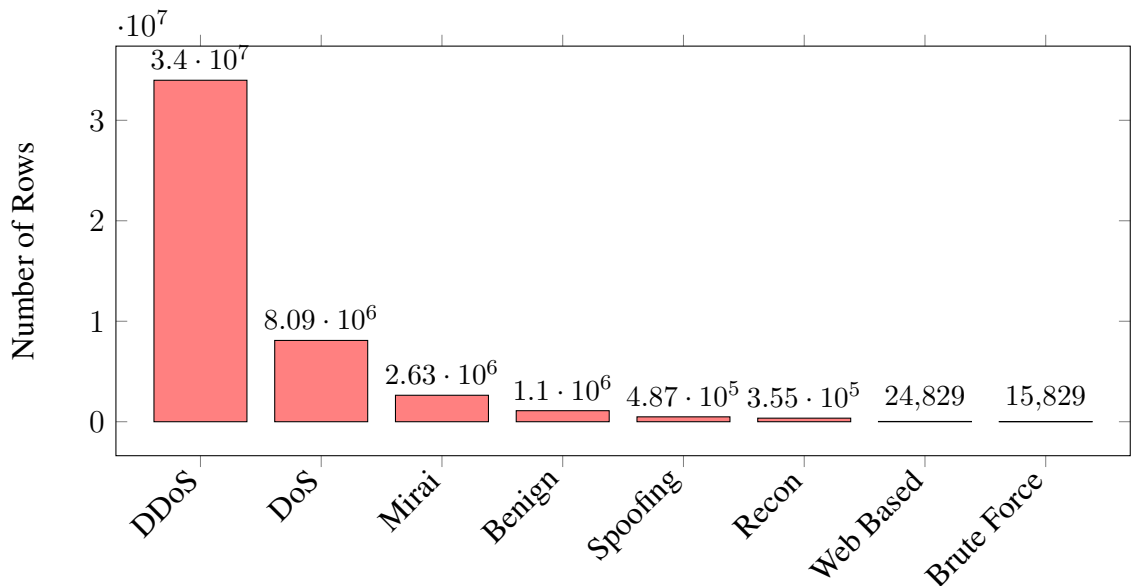


Figure 4.11: Distribution of Attack Categories in CIC IoT 2023 Dataset

Because the number of samples in this dataset exceeds 40,000,000, it can significantly increase the training, testing, and inference time when run on lightweight edge devices such as MCUs. Additionally, as shown in Figure 4.11, the dataset is unbalanced due to the large disparity in the number of rows for each attack type. Models trained on this dataset might become biased towards the majority classes, leading to poorer performance on the minority classes. To tackle this issue, we performed undersampling on majority classes, by reducing instances of DoS and DDoS categories. The final version of this dataset was divided into 70% for training and 30% for testing.

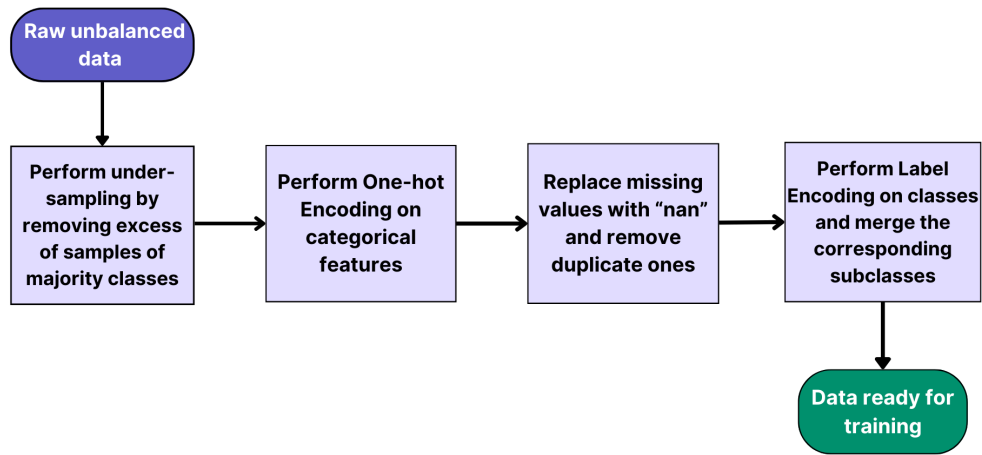


Figure 4.12: Dataset Pre-processing Steps

4.2.2 ML Model Building

Different algorithms were used to train different models. The corresponding results will be compared to decide which trained model to deploy on the edge, as shown in Chapter 5:

4.2.2.1 Naive Bayes

Prior probabilities of each class are calculated, along with the likelihoods or the probabilities of each feature for each class.

These results are saved for further calculations of posterior probabilities. Additionally, we use log-probabilities instead of raw probabilities to enhance numerical stability and avoid floating-point underflow.

Once the model is trained, we use the calculated priors and likelihoods to find the probabilities of a new instance and predict the correct class.

4.2.2.2 Decision Tree

The decision tree was initialized with a maximum depth of 50 nodes. The criterion chosen was ‘Gini Index’ and the splitter used was ‘best’. Once the model is trained, it is saved in a PKL file and loaded for further predictions. Traversing starts at the root of the decision tree and goes through its children nodes until a leaf node is reached. The corresponding class is then assigned as the predicted result.

4.2.2.3 Random Forest

The random forest model was initialized with 50 decision trees. By having a large number of estimators, the model can capture more complex patterns and reduce the variance.

4.2.2.4 Neural Network

This model consists of four layers: three hidden layers and one output layer as shown in Figure 4.13 (units stand for neurons).

Batch normalization was used to stabilize and accelerate the training process. The parameters were initialized in the following way: The gamma vector (γ) was set to 1, the beta vector (β) was set to 0, the moving mean was initialized to 0, and moving variance was initialized to 1. Initializing these parameters provide a stable starting point to avoid unnecessary shift. The activation functions used are ReLU for the hidden layers and Softmax for the output layer. ReLU (Rectified Linear Unit) is defined as:

$$\text{ReLU}(x) = \max(0, x) \quad (4.1)$$

The Softmax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (4.2)$$

The final output layer uses the Softmax activation function to convert the raw scores into probabilities for each class.

The Adam optimizer was used for training our ANN (Adaptive Moment Estimation), which is an extension of the Stochastic Gradient Descent algorithm. It computes adaptive learning rates for each parameter.

For multiclass classification, categorical cross-entropy loss function was set, which calculates the log loss for each class, weights it by the true class label, and sums these values across all classes. It is defined as:

$$\text{Loss} = - \sum_i y_i \log(p_i) \quad (4.3)$$

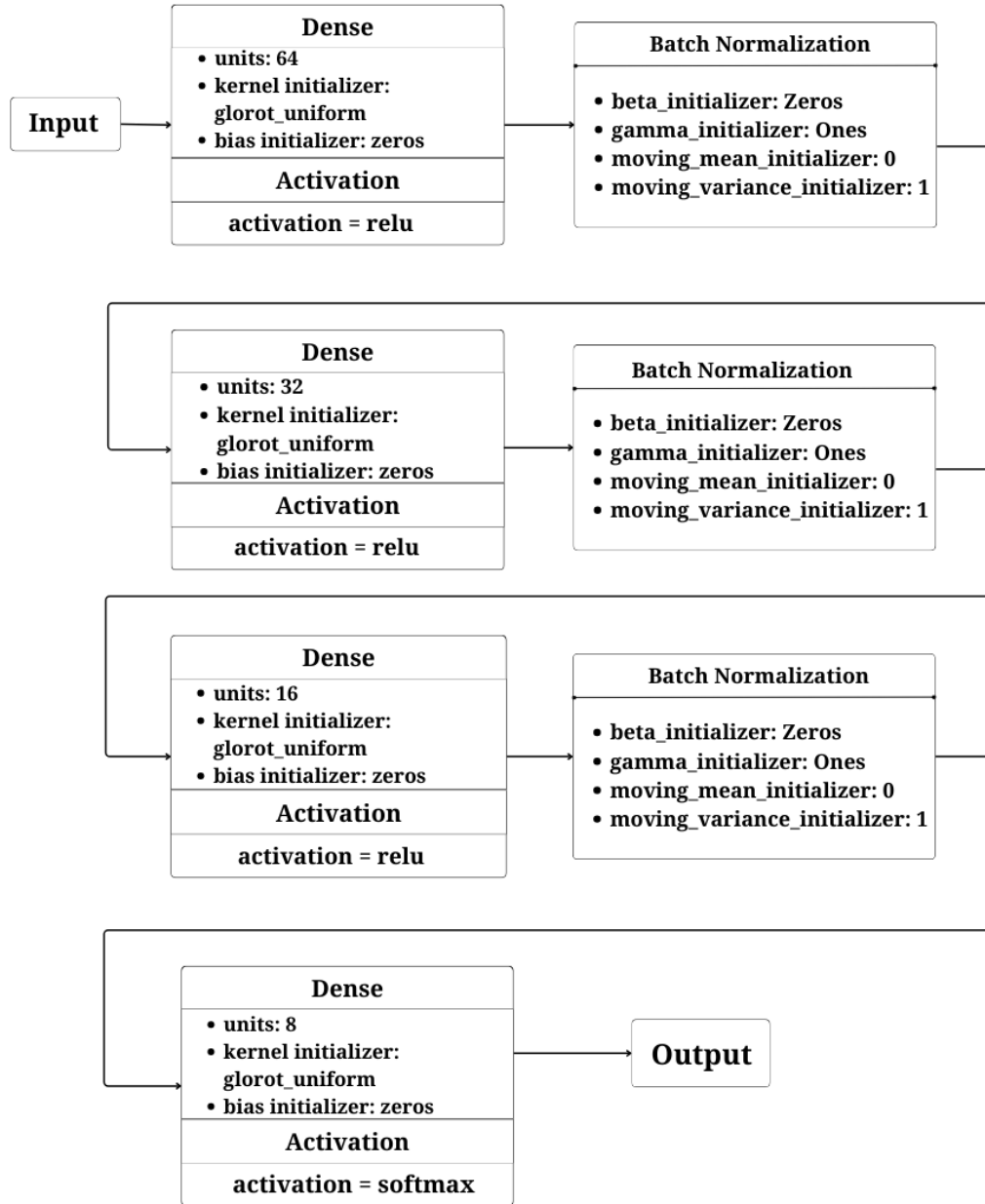


Figure 4.13: Model Layers Structure

4.3 Raspberry Pi Implementation

Among the models built above, the Decision Tree model was chosen, due to its high performance and relatively small size compared to other models. A detailed discussion is found in section 5.3.

The system is designed to sniff network packets, process them, and make predictions about their nature, distinguishing between benign traffic and potentially harmful activity.

Packets are captured using Tshark tool (Terminal oriented version of Wireshark for capturing and displaying packets) and stored in a PCAP file. Because raw data cannot be directly fed into the ML model, preprocessing is required to extract the necessary features which are used by the model to make predictions.

Preprocessing involves extracting relevant information from PCAP files, such as packet size, protocols used, and other network traffic characteristics that are mentioned in Table 4.1. These features are then formatted into an array that suits the input format of the Decision Tree model. Section 4.1.2 explains the process of extracting the necessary features.

Predicting the label of the received data is performed by a function of if-else statements, generated from the decision tree trained model. The flowchart in Figure 4.15 explains how this function is defined.

The advantage of this approach is the fast real-time prediction; direct execution of if-else statements is faster than model deserialization and avoids computational costs and memory overhead. Additionally, it is a self-contained code that is portable to different programming environments, as seen in section 4.1.3.

The program utilizes multi-threading to fulfill real-time requirements:

- Thread1: Sniffing Packets
- Thread2: Preprocessing
- Thread3: Predicting

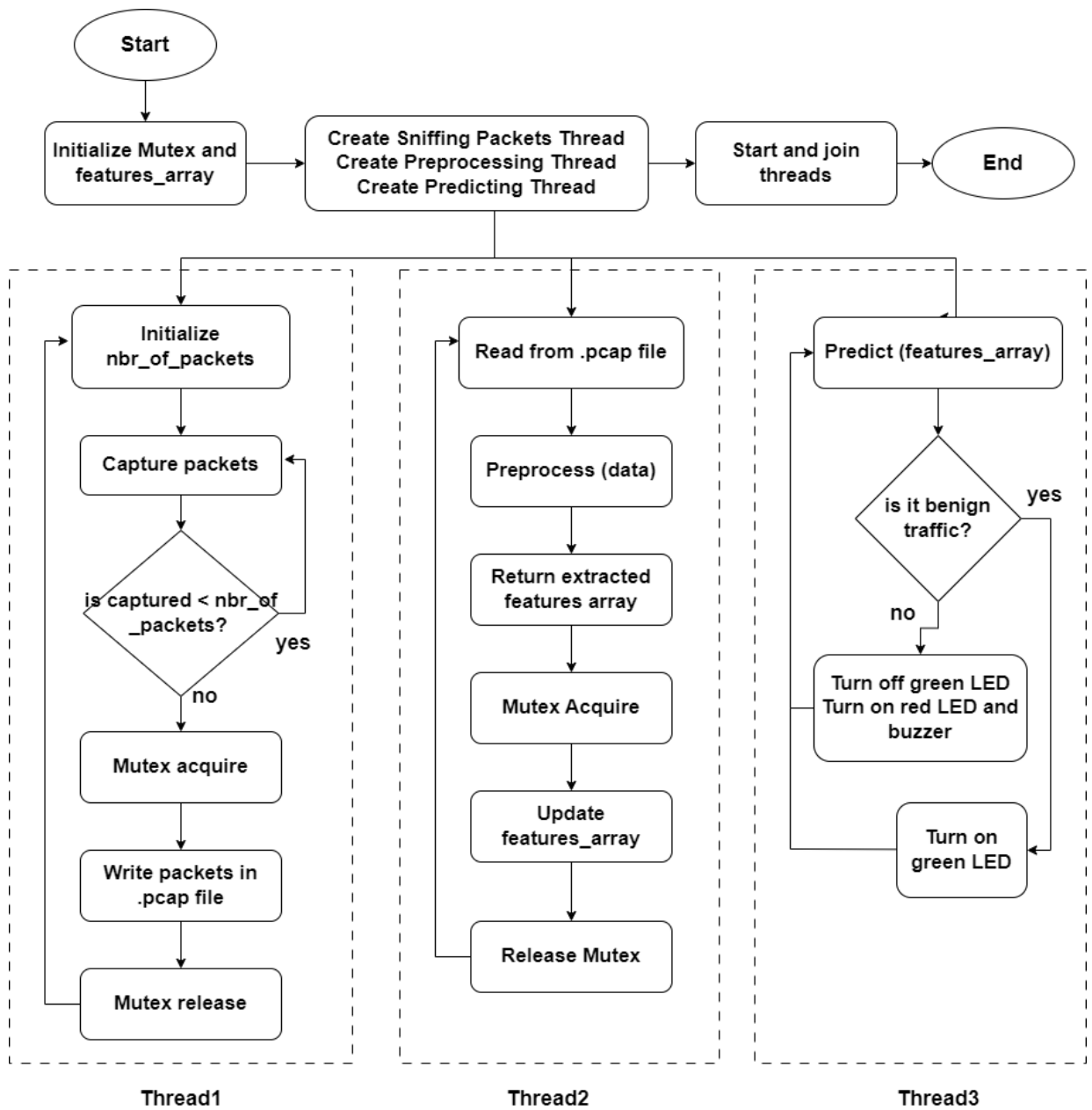


Figure 4.14: Implementation on Raspberry Pi Flowchart

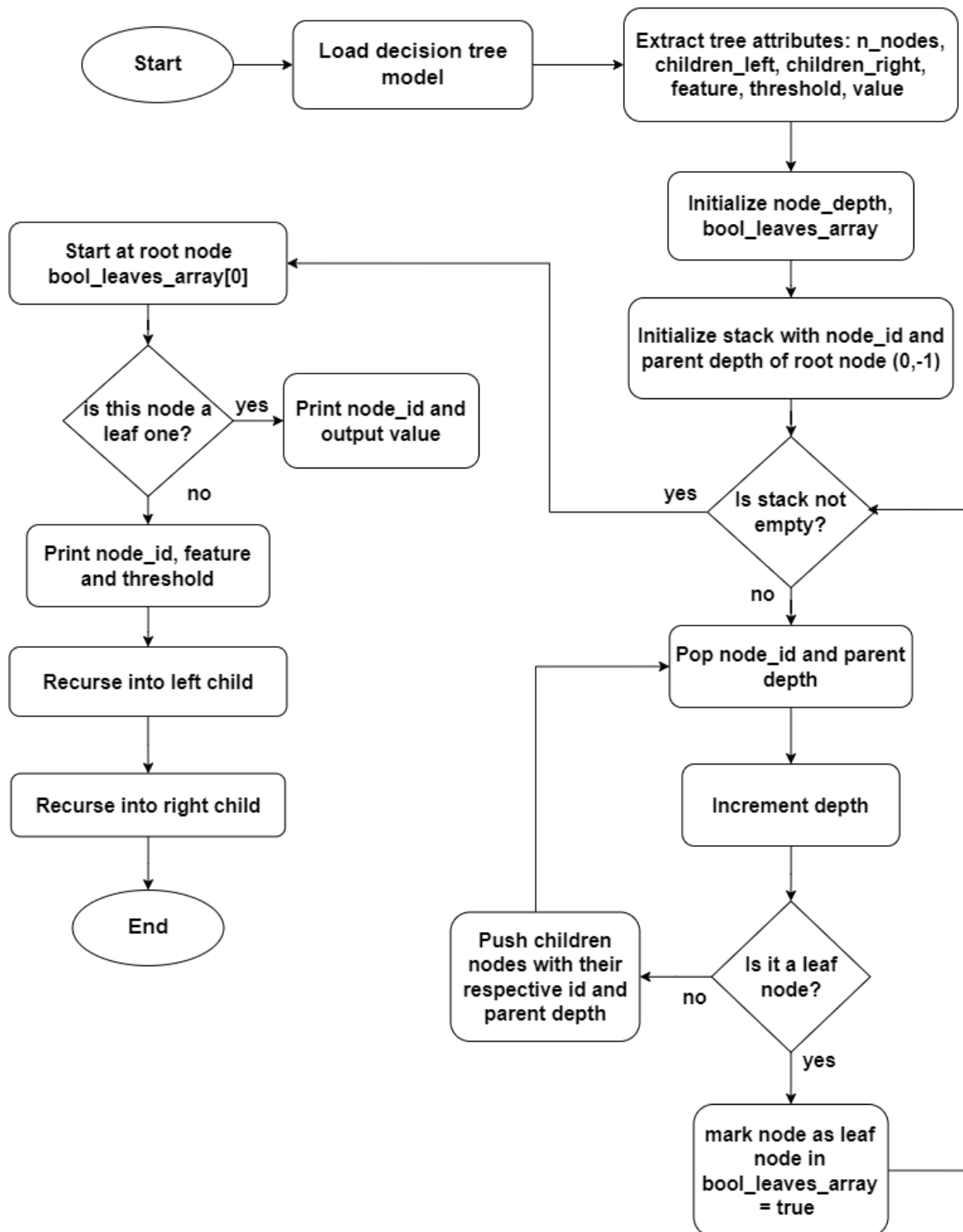


Figure 4.15: Generating the Predict() Function Flowchart

Chapter 5

Results and Analysis

In this project, four ML models were trained to detect network intrusions in IoT networks. The objective of this analysis is to evaluate the performance of these models, covering their strengths and weaknesses within the context of our application. Based on the insights gained from this analysis, we will select the most appropriate algorithm for this task to deploy on edge.

5.1 Hypothesis on the Performance of ML Models

Among the four ML models that were trained for a Network Intrusion Detection System, it is hypothesized that their overall performance, ranked from the least to the most effective, will follow this order: Naive Bayes, Neural Network, Decision Tree, then Random Forest.

This hypothesis relies on the idea that Random Forest will outperform other models due to its ensemble nature and robustness, followed by Decision Tree for its interpretability and efficiency. Neural Networks, while powerful, are hypothesized to rank third due to their resource demands. Naive Bayes is expected to have the lowest performance because of its strong independence assumptions, which may not represent the complex environment of IoT network traffic.

5.2 ML Models Evaluation

A detailed evaluation of each model's performance is illustrated using a detailed evaluation metrics to provide a clear comparison: accuracy, precision, recall, and F1-score.

5.2.1 Naive Bayes

The choice to explore Naive Bayes classifier was driven by the primary advantages of its simplicity and ease of implementation, based on straightforward probabilistic principles. This makes Naive Bayes suitable for real-time applications. It also performs effectively with small datasets. Unlike some more complex algorithms that require large amounts of data to achieve high performance, that is an advantage in the case of our dataset, which was under-sampled to overcome the imbalance.

Table 5.1: Confusion Matrix Results of Naive Bayes

Class	True Positives	False Positives	False Negatives	True Negatives (TN)
DDoS	0.97	1.063	0.032799	5.932663
DoS	0	0.000039	0.99888	6.999543
Benign Traffic	0.57	1.1305	0.43239	5.865572
Mirai	0.33	0.00829	0.67609	6.984082
Spoofing	0.097	0.04512	1.0001	6.856242
Reconnaissance	0.87	2.1137	0.13608	4.878682
Web Based	0	0	1.004	6.994462
Brute Force	0.048	0.16069	0.95	6.839772

Table 5.2: Performance Metrics for Naive Bayes

Class	Accuracy	Precision	Recall	F1 Score
DDoS	0.8707	0.4776	0.9672	0.6392
DoS	0.8750	0.0000	0.0000	0.0000
Benign Traffic	0.7489	0.3351	0.5685	0.4214
Mirai	0.9028	0.9755	0.3287	0.4902
Spoofing	0.8611	0.6824	0.0886	0.1575
Reconnaissance	0.7296	0.2918	0.8650	0.4372
Web Based	0.8742	0.0000	0.0000	0.0000
Brute Force	0.8722	0.2291	0.0482	0.0793
Weighted Average	0.8475	0.4039	0.7111	0.4897

Starting with accuracy as shown in Table 5.2, we observe varying values across different classes. The classifier achieves an accuracy of 0.8707 for the ‘DDoS’ class. However, the accuracy for other classes such as ‘DoS’, ‘Web Based’, and ‘Brute Force’ is notably lower.

Moving on to precision, it is high for classes like ‘Mirai’ and ‘Spoofing’. However, it is considerably lower for classes such as ‘DoS’ and ‘Web Based’, implying a higher rate of false positives in these predictions. The recall for the ‘DDoS’ class is high. However, the

recall for classes like ‘DoS’ and ‘Web Based’ is notably lower, indicating a higher rate of false negatives in these predictions.

5.2.2 Neural Network

Neural Networks were chosen among the models because of their capability to learn patterns from raw data, and discover underlying structures that may not be immediately apparent. Additionally, it requires less computational power compared to other deep learning algorithms such as Convolutional Neural Networks with a more complex architecture.

Table 5.3: Confusion Matrix Results of Neural Network

Class	True Positives	False Positives	False Negatives	True Negatives
DDoS	1	0.00006	1.9029	5.095502
DoS	0	0	1.001572	6.99689
Benign Traffic	0.99	3.620742	0.01491	3.37281
Mirai	0.2	0.16225	0.80064	6.835572
Spoofing	0.0017	0.00016	0.9937	7.0029
Reconnaissance	0.022	0.11313	0.981	6.882332
Web Based	0	0	1.001	6.997462
Brute Force	0	0	1.005	6.993462

Table 5.4: Performance Metrics of Neural Network

Class	Accuracy	Precision	Recall	F1 Score
DDoS	0.7269	0.9999	0.0005	0.0011
DoS	0.9997	0.0000	0.0000	0.0000
Benign Traffic	0.5069	0.2150	0.9851	0.3524
Mirai	0.8450	0.5528	0.1998	0.2930
Spoofing	0.7063	0.9054	0.0017	0.0034
Reconnaissance	0.6855	0.1620	0.0218	0.0387
Web Based	0.6998	0.0000	0.0000	0.0000
Brute Force	0.6998	0.0000	0.0000	0.0000
Weighted Average	0.7321	0.2753	0.1088	0.0708

For the ‘DDoS’ class, the high precision shown in Table 5.4 indicates that when the model predicts DDoS, it is usually correct, but the very low recall shows it misses most actual DDoS instances. In the ‘DoS’ class, the model fails to identify DoS instances. For

‘Benign Traffic’, The high recall means most benign traffic instances are correctly identified, but the low precision indicates many false positives. The ‘Mirai’ class’s results show that it identifies this category but there is room for improvement. For ‘Spoofing’, the classifier has a high precision but very low recall. The ‘Reconnaissance’ class’s results indicate poor performance in recognizing reconnaissance activities. Both the ‘Web Based’ and ‘Brute Force’ classes results suggest that the model fails to identify instances of these classes. Results indicate that the neural network classifier has limitations in effectively classifying the different types of traffic, with particularly low recall and F1 scores pointing to its struggle in correctly identifying positive instances.

5.2.3 Decision Tree

The choice of Decision Trees is due to its simplicity and high performance. They are also able to handle complex, non-linear relationships within data.

Table 5.5: Confusion Matrix Results of Decision Tree

Class	True Positives	False Positives	False Negatives	True Negatives
DDoS	1	0.00183	0.00029	6.996342
DoS	1	0.00016	0.00013	6.998172
Benign Traffic	0.99	0.28301	0.00632	6.719132
Mirai	1	0.00001	0	6.998452
Spoofing	0.94	0.0533	0.06	6.945162
Reconnaissance	0.91	0.0919	0.0852	6.911362
Web Based	0.9	0.0403	0.0977	6.960462
Brute Force	0.78	0.00412	0.225	6.989342

Table 5.6: Performance Metrics for Decision Tree

Class	Accuracy	Precision	Recall	F1 Score
DDoS	0.9999	0.9982	1.0000	0.9991
DoS	0.9999	0.9998	0.9999	0.9999
Benign Traffic	0.9967	0.7776	0.9937	0.8738
Mirai	0.9998	0.9999	1.0000	1.0000
Spoofing	0.9986	0.9460	0.9400	0.9430
Reconnaissance	0.9974	0.9085	0.9144	0.9114
Web Based	0.9985	0.9577	0.9021	0.9292
Brute Force	0.9935	0.9490	0.7759	0.8510
Weighted Average	0.9976	0.9517	0.9383	0.9425

For the ‘DDoS’ class, the Decision Tree achieves almost perfect performance as seen in Table 5.6. The classifier is highly accurate in identifying DDoS attacks with minimal errors. Similarly, the ‘DoS’ class also shows near-perfect results. For ‘Benign Traffic’, while the recall is very high, indicating that most benign traffic is correctly identified, the lower precision suggests a slightly higher rate of false positives. The ‘Mirai’ class exhibits perfect performance. The ‘Spoofing’ and ‘Web Based’ classes have strong performance as well. For the ‘Reconnaissance’ class, these values indicate good performance, however, some misclassifications still occur. For the ‘Brute Force’ class, the lower recall compared to precision indicates a higher rate of false negatives for this class.

5.2.4 Random Forest

Random forest is robust to noise and overfitting. By averaging the results of many decision trees, Random Forest reduces the risk of overfitting that can occur with a single decision tree.

Table 5.7: Confusion Matrix Results of Random Forest

Class	True Positives	False Positives	False Negatives	True Negatives
DDoS	1	0.0113	0.00009	6.9846
DoS	1	0.00016	0.000674	6.9976
Benign	1	0.3315	0.003278	6.6637
Mirai	1	0.000042	0.00292	6.9955
Spoofing	0.77	0.0271	0.2277	6.9736
Reconnaissance	0.92	0.1519	0.0762	6.8504
Web Based	0.9	0.0622	0.1026	6.9336
Brute Force	0.82	0.0017	0.175	7.0017

Table 5.8: Performance Metrics of Random Forest

Class	Accuracy	Precision	Recall	F1 Score
DDoS	0.9999	0.9884	0.9999	0.9941
DoS	0.9998	0.9998	0.9993	0.9996
Benign	0.9513	0.7514	0.9968	0.8559
Mirai	0.9999	0.9994	0.9971	0.9983
Spoofing	0.9912	0.9665	0.7713	0.8580
Reconnaissance	0.9817	0.8572	0.9232	0.8891
Web Based	0.9886	0.9354	0.8973	0.9160
Brute Force	0.9968	0.9972	0.8244	0.9034
Weighted Average	0.9910	0.9488	0.9240	0.9329

For the ‘DDoS’ class, the Random Forest achieves near-perfect performance. Similarly, for the ‘DoS’ class, the classifier shows excellent results. For the ‘Benign’ class, the recall is very high, but the lower precision suggests a higher rate of false positives. The ‘Mirai’ class also exhibits strong performance. For the ‘Spoofing’, ‘Reconnaissance’ and the ‘Web Based’ classes are well classified. For the ‘Brute Force’ class, the classifier shows a high precision, but the recall is slightly lower.

Results show that the Random Forest classifier performs very well across all classes. The results are similar to the ones obtained by Decision Tree.

5.3 ML Models Comparison

Beginning with Naive Bayes, it demonstrated moderate accuracy of 0.8475. However, its precision and recall scores varied across the different classes. It exhibited relatively high recall for DDoS attacks but struggled with precision, indicating a tendency to misclassify benign traffic as DDoS.

Moving to Neural Network, it resulted in a lower accuracy of 0.7321. Despite achieving high precision for certain classes like DDoS and Spoofing, there is a low recall across various categories. In addition, it was unable to effectively detect DoS attacks. These findings suggest that while the Neural Network may excel in certain areas, its overall effectiveness has some limits. As mentioned by Gaël Varoquaux, the research director at Inria and one of the creators of Scikit-learn, for the same amount of time spent on random search, tree-based models scores are always high above neural networks. [41]

In contrast, the Decision Tree model exhibited exceptional performance, with an accuracy of 0.9976. It demonstrated near-perfect precision and recall scores across most classes, indicating robustness in detecting different types of intrusions.

Finally, the Random Forest model also showcased good performance, with a weighted average accuracy of 0.9910. Similar to the Decision Tree, it achieved high precision and recall scores across various classes, demonstrating consistency in detecting diverse intrusion patterns.

Table 5.9: Average Accuracy of Each Model

Model	Accuracy	Precision	Recall	F1 Score
Naive Bayes	0.8475	0.4039	0.7111	0.4897
Neural Network	0.7321	0.2753	0.1088	0.0708
Decision Tree	0.9976	0.9517	0.9383	0.9425
Random Forest	0.9910	0.9488	0.9240	0.9329

In addition, categorical variables are often seen as a major problem for using Neural Networks on tabular data. Results obtained by G. Varoquaux, L. Grinsztajn and E. Oyallon show that when trained on numerical variables only, a narrower gap reveals between tree-based models and NNs, compared to when categorical variables are included [41].

It has also been shown in Enchun Shao's work that Decision Tree and Random Forest models performed better than Support Vector Machine in classifying many attack categories. [42]

Table 5.10: F1 Scores of ML Models Found by Enchun Shao

Model	Brute Force	Web-Based	DDoS	Port Scan
Decision Tree	> 80%	> 80%	> 85%	> 90%
Random Forest	> 90%	> 85%	> 85%	> 90%
SVM	< 80%	< 80%	< 60%	< 75%

These results are a proof that tree-based models are the most suitable for this kind of tasks.

5.3.0.1 Explaining the Bias of the Models

The Neural Network model is biased towards benign traffic, misclassifying certain types of attacks, such as reconnaissance, web-based attacks, and spoofing, due to the similar patterns to normal network traffic, making them harder to differentiate. However, 'DoS' and 'Mirai' attacks are better classified because they involve distinctive and anomalous patterns that deviate more from normal network behavior, which makes it easier for the model to learn and recognize. Mirai, for example, often uses GRE (Generic Routing Encapsulation) packets or large volumes of TCP SYN packets. Mirai's traffic can be identified by the presence of a large number of short-lived TCP connections, often with SYN packets without corresponding ACK packets [43]. On the other hand, 'Reconnaissance' attacks involve probing a network to gather information. The patterns are subtle because the volume of traffic might be low, and the behavior may resemble legitimate network scanning. Similar to 'Spoofing' attacks that involve falsifying the source IP address of packets, which may appear as a normal traffic.

This explains why the model tends to fail to classify these types of attacks, since the features used to represent them, such as packet count or byte count, might not reveal the nuanced behavior of a port scan or a spoofed packet. Thus, the model requires much more instances compared to other attacks like "DDoS".

In conclusion, based on our analysis and the F1-Score graph in Figure 5.1:

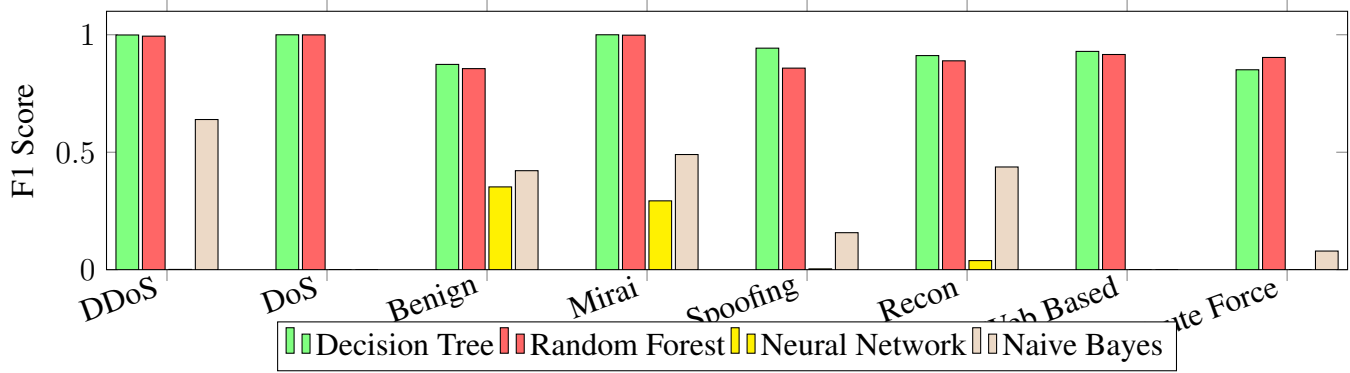


Figure 5.1: Comparison of F1 Scores for Different Models on Various Attacks

The models are ordered from least to most performant as follows: Neural Network, Naive Bayes, Random Forest, Decision Tree.

It has been shown that tree-based models are the most suitable for this task. Both Decision Tree and Random Forest performed excellently. Therefore, we have decided to select the Decision Tree model for further implementing on edge. This decision was made considering the model's performance and its suitability for deployment in resource-constrained environments. For less resource-constrained environments, Random Forest will be the right choice because of its robustness and good generalization.

5.4 Testing with External Pcap Files

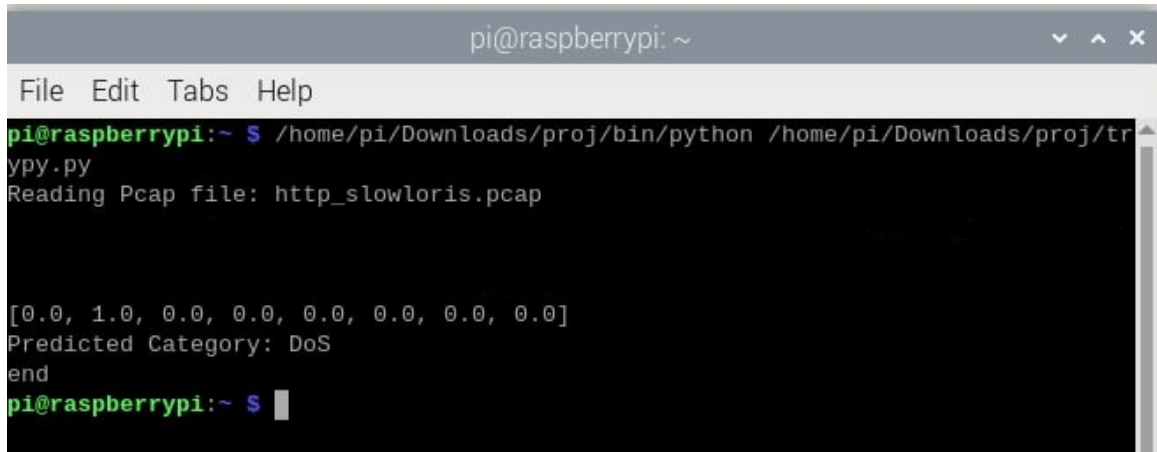
To test the performance of the model, some external Pcap files were downloaded from Github for "DoS", "DDoS" attacks, benign traffic. Additionally, real-time normal traffic Pcap files, captured by Tshark, were used.

```

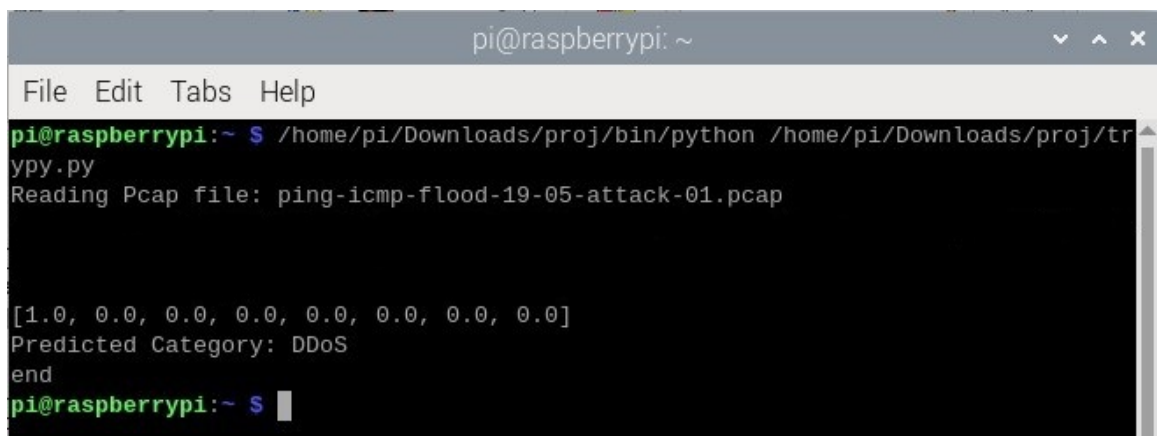
proj pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ /home/pi/Downloads/proj/bin/python /home/pi/Downloads/proj/try.py
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlan0'
** (tshark:4231) 23:42:38.313228 [Main MESSAGE] -- Capture started.
** (tshark:4231) 23:42:38.313586 [Main MESSAGE] -- File: "/home/captured.pcap"
10
tshark:
Packet capture run successfully !
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Predicted Category: Benign Traffic
end
pi@raspberrypi:~ $

```

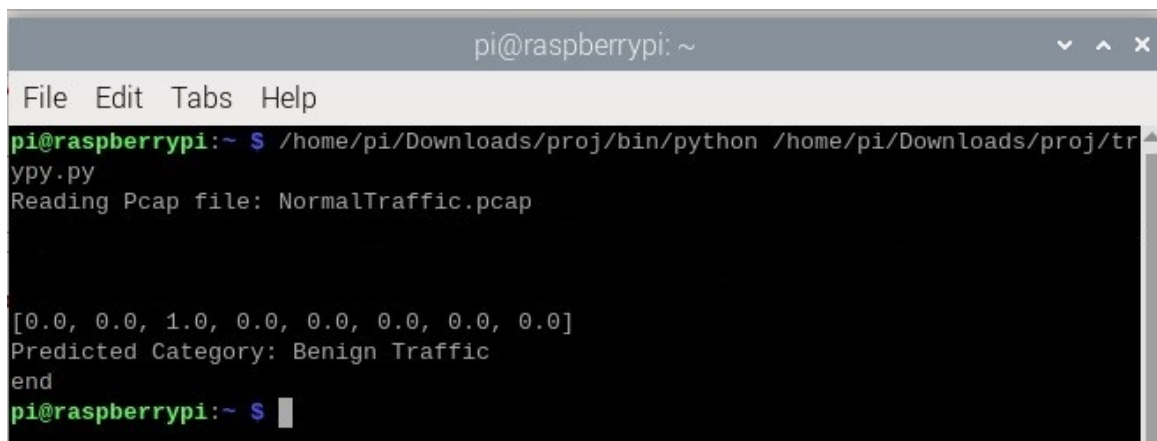
Figure 5.2: Prediction on Real-Time Traffic



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ /home/pi/Downloads/proj/bin/python /home/pi/Downloads/proj/tr  
ypy.py  
Reading Pcap file: http_slowloris.pcap  
  
[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
Predicted Category: DoS  
end  
pi@raspberrypi:~ $
```

Figure 5.3: Prediction on "DoS" Pcap File

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ /home/pi/Downloads/proj/bin/python /home/pi/Downloads/proj/tr  
ypy.py  
Reading Pcap file: ping-icmp-flood-19-05-attack-01.pcap  
  
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
Predicted Category: DDoS  
end  
pi@raspberrypi:~ $
```

Figure 5.4: Prediction on "DDoS" Pcap File

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ /home/pi/Downloads/proj/bin/python /home/pi/Downloads/proj/tr  
ypy.py  
Reading Pcap file: NormalTraffic.pcap  
  
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
Predicted Category: Benign Traffic  
end  
pi@raspberrypi:~ $
```

Figure 5.5: Prediction on Benign Traffic Pcap File

Chapter 6

Conclusion and Future Work

In conclusion, this thesis presents a Machine Learning based Network Intrusion Detection System designed for IoT devices. By deploying the system on edge, we ensure rapid response to attacks, while safeguarding the privacy of network users through localized data processing. Two prototypes were built, leveraging Raspberry Pi and ESP32 Microcontroller platforms, the latter comprising three subsystems each utilizing an ESP32 MCU.

Exploring four distinct Machine Learning algorithms, our system achieves impressive accuracy, reaching up to 99.76% and an F1-score of 94.25% for tree-based models. After a detailed evaluation and comparison, we identify the tree-based models as the most robust, demonstrating near-perfect precision and recall scores across various intrusion types. In contrast, the Neural Network exhibits limitations in overall effectiveness, while the Naive Bayes model shows inconsistency in detecting specific intrusion categories.

For future work, we aim to add support for IPv6 packets to our system. Currently, our prototype handles IPv4 traffic, but as IPv6 adoption grows, it's crucial for our NIDS to be compatible with both protocols. This will ensure that our system remains versatile and capable of securing networks that use either IPv4 or IPv6. Additionally, we intend to explore different low-end hardware platforms for deploying our NIDS. While we've successfully implemented it on the ESP32 Microcontroller, expanding to other hardware options will increase the system's accessibility and scalability. Moreover, we plan to leverage the WiFi capabilities of the ESP32 to extend our NIDS to wireless IoT networks to address the challenges posed by wireless environments. Our research will also focus on developing more complex deep learning models, such as LSTM, to further enhance detection capabilities. We plan to conduct real-time simulation tests using hping3, slowloris, and nmap to assess the system's responsiveness under different attack scenarios.

Bibliography

- [1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, “Understanding the mirai botnet,” in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.
- [2] P. A. Networks. 98 percent of iot traffic is unencrypted. [Online]. Available: <https://unit42.paloaltonetworks.com/iot-threat-report-2020/>
- [3] Cisco. What is cybersecurity? [Online]. Available: <https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html>
- [4] L. B. William Stallings, *Computer Security: Principles and Practice*. PEARSON, 2014.
- [5] S. Wilson, *Cybersecurity and Artificial Intelligence: Threats and Opportunities*. Contrast Security, 2023.
- [6] W. Samuel Marchal, Bartosz Nawrotek, *Applying artificial intelligence in Cybersecurity*. Finnish Transport and Communications Agency Traficom, National Cyber Security Centre Finland, 2024.
- [7] R. Bace and P. Mell, *Intrusion Detection Systems*. NIST National Institute of Standards and Technology, 2001.
- [8] ReasonLabs. What is intrusion? [Online]. Available: <https://cyberpedia.reasonlabs.com/EN/intrusion.htm>
- [9] A. Nasaf, “Intrusion detection system: A survey and taxonomy,” *HAL Open Science*, 2021.
- [10] H. Badgujar. What is ids (intrusion detection system) how it works? [Online]. Available: <https://medium.com/@hrushikeshbadgujar/what-is-ids-intrusion-detection-system-how-it-works-732d81a13fb5>

- [11] L. Hung-Jen, R. L. Chun-Hung, L. Ying-Chih, and T. Kuang-Yuan, "Intrusion detection system: A comprehensive review," in *Journal of Network and Computer Applications*.
- [12] T. Mehmood and H. B. Md Rais, "Machine learning algorithms in context of intrusion detection," in *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, 2016, pp. 369–373.
- [13] Host-based intrusion detection. KROLL REDSCAN. [Online]. Available: <https://www.redscan.com/services/hids/>
- [14] M. S. Aliero, A. M. Ahmad, U. S. Kalgo, and S. A. Aliero, "An overview of internet of things: understanding the issues and challenges of a more connected world," *International Journal of Computing and Communication Networks*, vol. 2, no. 1, pp. 1–11, 2020.
- [15] Artificial Intelligence. Oxford English Dictionary. [Online]. Available: https://www.lexico.com/definition/artificial_intelligence/
- [16] AI set to exceed human brain power. CNN. [Online]. Available: <http://edition.cnn.com/2006/TECH/science/07/24/ai.bostrom/>
- [17] Biggest confusion: Ai vs ml vs deep learning. Medium. [Online]. Available: <https://popatavani666.medium.com/biggest-confusion-ai-vs-ml-vs-deep-learning-8aa11343fd12>
- [18] A. L. Samuel, "Some studies in machine learning using the game of checkers." *IBM Journal of research and development*, 1959.
- [19] Everything you need to know about machine learning. Analytics Vidhya. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/>
- [20] Y. W. Qiong Liu, "Supervised learning," *ResearchGate*, January 2012.
- [21] K. Pykes. Introduction to unsupervised learning. [Online]. Available: <https://www.datacamp.com/blog/introduction-to-unsupervised-learning>
- [22] M. N. Antonio Coronato, Syed Tahir Hussain Rizvi, "A gentle introduction to reinforcement learning and its application in different fields," *ResearchGate*, January 2020.

- [23] H. Zhang, "The optimality of naive bayes," *ResearchGate*, January 2004.
- [24] O. M. Lior Rokach, "Decision trees," *ResearchGate*, January 2005.
- [25] Decision tree classification algorithm. Javatpoint. [Online]. Available: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
- [26] L. Breiman, "Random forests," *ResearchGate*, October 2001.
- [27] Random forest. Medium. [Online]. Available: <https://medium.com/@denizgunay/random-forest-af5bde5d7e1e>
- [28] Y. B. Ian Goodfellow and A. Courville, *Deep Learning*. The MIT Press, 2016.
- [29] Perceptron model: The foundation of neural networks. Medium. [Online]. Available: <https://medium.com/@ilyurek/perceptron-model-the-foundation-of-neural-networks-4db25b0148d>
- [30] V. D. F. Facundo Bre, Juan M. Gimenez, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *ResearchGate*, November 2017.
- [31] J. Schmidhuber, "Deep learning in neural networks: An overview," *ScienceDirect*, January 2015.
- [32] Raspberry pi 4 your tiny, dual-display, desktop computer. Raspberry Pi 4. [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- [33] Esp32 chips. Espressif Systems. [Online]. Available: <https://www.espressif.com/en/products/socs/esp32>
- [34] Esp-idf documentation. Espressif Systems. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>
- [35] Enc28j60 data sheet. Microchip. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf>
- [36] What is a pcap file. Endace. [Online]. Available: <https://www.endace.com/learn/what-is-a-pcap-file>
- [37] Libpcap file format. Wireshark. [Online]. Available: <https://wiki.wireshark.org/Development/LibpcapFileFormat>
- [38] R. Barry and T. F. Team, *Mastering the FreeRTOS™ Real Time Kernel*. FreeRTOS.

- [39] R. L. Ankit Thakkar, “A review of the advancement in intrusion detection datasets,” *ScienceDirect*, 2020.
- [40] UNSW Sydney - The University of New South Wales, “The Bot-IoT Dataset,” <https://research.unsw.edu.au/projects/bot-iot-dataset>, 2021.
- [41] G. V. Léo Grinsztajn, Edouard Oyallon, “Why do tree-based models still outperform deep learning on tabular data?” *Journal of Edge Computing*, July 2022.
- [42] E. Shao, “Encoding IP address as a feature for network intrusion detection,” Master’s Thesis, Faculty of Purdue University, 2019.
- [43] M. Antonakakis, “Understanding the Mirai Botnet,” *Journal of Edge Computing*, 2017.