# 1. INTRODUCTION

The network system to be dependable and efficient when conducting transactions over the IoT, security is a major concern [1]. Securing IoT infrastructure to enhance its performance is encouraged by the great rise of IoT in various industries, such as surveillance, healthcare, transportation, manufacturing industry, education, and others. Earlier IOT collected data from a variety of sensors, making it easier for cloud servers to execute or manage these transactions. To enhance the IOT infrastructure with desirable features, fog computing is recently suggested approaches [2]. Before sending the accumulated information to the cloud server, mist processing can skillfully embrace some provincial investigation of data [3]. It is reasonable for Web of Things (IoT) applications like vehicular impromptu organizations (VANETS) since it helps with keeping up with dormancy constraints in specific time-packed [4]_[11]. These developments in fog server integration with IoT infrastructure encourage hostile actors to aim their attacks at fog servers in an effort to degrade their performance. Thus, among the most significant concerns that might impact fog computing performance are security and protection of the system [12]. When it comes to providing services to real consumer applications. On the other hand, attackers are always trying to test this by conducting various assaults, like denial-of-service or distributed denial-of-service attacks [13]. The term "bot-master" describes an individual who initiates such an attack, whereas "botnet" describes a group of people doing it. cited as [14]. Phishing, spamming, click fraud, and other forms of server attacks are all within the capabilities of the bot-master, the attacker node. A botnet can be remotely controlled through a command-and-control channel. An attacker can take control of a compromised system using a command-and-control channel, which allows them to send messages and commands. By issuing these commands, the attacker can compromise the compromised network and steal data [8]. In a botnet assault, a bot-master commands a network of compromised computers, or "nodes," to conduct attacks on a server.

Maintaining a secure fog computing paradigm is an ongoing challenge. There are many solutions, meanwhile, center on making the fog server as adaptable and monitored as possible at all times. Fog servers are equipped with software-denied networking (SDN) to tackle problems with flexibility and constant monitoring [15]. Separate control plans offer a flexible device management strategy in software-defined networking (SDN), an emerging paradigm in networking that helps to make the network more adaptable, which in turn aids in network administration, traffic analysis, and routing control architectures [16], [17]. With software-defined networking (SDN), the fog computing system can be centrally controlled. What follows is a discussion of the features of the SDN-based fog computing system (V).

- Thousands of devices transmitting data over the fog can have their safe connections managed by SDN.
- Software-defined networking (SDN) allows for low-latency, real-time monitoring and awareness.
- SDN's adaptable design allows it to dynamically balance the load.
- Thousands of devices transmitting data over the fog can have their safe connections managed by SDN.

- Software-defined networking (SDN) allows for low-latency, real-time monitoring and awareness.
- SDN's adaptable design allows it to dynamically balance the load.
- Thanks to its programmability. SDN allows for application and policy customization. [18].

Because its organization control design is straightforwardly programmable through order demands, the product denied network assumes a critical part. IoT gadget investigation and the executives can be supported by a mist registering engineering in light of programming characterized organizing. The thought behind programming characterized organizing (SDN) is to separate the network into a data plane and a control plane in order to make network management more consistent. The fog computing system can gain programmability, adaptability, and variety with SDN. One major worry with high-speed networks is finding the botnet attack [19]. The presented work demonstrates an approach SDN to provide security of fog computing by identifying botnet attacks with a high detection rate. One potential countermeasure to boost system performance is to use a deep learning (DL) based detection strategy in SDN-based fog computing applications [20]. The DL approach can be adjusted to different situations to find the network's unusual behavior. SDN-based haze registering design more proficient usual behavior. SDN-based haze registering design more proficient and viable, we proposed a cross breed profound learning location strategy. It is obvious from the outcomes that the proposed strategy is prevalent as far as both execution and recognition rate.

# 2. LITERATURE SURVEY

R. Chen,W. Niu, X. Zhang, Z. To improve the progress of packet processing technologies such as New Application Programming Interface (NAPI) and zero copy and propose an efficient quasi-real-time intrusion detection system. Our work detects botnet using supervised machine learning approach under the high-speed network environment. Our contributions are summarized as follows: (1) Build a detection framework using PF_RING for sniffing and processing network traces to extract flow features dynamically. (2) Use random forest model to extract promising conversation features. (3) Analyze the performance of different classification algorithms. The proposed method is demonstrated by well-known CTU13 dataset and nonmalicious applications [21]. S. Al-mashhadi, M. Anbar, I. Hasbullah, The experimental results show our conversation-based detection approach can identify botnet with higher accuracy and lower false positive rate than flow-based approach [22] [29-30]. A. O. Proko_ev, Y. S. Smirnova With the rapid development and popularization of Internet of Things (IoT) devices, an increasing number of cyber-attacks are targeting such devices. It was said that most of the attacks in IoT environments are botnet-based attacks. Many security weaknesses still exist on the IoT devices because most of them have not enough memory and computational resource for robust security mechanisms. Moreover, many existing rule-based detection systems can be circumvented by attackers. In this study, we proposed a machine learning (ML)-based botnet attack detection framework with sequential detection architecture. An efficient feature selection approach is adopted to implement a lightweight detection system with a high performance. The overall detection performance achieves around 99% for the botnet attack detection using three different ML algorithms, including artificial neural network (ANN), J48 decision tree, and Naïve Bayes [23-25]. A botnet comprises a network of malware-infected computing devices. A malware transforms compromised computing devices into robots (bots) controlled remotely by the attacker, known as a botmaster [26-27]. Education,I.C.(2021). Artificial intelligence (AI) is technology that enables computers and digital devices to learn, read, write, create and analyze.[28]. Education,I.C.(2021, machine learning algorithms, classification analysis, regression, data clustering, feature engineering and dimensionality reduction, association rule learning, or reinforcement learning techniques exist to effectively build data-driven systems [29-30]. In supervised learning, the machine is trained on a set of labeled data, which means that the input data is paired with the desired output. The machine then learns to predict the output for new input data. Supervised learning is often used for tasks such as classification, regression, and object detection. In unsupervised learning, the machine is trained on a set of unlabeled data, which means that the input data is not paired with the desired output. The machine then learns to find patterns and relationships in the data. Unsupervised learning is often used for tasks such as clustering [31-32].

By using Machine learning algorithms to detect attacks due to their computational efficiency and high accuracy. In the next section, we will delve into several relevant studies, including botnet detection and various algorithms such as dimensionality reduction and classifiers, multistage attack detection, and concepts related to proposed various methods approaches across different datasets (N-BaIoT, IoT-23, Kitsune, MedBIoT) [33]. These studies applied chi-square feature selection to distinguish between malicious and benign data classes, achieving high-accuracy results. Notably, the RF classifier demonstrated superior performance for accuracy and speed during testing.

Another approach by Maudoux et al. [34] employed a multi-step methodology for botnet detection. They divided network packets into smaller segments using the Weka tool, applied four Decision Tree models, and enhanced detection accuracy using an AdaBoost meta classifier and a combined Forest model via the Stacking method. This approach proved effective in real-time detection of Command & Control (C&C) exchanges due to its high precision in analyzing traffic patterns.

In a study by Nithya et al. [35], Principal Component Analysis (PCA), significantly improved performance metrics when combined with algorithms like RF, Decision Tree, and SVM on the IoT-23 dataset. This highlighted the critical role of dimensionality reduction techniques in enhancing model accuracy.

Mehra et al. [36] explored various feature sets, including 36 static features extracted from ELF files and symbol tables, 13 dynamic features based on system call frequencies, and 14 network features like destination IPs and packet counts. They trained models such as Logistic Regression, SVM, KNN, and Random Forest, identifying 28 key features essential for detecting IoT botnets across different scenarios.

Tikekar et al. [37] conducted a comprehensive analysis showing that while machine learning-based botnet detection approaches follow similar logical paths, the selection of essential features plays a crucial role in achieving effective malware detection. This underscores the ongoing need for developing optimized machine learning workflows to mitigate botnet threats effectively.

IoT security measures can be implemented at both device and network levels. Device-level security often faces challenges due to varying vendor support and incomplete security implementations. Conversely, network-level solutions provide broader access to IoT networks but must address potential vulnerabilities. Detection methods depend on attack, distinguishing between single-stage attacks like DoS and worms, which are comparatively easier to detect, and multi-stage attacks such as DDoS and exploit kits, which pose greater challenges due to complex attack patterns and computational requirements [38].

IoT fingerprinting is another critical concept for identifying devices based on their behavioral characteristics. This approach aids in multiclass classification problems within IoT botnet detection by capturing device-specific behaviors from network traffic. However, the challenge lies in distinguishing genuine device

In their study, Parakash et al. tested the efficacy of three popular ML algorithms- SVM, NB, and the K-Nearest Neighbors algorithm-in identifying DDoS packets. According to the results, the KNN algorithm detects DDoS attacks with a 97% accuracy rate, while the SVM method achieves 82% accuracy and the NB algorithm 83% accuracy [33]. The authors of [34] suggested a method for detection that combines the SVM calculation with an inactive break change calculation (IA) that they had developed. They proved that their recommended process is superior and produces better outcomes. Use of NB, SVM, and neural networks is also seen in [35]. The neural network and NB models outperformed the SVM model, reaching 100% accuracy. A similar average accuracy of 95.24% was attained by Ye et al. [36] with the application of the SVM algorithm. Using algorithms like decision tree classifier and Naive Bayesian, the authors conducted 190 attained by Ye et al. [36] with the application of the SVM algorithm. Using algorithms like decision tree classifier and Naive Bayesian, the authors conducted tests in [37]. They were able to detect 99.6 percent of the time.

A subset of ML are DL algorithms. The ability to handle both organized and unstructured data types is a plus. Large amounts of data generated by IoT devices and data that is not structured do not yield improved outcomes when using ML algorithms [38]. So, when compared to more conventional ML algorithms like KNN, SVM, NB, etc., DL algorithms are suited for the Internet of Things. Malware in Internet of Things devices can be detected using a variety of DL and hybrid DL techniques [39][41]. A method for protecting the IoT ecosystem from malware and cyber assaults such as distributed denial of service (DDoS), brute force, bot, and infiltration was detailed in [42].

# 3. System Analysis

## 3.1 EXISTING SYSTEM

These days, botnet attack detection is a hot topic among researchers [28][30]. Finding infected devices before they can launch malicious attacks on the network is the primary goal of botnet detection. To protect the network from botnet attacks, the authors offer a plethora of solutions. Anomaly detection strategies utilizing artificial intelligence, particularly ML and DL algorithms, are the primary Paragraph attacks, and the authors offer a plethora of solutions. Anomaly detection strategies utilizing artificial intelligence, particularly ML and DL algorithms, are the primary emphasis of these approaches. To identify botnets, researchers have utilized ML and hybrid ML methods including BayesNet (BN), Backing Vector Machine (SVM), J48, Choice Tree (DT), and Guileless Bayes (NB) in various examinations [21][23]. Moreover, Al calculations: regulated, solo, and semi-administered.

**Disadvantages**

The SDN-based fog computing architecture using hybrid deep learning detection. It is clear from the results that the recommended strategy is unrivaled as far as both execution and recognition rate. Since it is programmable, customizing the arrangements and applications is absurd.

## 3.2 IOT

IoT,refers to a network where multiple devices or objects are interconnected These gadgets exchange messages with one another and utilize transmitted data. IoT devices perform diverse functions across various sectors, such as industrial machinery, healthcare instruments, smart lighting, televisions, smartphones, and intelligent appliances like refrigerators.

The foundational architecture of IoT comprises three technological layers facilitating device communication. The initial layer is the physical or perception layer, where end devices act as interfaces for users, enabling them to access necessary information. The second layer is the network layer, through which devices acquire data via cellular, LAN, or satellite networks, utilizing specialized protocols. These attacks can target the end devices and exploit their vulnerabilities due to their sophisticated hardware and functionalities.

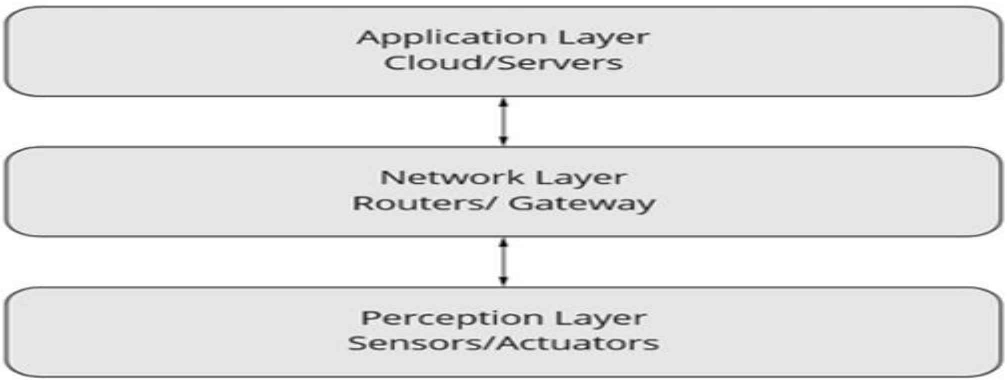


*Figure 3.2 Fundamental IoT Architecture*

Numerous botnet attacks, such as Mirai, Reaper, Echobot, Emotet, Gamut, and Necrus malware [12], have targeted IoT devices, exploiting their vulnerabilities. Current IoT device mechanisms often lack effective measures to mitigate these threats. The IoT devices continue to rise projected to exceed 75 billion by 2025 according to Statista [13] — concerns regarding data privacy and security are intensifying. This proliferation means IoT gadgets are being incorporated into everyday life, amplifying the prominence of the threat landscape. These devices, interconnected and forming small hubs on the internet, pose significant security challenges.

# 4. SOFTWARE REQUIREMENTS SPECIFICATIONS

In a fog computing environment that is built on software-defined networking (SDN), the system recommends a deep learning framework that is efficient at identifying Botnet attacks.

• A Botnet attack and benign samples are both included in the N_BaloT Dataset is used for training and testing.

• The suggested method is surveyed using metrics such as accuracy, F1-score, review, precision, and other noteworthy execution assessment proportions of machine and deep learning computations.

• The strategy of 10-crease cross-approval was also utilized to ensure impartial outcomes.

**Advantages**

• When great many gadgets are connected over the haze for information move, the framework can manage their safe connection.

• The system can provide low-latency, real-time monitoring and awareness.

• Thanks to its adaptable design, the system can dynamically distribute the workload.

## 4.1 SYSTEM REQUIREMENTS:

### 4.1.1 H/W System Configuration: -

| | | |
|---|---|---|
| ➤ Processor | : | Intel core I3 |
| ➤ RAM | : | 2 GB |
| ➤ Hard Disk | : | 126 GB |
| ➤ Key Board | : | Windows Keyboard |
| ➤ Mouse | : | Optical Mouse |
| ➤ Monitor | : | SVGA |

### 4.1.2 SOFTWARE REQUIREMENTS:

| | | |
|---|---|---|
| Operating system | : | Windows 10 and above. |
| Coding Language | : | Python. |
| Front-End | : | Python. |
| Back-End | : | Django-ORM |
| Data Base | : | MySQL (WAMP Server). |

## 4.2 Study of Feasibility

This step includes examining the undertaking's plausibility and advancing a strategic agreement with a significant level arrangement and assessed costs.

**There are mainly three categories:**

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

### 4.2.1 ECONOMICAL FEASIBILITY

The reason for this study is to inspect the framework's expected financial impact on the organization. The enterprise has limited resources for investing in the system's research and development. All cases must have a rationale. As a result, the built system was able to stay within budget, and most of the advancements used are uninhibitedly accessible. Purchasing just the customized items was important.

### 4.2.2 TECHNICAL FEASIBILITY

The technical requirements, or viability, of the system are the focus of this study. It is imperative that no system put a heavy strain on the current state of technical resources. As a result, the available technical resources will be put to good use. The client will be subject to high expectations as a result of this. There should be little to no change needed to implement the designed system, hence its requirements should be low.

### 4.2.3 SOCIAL FEASIBILITY

The study's focus is on gauging the user's level of satisfaction with the system. Instruction on how to make the most of the framework is a piece of this. The framework shouldn't cause the client to feel scared rather they should embrace it as Is to gauge the user's level of satisfaction with the system. Instruction on how to make the most of the framework is a piece of this. The framework shouldn't cause the client to feel scared; rather, they should embrace it as an essential tool. Users' level of acceptance is directly proportional to the efficacy of the strategies used to acquaint and educate them with the system. Since he is the system's end user, it's important to boost his self-assurance so he can provide helpful feedback.

# 5. PROPOSED ARCHITECTURE

## 5.1 Proposed Architecture:

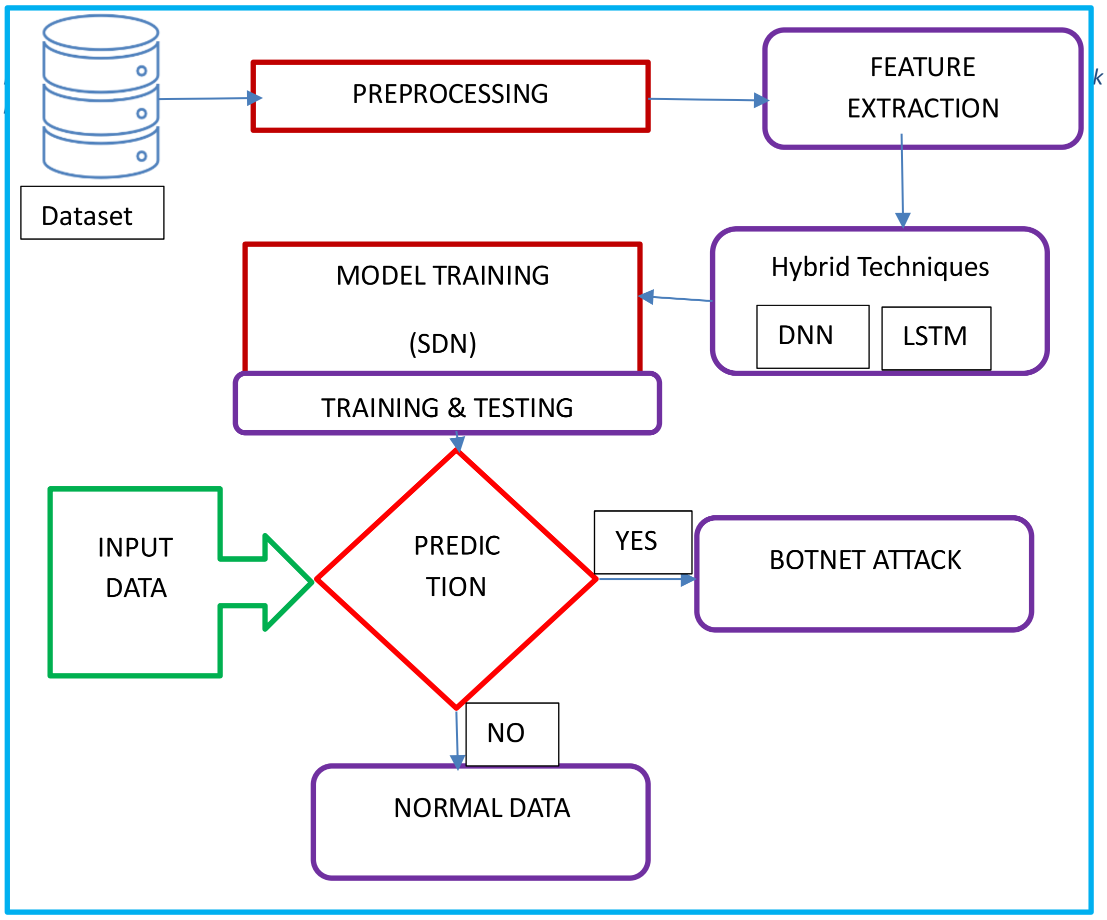


*Figure 5.1: The architecture of the proposed system integrates three critical components: SDN for flexible network man- agement, LSTM networks for anomaly detection, and fog computing for decentralized data processing*

The lifecycle of a botnet encompasses several phases: formation, Command & Control (C&C), attack execution, and post-attack activities [14]. It begins with the compromise of vulnerable machines to establish a chain of bots. Next, the Command & Control mechanism coordinates instructions for launching attacks. Bots are then used to execute these attacks. In the post-attack phase, the botmaster may attempt to compromise additional bots to reinforce the botnet's capabilities [14].

The rapid evolution of malicious activities has led to increasingly sophisticated botnet attacks. These attacks serve various purposes, including Distributed Denial of Service (DDoS) attacks [15], theft of cryptocurrencies [16], and brute-force attacks to obtain credentials [17]. Various types of IoT malware have gained notoriety, with the Mirai malware being particularly infamous since its 2016 attack, demonstrating considerable destructive power [18]. The publication of Mirai's source code online has facilitated the development and enhancement of similar attacks. According to a Security Intelligence report [19], IoT attacks rose by nearly

500% in 2021 compared to 2019. This trend highlights the growing diversity and evolution of malware, with examples such as Mozi, Echobot, Zeroshell, Gafgyt, and Loli emerging to compromise existing infrastructures and deploy payloads.

## 5.3 Botnet Detection Methods

According to R. B. Auliar et al. [20], the detection techniques for Mirai and similar botnet attacks can be categorized into two main groups:

1. **Signature-Based Detection:** This approach involves identifying known patterns or signatures of malicious activity associated with specific botnets like Mirai. Signature-based detection relies on predefined rules or patterns derived from analyzing the behavior or characteristics of known attacks. When network traffic or system behavior matches these signatures, consider it as a botnet.

2. **Anomaly-Based Detection:** In distinction to signature-based methods, anomaly-based detection focuses on identifying deviations from normal or expected behavior within network traffic or system activities. This method establishes a baseline of normal behavior and flags any activities from this baseline as potentially malicious. These methods are more difficult for identifying and mitigating botnet attacks, offering different strengths in detecting both known and emerging threats.

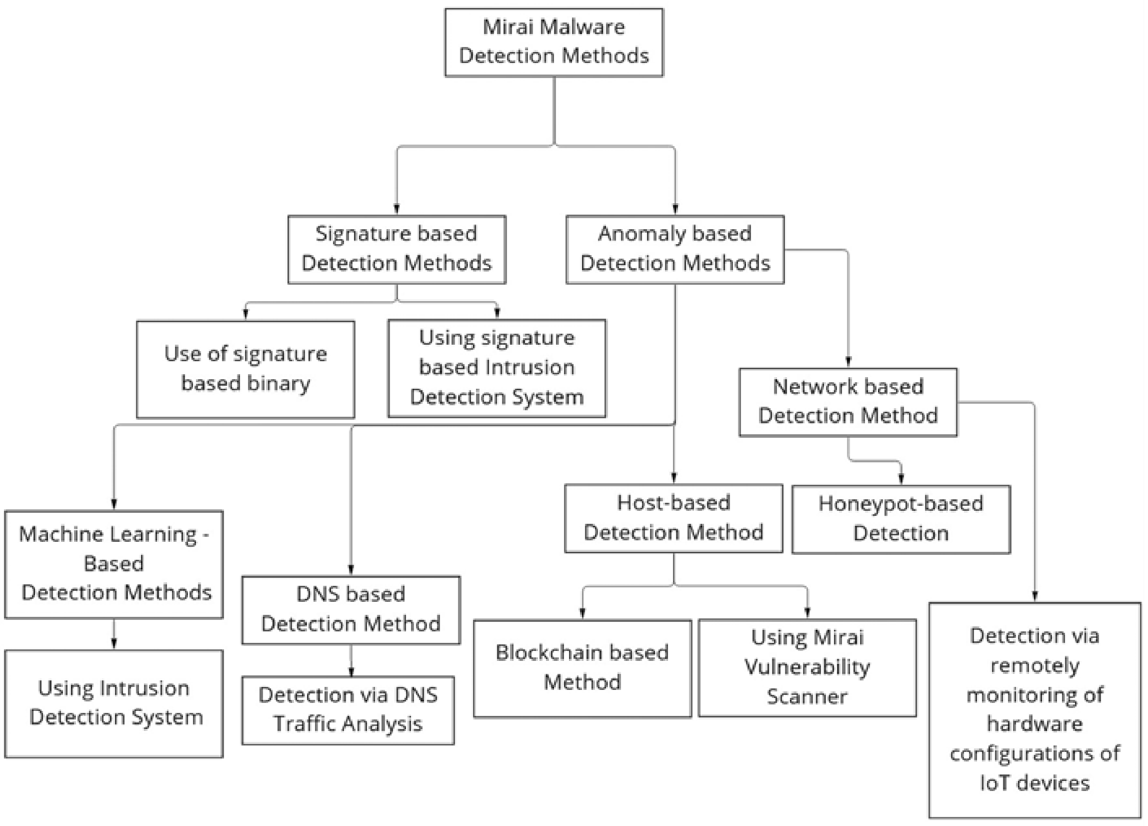

Mirai Malware Detection Methods
Signature based Detection Methods
Anomaly based Detection Methods
Use of signature based binary
Using signature based Intrusion Detection System
Network based Detection Method
Machine Learning - Based Detection Methods
DNS based Detection Method
Host-based Detection Method
Honeypot-based Detection
Using Intrusion Detection System
Detection via DNS Traffic Analysis
Blockchain based Method
Using Mirai Vulnerability Scanner
Detection via remotely monitoring of hardware configurations of IoT devices

*Figure 5.3 Mirai Malware Detection Methodology*

Anomaly-based detection techniques are a type of malware detection that identifies abnormal or unusual patterns in network traffic, system behavior, or other data sources. Unlike signature-based detection, which relies on known patterns of malicious activity, anomaly-based techniques do not require specific signatures. Instead, they establish a baseline of flag deviations from this baseline as potential indicators of malicious activity.

Key aspects of anomaly-based detection include:

1. **Traffic Volume Analysis:** Monitoring and analyzing network traffic volumes to identify unexpected spikes or anomalies that could indicate a botnet or other malicious activity.

2. **Suspicious Ports Detection:** Identifying unusual or suspicious use of network ports that may suggest unauthorized access attempts or communication associated with botnets.

3. **Unusual System Behavior:** Monitoring system activities and behaviors to detect unauthorized access attempts, unusual file access patterns, or abnormal resource usage that could be indicative of malware activity.

Anomaly-based detection methods are valuable for detecting new and previously unknown botnet attacks because they do not rely on specific signatures or patterns associated with known threats. These techniques often incorporate machine learning algorithms, statistical analysis, or heuristic approaches to continuously adapt and improve detection capabilities against evolving threats.

Among the various subcategories of anomaly-based detection methods, host-based detection involves monitoring the internal systems of IoT devices exclusively. This method incorporates innovative approaches such as Blockchain-based detection and the use of Mirai Vulnerability scanners. Blockchain-based detection, for instance, employs autonomous systems to monitor packet counts against predefined thresholds. When thresholds are exceeded, IP addresses are automatically blocked, preventing unauthorized access [22]. Conversely, Mirai scanners simulate brute force attacks on open ports within IoT networks to find access points for attacks. Detected vulnerabilities are promptly reported, enabling network security enhancements [23].

DNS-based detection, another subcategory, monitors DNS traffic to detect malware, particularly effective against DNS-based botnets. Novel DNS-based schemes have been developed to combat Mirai-like malware, enhancing detection capabilities [24].

Network-based detection involves scanning network traffic and configurations. Techniques such as Mirai malware imitation assess IoT device vulnerabilities and implement security measures accordingly. Additionally, honeypot configurations are utilized to capture suspicious activities within IoT networks by simulating vulnerable systems that attract malicious traffic [25].

Machine Learning-based detection represents a sophisticated approach to identifying malicious behavior in network traffic. By leveraging advanced data processing techniques and automation, Machine Learning models discern anomalies and predict malicious intent with high accuracy and efficiency. This method is particularly advantageous in IoT environments where traditional intrusion detection systems may struggle due to limitations in cryptographic mechanisms [26].

Given the evolving nature of botnet attacks, the focus on Machine Learning-based detection methods in this paper aims to explore their efficacy in preventing threats and safeguarding IoT environments. These methods provide rapid detection, precise prediction of anomalies, and effective prevention of botnet attacks, underscoring their critical role in IoT security strategies.

## 5.4 FOG COMPUTING ARCHITECTURE:

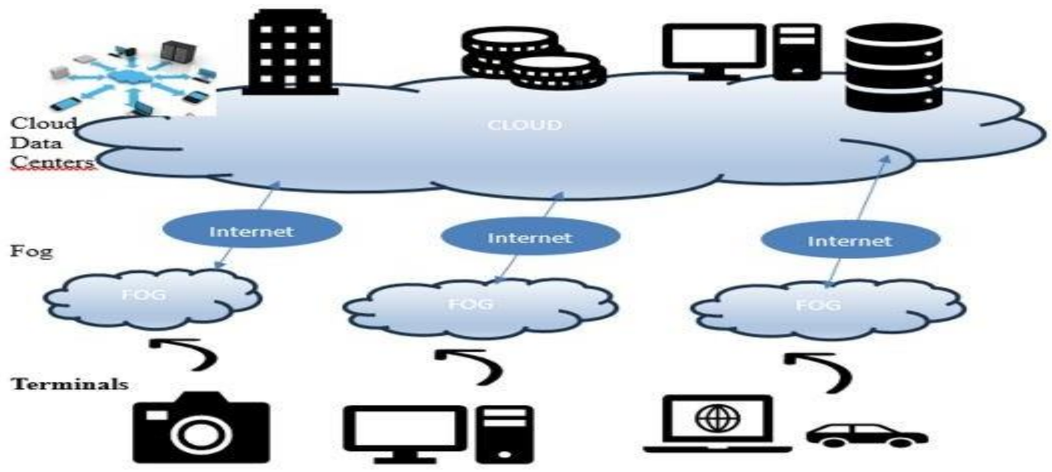


*Figure 5.4: FOG Computing Architecture*

## 5.5 Modules:

### 5.5.1 Service Provider for Authentication:

We will enter username and password for login in into service provider.  After login accessing data available in training data as well as testing data. Get a look at the prepared and tried precision in a bar diagram, see the consequences of the prepared and tried exactness, see the attack type ratio of data sets with the predicted attacks.

### 5.5.2 Observe and Allow Users:

The admin can allow the user and enroll them and provide grant permission to access information.

## 5.6 Research Design and Research Methodology

The research methodology begins with preparing the data, ensuring the dataset's structure is well-defined, understanding the distribution of data points, and carefully selecting samples for experimental analysis. The next phase focuses on feature selection, evaluating methods such as filter, wrapper, and embedded techniques. But computationally intensive, and embedded methods strike a balance by incorporating feature selection within the model training process.

Following feature selection, the modeling process involves scaling the data to ensure consistency in model inputs, employing nested cross-validation to accurately assess model performance, selecting appropriate classifiers tailored to the classification task, and utilizing performance metrics to evaluate model effectiveness.

Finally, the research concludes by identifying optimal methods applied in both binary and multiclass classification models, highlighting their efficacy in accurately classifying IoT botnets based on the experimental outcomes.

### 5.6.1 Data Set:

For the IoT botnet detection problem, The existing methods well trained on the non-IoT dataset cannot be utilized on the IoT dataset as they will demonstrate high false-positive and high false-negative accuracy results [41]. Hence, the nature of the dataset takes a prominent place in IoT botnet detection. In this research, the MedBIoT dataset will be utilized [42]. In comparison to N-BaIoT (2018), Bot-IoT (2018), and IoT-23(2020) datasets, the N_BaIoT(2023) dataset contains a medium-sized IoT network infrastructure with 83 devices and 100 features.

In this research framework, various combinations of classes are aggregated to form new datasets, which serve as testing grounds of a machine learning pipeline in addressing both binary and multiclass classification challenges.

Initially, the experiments focus on distinguishing between malicious and benign behaviors. Each dataset includes instances of specific malware types like Mirai, Bashlite, Torii, along with corresponding non-infected traffic segments. It will provide, how well the machine learning pipeline can differentiate between malicious botnet traffic and legitimate network activity.

In subsequent experiments, the scope expands to include multiple malware categories and their associated attack stages. One dataset is structured with 8 distinct classes, while another extends to 12 classes. This broader classification approach aims to develop models capable of not only identifying different types of malwares but also detecting specific stages of attacks and identifying compromised IoT devices.

These experiments are designed to comprehensively test the machine learning pipeline's ability to handle complex classification tasks within IoT botnet detection, across various scenarios and class combinations.

*Table 5.6: Class Combinations*

| Model | Name of classes |
|---|---|
| 8 classes model | Mirai and switch, Mirai and lock, Mirai and light, Mirai and fan, Bashlite and switch, Bashlite and lock, Bashlite and light, Bashlite and fan categories |
| 12 classes model | spread and switch, spread and lock, spread and light, spread and fan, C&C and switch, C&C and lock, C&C and light, C&C and fan, legitimate and switch, legitimate and lock, legitimate and light, legitimate and light, legitimate and fan |

To ensure balanced representation across binary and multiclass classification tasks, 3000 examples were tried from each dataset file and assigned to their respective labels. This approach results in datasets comprising 6000 instances for binary classification (malware versus benign traffic), 12000 instances for distinguishing among three types of botnet attacks and legitimate traffic, 24000 instances for an 8-class classification model, and 36000 instances for a 12-class classification model.

Given the original datasets vary in size, employing these techniques in which each class within the dataset has an equal proportion of data points. This balanced approach mitigates the risk of bias during model training by preventing any single class from dominating the training dataset based on its initial quantity of data. Thus, the feature selection process can proceed effectively, leveraging datasets where each class is represented proportionally to achieve more reliable and generalized machine learning models.

### 5.6.2 Pre-Processing

In the process of preparing the datasets for analysis and classification within our system, several pre-processing steps were essential. Here's an overview of how we transformed the raw data from pcap files into a standardized representation using the Scapy library:

1. **Dataset Overview**: The dataset consists of two pcap files—one for training and one for testing. These files contain a mixture of benign traffic and various types of malicious traffic, encompassing over a dozen different botnets.

2. **Utilization of Scapy**: Scapy, a powerful Python library for packet manipulation, was instrumental in converting network packets into accessible objects within our code. This capability facilitated easier manipulation and extraction of data from the network logs.

3. **Handling Large Datasets**: Given the dataset's size and diversity, we opted to load each pcap file and splitting it. While this approach may be slower initially due to loading times, it ensures that no information is lost during the process. Specifically:

- The larger training dataset is divided into two parts for more manageable processing, accommodating resource constraints.

- The testing dataset, being smaller, was loaded as a single entity for simplicity and efficiency.

4. **Creating Bidirectional Flows**: The next critical step was to convert the dataset's information into bidirectional flows. Bidirectional flows represent the interactions between two specific machines during a session, encapsulating the packets exchanged. This change was made using Scapy's `sessions` method, which organizes and saves these connections into a Python dictionary.

5. **Saving Flows**: Once converted into bidirectional flows, the training and testing datasets were saved into separate .flow files. These files serve as structured representations of the connections between machines, preserving the essential data for the subsequent phases of analysis and classification.

### 5.6.3 Feature Selection

In machine learning classification tasks, feature selection plays a critical role, particularly in high-dimensional datasets such as MedBIoT, which contains 100 features. Managing high-dimensional datasets for data acquisition and analysis, posing challenges in IoT environments.

**There are several approaches to feature selection:**

- **Filter methods** evaluate features independently of the classifier, using statistical measures to rank them.

- **Wrapper methods** select features on model performance, employing iterative techniques that assess subsets of optimal combination.

- **Embedded methods** integrate feature selection into the model training process itself, optimizing feature selection alongside parameter tuning to enhance model efficiency.

Machine Learning models are most informative features while disregarding those that contribute minimally or negatively to classification accuracy. This systematic approach ensures that cybersecurity analysts can effectively interpret network traffic data and respond promptly with appropriate incident handling measures.

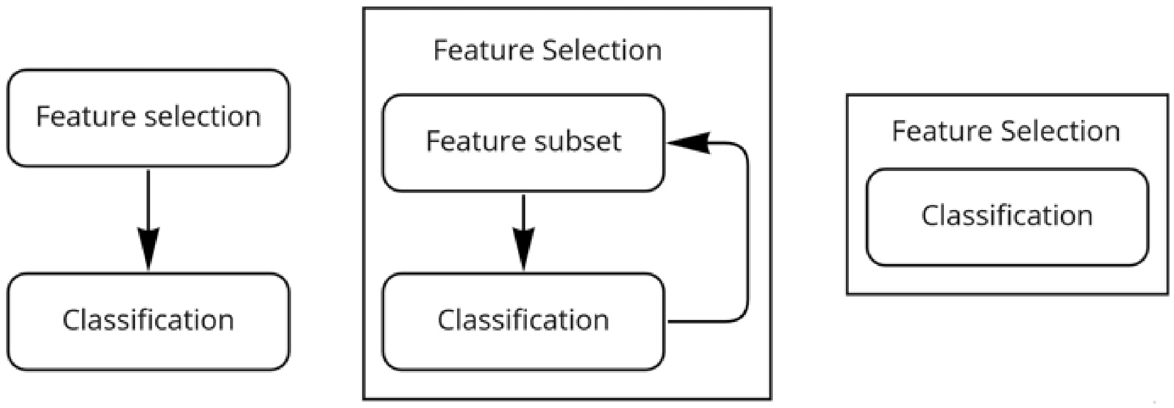


*Figure 5.6.3: feature selection*

### 5.6.4 Selected Feature Selection Method for Research:

In handling datasets with categorical class labels, converting these labels into numerical values is essential for computational efficiency. In initial experiments, datasets were consolidated and sampled. Specifically, Fisher score calculations were employed to assess the different types of malware (Mirai, Bashlite, Torii) and benign traffic.

The Fisher score method focuses on evaluating the discriminatory power of features within binary classification tasks. For instance, it quantifies how well each feature can differentiate between Mirai malicious data and benign data, as well as for other malware types like Bashlite and Torii.

To implementation of the Fisher score in Python facilitated precise control over each iteration of the feature selection process. This approach allowed for the systematic evaluation of features across various experiments, ensuring consistency and reliability in feature ranking. Ultimately, the top 20 features with the highest Fisher scores were selected and presented in descending order of importance, providing insights into which features contribute most significantly to classification accuracy.

Throughout these experiments, the number of features varied, reflecting the diversity in dataset compositions and the need to adapt feature selection strategies accordingly. This methodical approach ensures that the selected features are optimal for enhancing classification performance in identifying and distinguishing different types of networks within IoT environments.

The system's module output, each line of the .flow file encapsulates bidirectional communication details between two machines, along with relevant feature values earmarked for subsequent classification tasks. The catalogs parsed data from the flow file, detailing various features essential for classification. Notably, certain extracted information, such as Host name and URLGETEntropy, pertains specifically to HTTP and HTTPS connections, reflecting their contextual relevance.

For clarity, Listing 3.5 exemplifies the initial lines of the training .flow file, although truncated to enhance readability. This excerpt offers a glimpse into how data is structured within the file, showcasing the formatted representation of bidirectional conversations and associated feature values crucial for the classification process. This systematic approach ensures that pertinent information is effectively captured and utilized for analyzing network traffic behaviors and detecting anomalies or malicious activities within the IoT environment.

The pre-processing phase of our system, aimed at preparing network traffic data for classification, involves several key steps and efficient analysis. Here’s a summary of these steps:

1. **Data Splitting or Filtering**:

- Initially, we either split the dataset into manageable segments or filter it using tools like `tshark` and `editcap`. First step is more difficult for handling large datasets efficiently. If the system's resources allow, loading the entire dataset at once may be feasible, but often splitting or filtering is necessary to manage memory and processing constraints effectively.

2. **Loading and Conversion to Bidirectional Flows**:

- The next step involves loading the processed data into our system. Using the Scapy library, we convert the raw packet data into bidirectional flows. Each flow represents the interaction between two specific machines during a session, encapsulating the relevant packets exchanged.

3. **Feature Extraction and Saving to .flow Files**:

- During feature extraction, we identify and extract the information deemed useful for classification. This could include attributes such as source and destination IP addresses, ports, protocols, and any other relevant metadata. These features are then saved into structured .flow files, where each line corresponds to a connection or flow.

4. **Implementation of Helper Python Script**:

- To streamline this pre-processing phase, we developed a Python script named `parser.py`. This script:

  - Accepts a pcap file as input.

  - Utilizes Scapy’s `sessions` method to process packets and transform them into bidirectional flows.

  - Iterates through each session, extracts the relevant flows, and saves them into respective .flow files.

By automating these steps with the `parser.py` script and leveraging Scapy's capabilities, the network traffic data is transformed into a structured setup for subsequent classification tasks. This approach not only enhances the efficiency of data handling but also prepares the dataset comprehensively for the classification phase, various machine learning algorithms to find and classify potential security threats within the network traffic.

**5.6.5 Implementation**

1. **Analysis**: This initial step involves conducting a forensic examination of the data extracted from logs. The primary objective is to observe and understand the behavior of threats within the network and system environment. This phase is more difficult for gaining insights into how malicious entities operate and interact within the network infrastructure.

2. **Pre-Processing**: This step serves dual purposes:

- **Data Transformation**: The raw log data undergoes transformation into a structured format that facilitates easier manipulation and analysis. This may include parsing logs, converting formats, and ensuring uniformity in data representation.

- **Feature Extraction**: Features relevant to network connections are extracted from the transformed data. These features capture key attributes or characteristics of network activities, which are essential for subsequent classification tasks.

3. **Classification**: In this stage, each connection's feature vector, extracted in the pre-processing step, is classified. Classification algorithms assign labels to these feature vectors based on learned patterns or rules. The goal is to categorize connections as either benign or malicious, thereby identifying potential threats within the network.

4. **Evaluation**: Upon classification, the system evaluates the results to measure the classification model. This involves:

- Computing metrics such as accuracy, precision, recall, and F1-score to quantify the model's performance.

- Generating visualizations and statistics to find into the classification outcomes.

- Analyzing these metrics and visuals helps in determining the efficacy of the solution in detecting and distinguishing between malicious and benign connections.

5. **Decision**: The final phase, the Decider, involves using the results from the evaluation phase to make informed decisions. Specifically, this phase identifies and flags malicious IP addresses present in the network trace logs. It leverages the insights gained from the classification and evaluation steps to prioritize and take appropriate action against identified threats.

## **5.6.6 Tools and Technologies Used**:

- **Python 3.6**: Chosen for its versatility and extensive libraries suitable for machine learning, and scripting.
- **Scikit-Learn**: Utilized for its comprehensive implementation of machine learning algorithms, enabling efficient model training and evaluation.
- **Wireshark**: Employed for its graphical interface, facilitating the visual analysis of network traces, which aids in understanding network behaviors.
- **Scapy**: Used for packet manipulation, allowing scripts to parse, modify effectively**.**

**PYTHON:**

- Python is an object-oriented, undeniable level, intelligent, and deciphered programming language. It is made to be not difficult to peruse and comprehend.

- At runtime, the interpreter processes Python, making it an interpreted language. Before running your software, compiling it is not necessary. This is very much like PHP and PERL.
- Python's interactive nature means that programmers may sit down at a command prompt and have one-on-one conversations with the interpreter.
- Python is compatible with the Item Arranged programming approach, which includes encasing code inside objects.
- It is also provide broad variety of programs, including web browsers, games, and basic text processing, Python is an excellent language for programmers at the starting level.

PYTHON’S ORIGINS

- Guido van Rossum of the Netherlands' Public Exploration Establishment for Arithmetic and Software Engineering made Python in the last part of the '80s and mid-'90s.
- You can't copyright Python. Similarly, as with Perl, the GNU Overall Population Permit (GPL) presently makes Python's source code public.
- The establishment is presently liable for keeping up with Python, yet Guido van Rossum keeps on assuming an urgent part in directing its encouraging.

FEATURES OF PYTHON

Some of the features of Python are:

- Simple and easy to learn: Python's syntax is well-defined, and the language having very less keywords. In this way, the learner is able to acquire the language with relative ease.
- Python code is simpler on the eyes since it is more compact and clear cut.
- Python's source code is moderately simple.
- Python has an enormous standard library that is interoperable with UNIX, Windows, and Mac, making it especially versatile.
- Python's intuitive mode works with the testing and troubleshooting of code pieces in an intuitive fashion.
- Python is very portable because it has a consistent interface and can run on many different types of hardware.
- The Python interpreter is extensible in the sense that it can have low-level modules added to it. Programmers can enhance or personalize their tools with these modules to make them more efficient.

- Python's structure and support make it more suitable for big programs compared to shell scripting, making it scalable.

A long number of useful features is available in Python:

- Besides object-oriented programming (OOP), it also works with structured and functional programming.
- It has two main uses: as a scripting language and as byte-code for creating big applications.
- It takes into account dynamic sort confirmation and offers exceptionally theoretical unique information types.
- Programmed rubbish pickup is upheld by IT.
- Java, CORBA, ActiveX, COM, and C++ are all compatible with it.

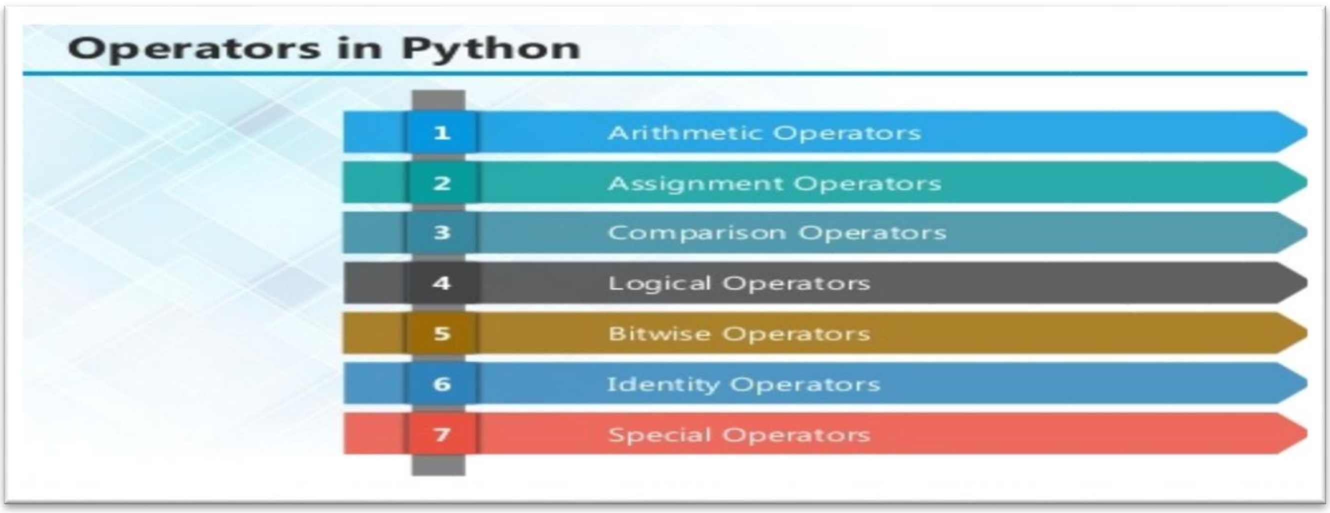


*Figure 5.6.1: Operators of Python*

### 5.6.7 Classification:

It sounds like your implementation process is well-structured and utilizes the Scikit-learn library effectively for machine learning tasks.

**Choice of Scikit-learn**:

- Scikit-learn is chosen as the primary library for implementing the system due to its comprehensive collection of machine learning algorithms and utilities. This simplifies the implementation process by providing ready-to-use tools for training and evaluating classifiers.

2. **Incremental Development Approach**:

  - You adopted an incremental development approach, starting with testing the system using data from a single botnet type during the pre-processing phase. This initial phase helped validate the functionality and performance of the framework.
  - Subsequently, you expanded the implementation to handle all botnet data available in the datasets..

3. **Classifier Selection**:
  - During testing and implementation, you experimented with several popular classifiers, including Support Vector Classifiers (SVC), Decision Trees, and Neural Networks.

## 5.7 Leveraging LSTM Algorithms for Botnet Detection in Deep Learning

Botnets represent a significant threat to network security, encompassing networks of compromised devices controlled by malicious actors. Traditional methods of detecting botnets often struggle with the evolving nature of these threats. Deep learning techniques, particularly LSTM networks, have shown promise in enhancing botnet detection capabilities due to their ability to capture temporal dependencies in data streams. This essay explores the application of LSTM algorithms in botnet detection, highlighting their advantages, challenges, and future prospects.

**5.7.1 Introduction:** The proliferation of botnets poses severe challenges to cybersecurity, with attacks ranging from DDoS (Distributed Denial of Service) to data exfiltration. Conventional signature-based detection methods are inadequate against polymorphic and stealthy botnet variants. Deep learning offers new avenues for detecting these sophisticated threats by leveraging the neural networks to find complex patterns from huge amounts of data.

**5.7.2 Understanding LSTM Networks:** LSTM networks are a type of RNN to provide long-term holding in sequential data. Unlike traditional RNNs, LSTMs incorporate to allow them to remember information over extended time intervals. This architecture makes LSTMs well-suited for time-series data analysis, such as network traffic logs, where understanding sequences of events is crucial for identifying anomalies indicative of botnet activities.

**5.7.3 Application of LSTMs in Botnet Detection:** In botnet detection, LSTMs can be proficient on sequences of network traffic, such as packet sizes, inter-arrival times, and protocol headers. By processing these sequences, LSTMs can learn normal patterns of network behavior and detect deviations that may signal botnet activities. The ability of LSTMs to model temporal dependencies enables them to distinguish between legitimate user behavior and botnet-driven anomalies effectively.

**5.7.4 Advantages of LSTM for Botnet Detection:**

1. **Sequence Learning:** LSTMs excel in learning from sequences, making them ideal for capturing the dynamic and evolving nature of botnet behaviors.

2. **Feature Extraction:** LSTMs can extract Features automatically by using RNN technique.

3. **Anomaly Detection:** By learning normal patterns, LSTMs can detect subtle deviations indicative of botnet activities, even in encrypted traffic.

**5.7. 5 Challenges and Considerations:**

1. **Data Imbalance:** Imbalanced datasets (where botnet traffic is rare compared to normal traffic) can bias LSTM models, requiring techniques like oversampling or cost-sensitive learning.

2. **Training Complexity:** LSTMs are computationally intensive and large data can be obtained in cybersecurity domains.

3. **Interpretability:** Deep learning models, including LSTMs, are often considered black boxes, showing how large data can be critical for cybersecurity analysts.

**Future Directions:** Future research in LSTM-based botnet detection could focus on:

1. **Hybrid Models:** Integrating LSTMs with various deep learning and Machine learning algorithms to enhance detection accuracy.

2. **Real-Time Detection:** Optimizing LSTM architectures for real-time analysis to mitigate botnet threats promptly.

3. **Adversarial Robustness:** Developing LSTMs that are resilient to adversarial attacks aimed at evading detection.

LSTM algorithms represent a promising frontier in botnet detection within the realm of deep learning. Their ability to learn from sequential data and detect anomalies makes them valuable tools in combating increasingly sophisticated botnet threats. While challenges such as data imbalance and interpretability remain, ongoing research and advancements in LSTM architectures hold the security landscape against botnets.

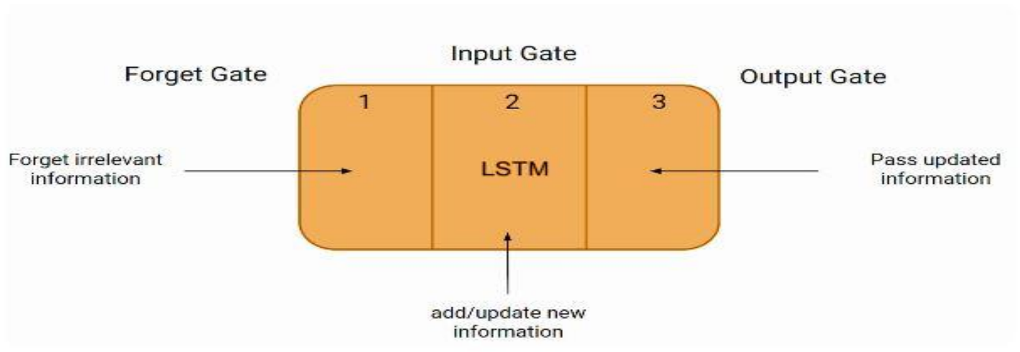


*Figure5.7.5: LSTM with Gates*

## 5.8 Utilizing RNN Algorithms for Botnet Detection in Deep Learning

Botnets pose a significant threat to cybersecurity, leveraging networks of compromised devices for malicious activities. Previous methods often struggle with the dynamic and stealthy nature of botnet behaviors. This essay explores the application of Recurrent Neural Network (RNN) algorithms, a fundamental deep learning technique, for enhancing botnet detection capabilities. It discusses the principles behind RNNs, their application in sequential data analysis, and their potential in identifying anomalous patterns indicative of botnet activities.

**5.8.1 Introduction:** With the proliferation of interconnected devices and the rise of IoT (Internet of Things), the threat posed by botnets has grown exponentially. These networks of compromised devices, controlled by malicious actors, can execute coordinated attacks such as DDoS, spam dissemination, and data theft. Traditional signature-based methods of detecting botnets are increasingly inadequate against polymorphic and evolving botnet variants. Deep learning, particularly RNNs, offers a promising approach to mitigate these threats by leveraging the neural networks from sequential data.

**5.8.2 Understanding Recurrent Neural Networks (RNNs):** RNNs techniques are the part of neural networks designed to handle sequential data by maintaining a state or memory of previous inputs. RNN can be suitable for applications where understanding the order and timing of events is crucial, such as natural language processing, time-series analysis, and in our context, network traffic monitoring for botnet detection.

**5.8.3 Application of RNNs in Botnet Detection:** In the domain of cybersecurity, RNNs can be applied to analyze sequences of network traffic data, including packet headers, traffic volume patterns, and communication protocols. By training on labeled datasets containing both normal and botnet-infected traffic, RNNs can learn the behavior that are indicative of botnet activities. This learning process allows RNNs to detect anomalies in real-time or near real-time, thereby enhancing the responsiveness of security systems to emerging threats.

**5.8.4 Advantages of RNNs for Botnet Detection:**

1. **Sequential Modeling:** RNNs excel in modeling sequential dependencies, making them effective in capturing the dynamic and evolving nature of botnet behaviors.
2. **Real-Time Detection:** RNNs can process streaming data which is crucial for mitigating the impact of botnet attacks promptly.
3. **Adaptive Learning:** RNNs can adapt to changes in botnet strategies and behaviors over time, thereby improving detection accuracy in the face of evolving threats.

**5.8.5 Challenges and Considerations:**

1. **Vanishing and Exploding Gradient Problem:** RNNs are prone to issues such as vanishing or exploding gradients, which can affect their ability to learn long-term dependencies effectively.

2. **Training Data Availability:** Effective training of RNNs requires large volumes of labeled data, which can be challenging to obtain in cybersecurity domains due to the rarity of botnet traffic and the need for diverse attack scenarios.

3. **Model Interpretability:** Like other deep learning models, RNNs can be considered black boxes, making it more crucial to make decisions, which is critical for cybersecurity analysts and decision-makers.

**Future Directions:** Future research in RNN-based botnet detection could focus on:

1. **Improved Architectures:** Developing enhanced RNN architectures or hybrid models that combine RNNs with other techniques to achieve higher detection accuracy.

2. **Enhanced Training Techniques:** Addressing issues like data scarcity through synthetic data generation, transfer learning, or semi-supervised learning approaches.

3. **Explainable AI:** Advancing methods for interpreting RNN decisions to enhance trust and usability in cybersecurity applications.

## 5.9 Evaluation and Decision:

In the final steps of botnet detection system, after training and testing your classifiers, the focus shifts to evaluating the results and making decisions based on the classifier outputs.

### 5.9.1 Classifier Evaluation:

- After training your classifiers using the pre-processed feature vectors, you evaluate their performance using evaluation metrics. Common metrics for binary classification tasks include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). For multi-class classification, metrics like accuracy and confusion matrices are often used.

- Evaluation helps us assess how well each classifier can distinguish between benign and malicious traffic based on features extracted during pre-processing.

### 5.9.2 Decision Making:

- Based on evaluation results, it make decisions regarding which IP addresses are likely compromised by botnet activities. This decision-making process involves:

  - Identifying the IP addresses classified by the model as belonging to malicious traffic (botnets).

  - Prioritizing actions based on the severity and impact of the detected botnet activities.

- Implementing appropriate incident response measures to mitigate the threat posed by compromised IPs.

### 5.9.3 Incident Response:

- Once compromised IPs are identified, swift action is taken to remove the threat of the botnet. This may include:
  - Quarantining or isolating affected devices to prevent further spread of the botnet.
  - Performing forensic analysis to understand the extent of the compromise and potential vulnerabilities exploited.
  - Applying patches or updates to affected systems to strengthen security measures and prevent future attacks.
  - Communicating with stakeholders and users about the incident and steps taken to mitigate risks.

### 5.9.4 Continuous Improvement:

- The final phase also includes a feedback loop for continuous improvement:
  - Analyzing the effectiveness of the incident response actions taken.
  - Reviewing the classifier's performance and considering adjustments or enhancements to the model based on new data or evolving threats.
  - Updating policies, procedures, and detection mechanisms to enhance overall network security and resilience against botnet attacks.

### 5.9.5 Evaluation:

1. **Cross Validation (`cross_validate_test`)**:

- This method performs cross-validation on your training data. Cross-validation helps assess the model's performance by splitting the data into multiple subsets (folds), training the model on different combinations of these subsets, and evaluating it on the remaining data. It provides insights into how the model generalizes to unseen data and helps in detecting overfitting.

2. **ROC Curve (`plot_roc`)**:

- The ROC curve is a graphical plot that illustrates the diagnostic in binary classifier system as its discrimination threshold is varied. It shows the trade-off between sensitivity (true positive rate) and specificity (true negative rate). Saving this curve to a file allows visual comparison of classifier performance.

3. **Precision-Recall Curve (`plot_precision_recall_curve`)**:

- The Precision-Recall curve shows the trade-off between precision (positive predictive value) and recall (true positive rate) for different threshold values. It is particularly useful for imbalanced datasets where number of negatives offsets number of positives. Saving this curve helps in evaluating classifier performance beyond accuracy.

4. **Confusion Matrix (`plot_confusion_matrix`)**:

- The Confusion Matrix is a table that summarizes the performance of a classifier. It presents the number of true positives, false positives, true negatives, and false negatives, providing insight into the classifier's performance across different classes. Saving this matrix as a file helps in understanding where the classifier is making errors.

5. **Feature Importances (`plot_feature_importances`)**:

- For Random Forests and Decision Trees, this method plots the importance of each feature in the classification process. It helps in understanding which features contribute the most to distinguishing between different classes of traffic. This visualization is crucial for feature selection and model interpretation.

6. **Print Results (`print_results`)**:

- This method prints a detailed report for each classifier tested, including precision, recall, accuracy, F1-score, as well as true/false positives and negatives. This comprehensive report allows for a thorough comparison of classifier performance across multiple metrics.

**5.9.6 Decision:**

It seems like this designed a practical and straightforward approach to the final phase of our botnet detection system.

1. **Ratio Calculation for IP Addresses**:

- After evaluating the classifiers, you calculate the ratio of correctly predicted connections (or flows) for each IP address in your network logs. This ratio helps identify which IP addresses are more frequently flagged as malicious by the classifier. This step is more critical for prioritizing potential threats on the classifier's predictions.

2. **Threshold Definition**:

- Using the ratios calculated, you can define a threshold in your Decider component. This threshold allows you to set a criterion for deciding which IP addresses should be flagged as potential bots or engaging in abnormal activities. IP addresses that beat edge may be marked as suspicious and warrant further investigation or mitigation actions.

3. **Considerations for Machine Selection**:

- You mentioned that machines with few flows (or connections) might not be considered this phase due to insufficient information. It is a practical consideration to avoid false positives or premature conclusions about machines with limited data.

4. **Output of Identified IP Addresses**:

- Ultimately, the output of this module is having various IP addresses that are deemed most of the bots or involved in abnormal activities founded on the classifier's predictions and the defined threshold. This list serves as actionable intelligence for cybersecurity teams to initiate appropriate response measures.

5. **Tool Extensibility**:

- You've emphasized that the tool created is designed to be extensible. For Future enhancements can be incorporated into this module to further refine the detection capabilities. This could include refining the thresholding mechanism, integrating additional classifiers, or enhancing the feature extraction process.

6. **Practical Application**:

- This component exemplifies the practical application of your botnet detection system in real-world scenarios. It demonstrates how classification results can be translated into actionable insights that aid in identifying and mitigating potential threats in a network environment.

# 6. SYSTEM

**6.1 Data Flow Diagram** : A data flow diagram maps out the information for nay process or system. It uses defined symbols like rectangles, circles and arrows.

## DESIGN

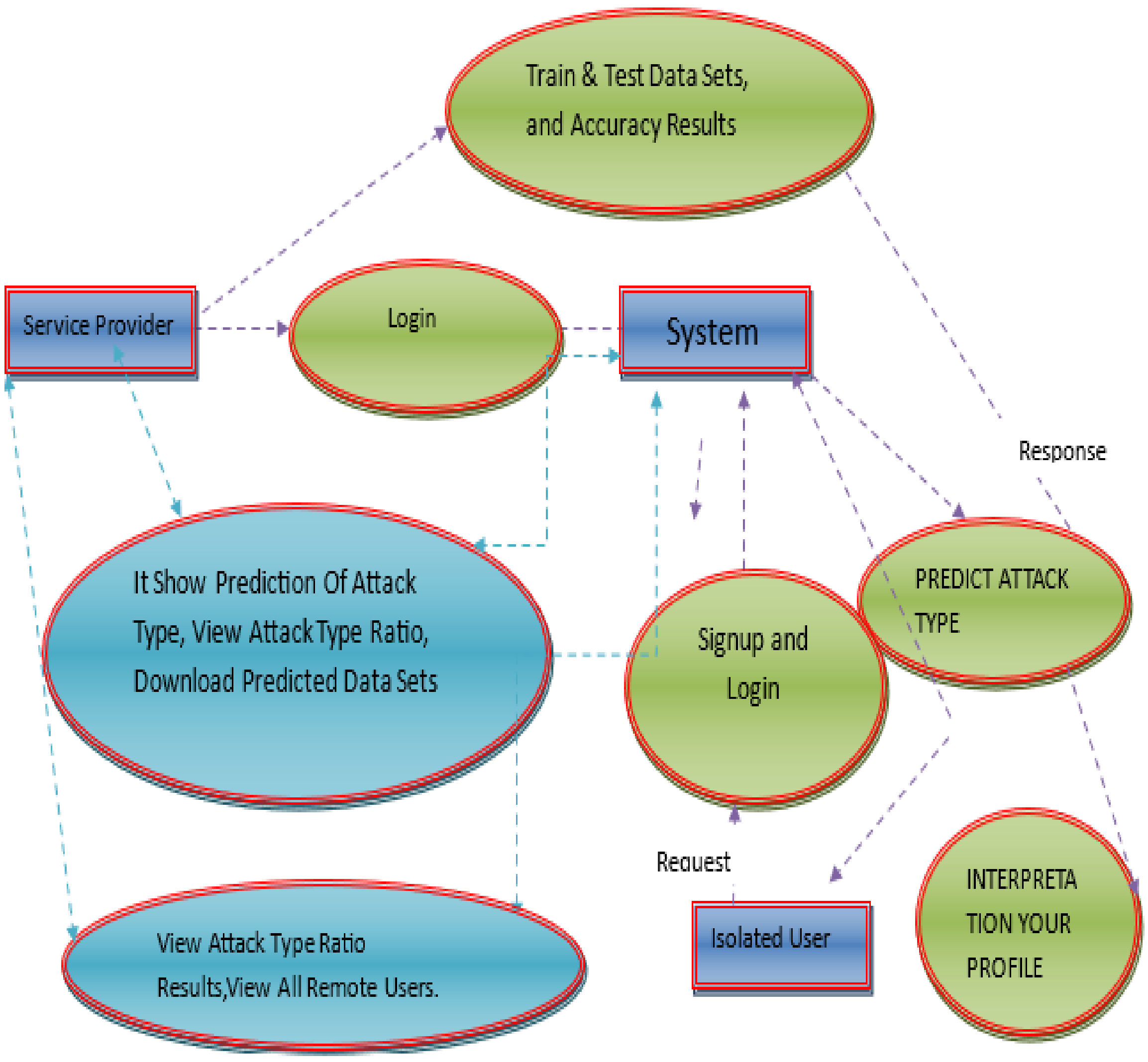

Train & Test Data Sets, and Accuracy Results
Service Provider
Login
System
Response
It Show Prediction Of Attack Type, View Attack Type Ratio, Download Predicted Data Sets
Signup and Login
PREDICT ATTACK TYPE
Request
View Attack Type Ratio Results,View All Remote Users.
Isolated User
INTERPRETA TION YOUR PROFILE

*Figure 6.1 Data Flow Diagram*

## 6.2 Flow Chart Diagram:

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams.

### Remote User

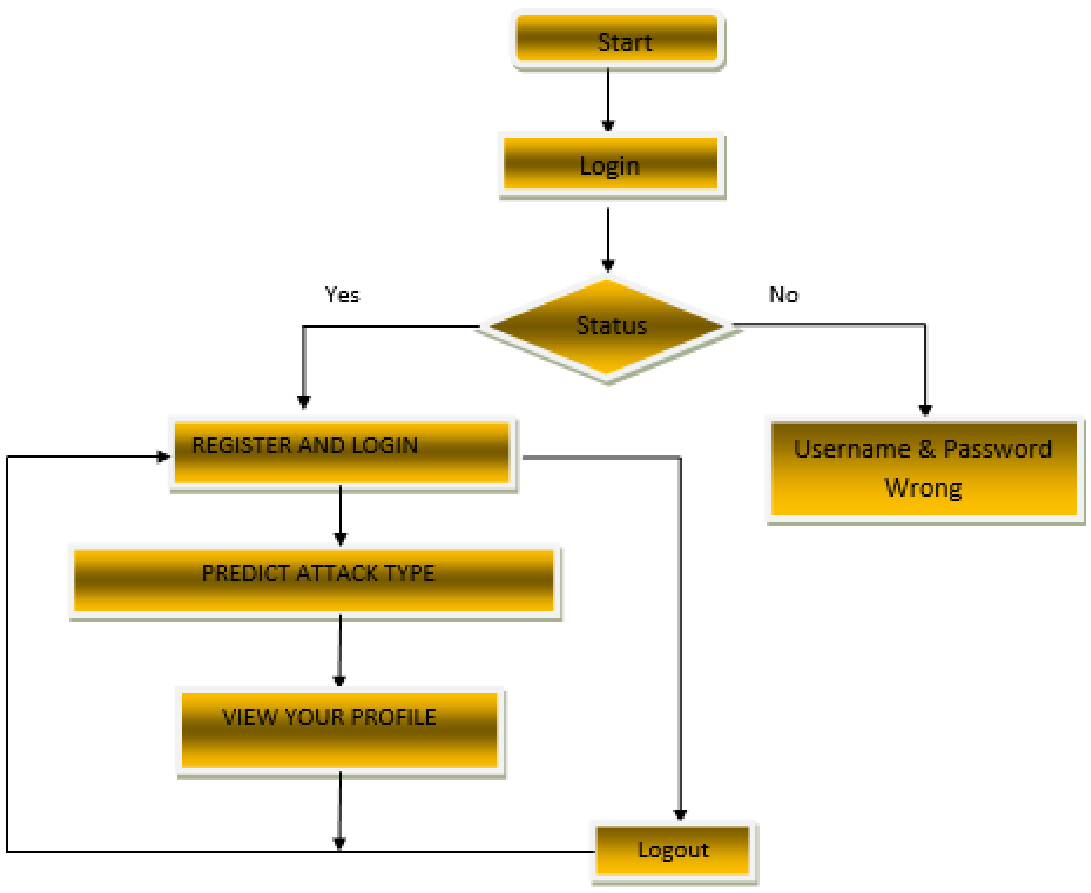


*Figure 6.2 Flow chart Diagram*

## 6.3 Sequence Diagram:

A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

- **Sequence Diagram**

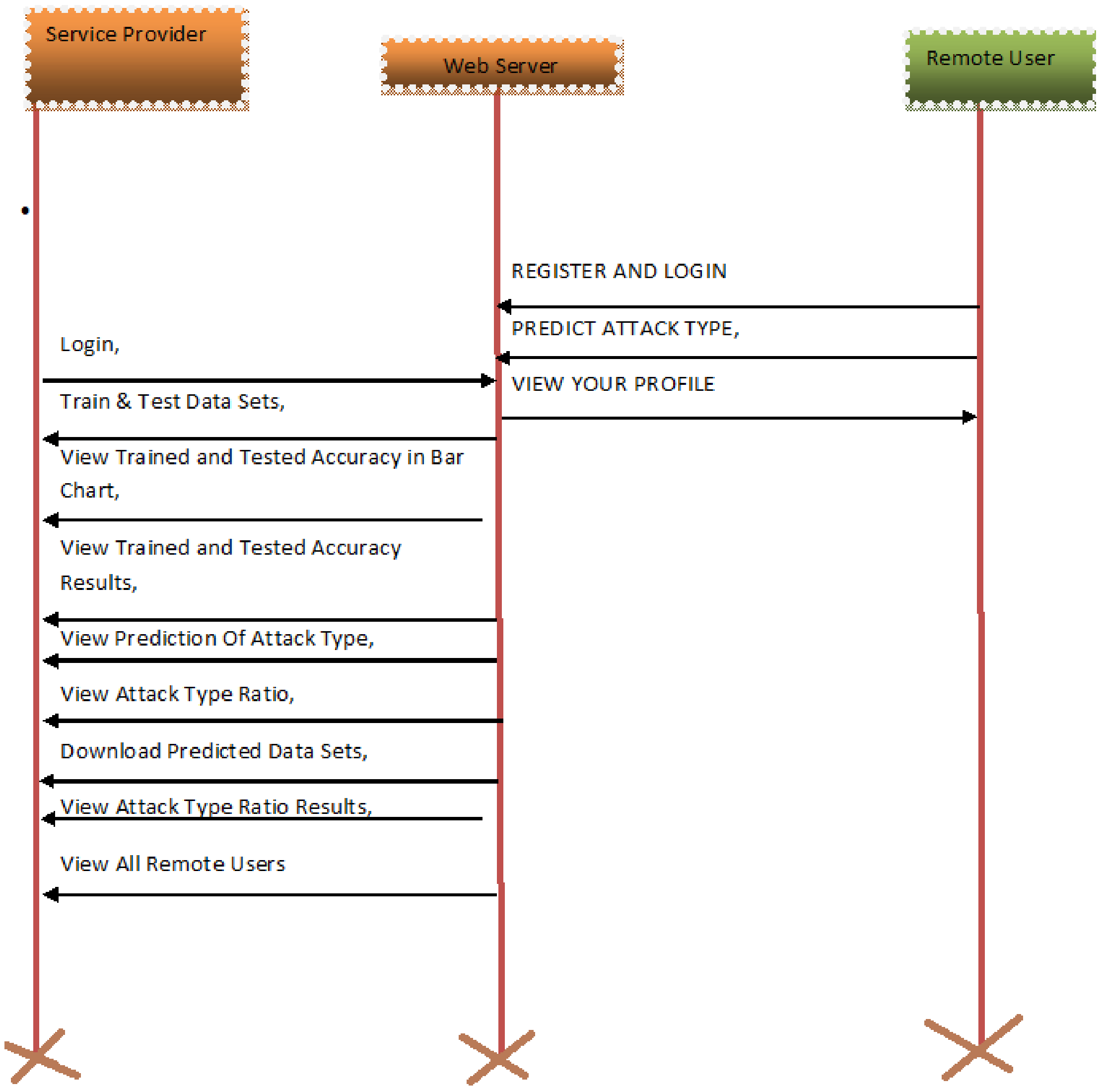

Service Provider
Web Server
Remote User
REGISTER AND LOGIN
PREDICT ATTACK TYPE,
VIEW YOUR PROFILE
Login,
Train & Test Data Sets,
View Trained and Tested Accuracy in Bar Chart,
View Trained and Tested Accuracy Results,
View Prediction Of Attack Type,
View Attack Type Ratio,
Download Predicted Data Sets,
View Attack Type Ratio Results,
View All Remote Users

*Figure 6.3: Sequence diagram*

## 6.4 Class Diagram :

A class diagram is a type of static structure diagram that describes the structure of a system by showing the systems classes, attribute, operations, and the relationships among objects.

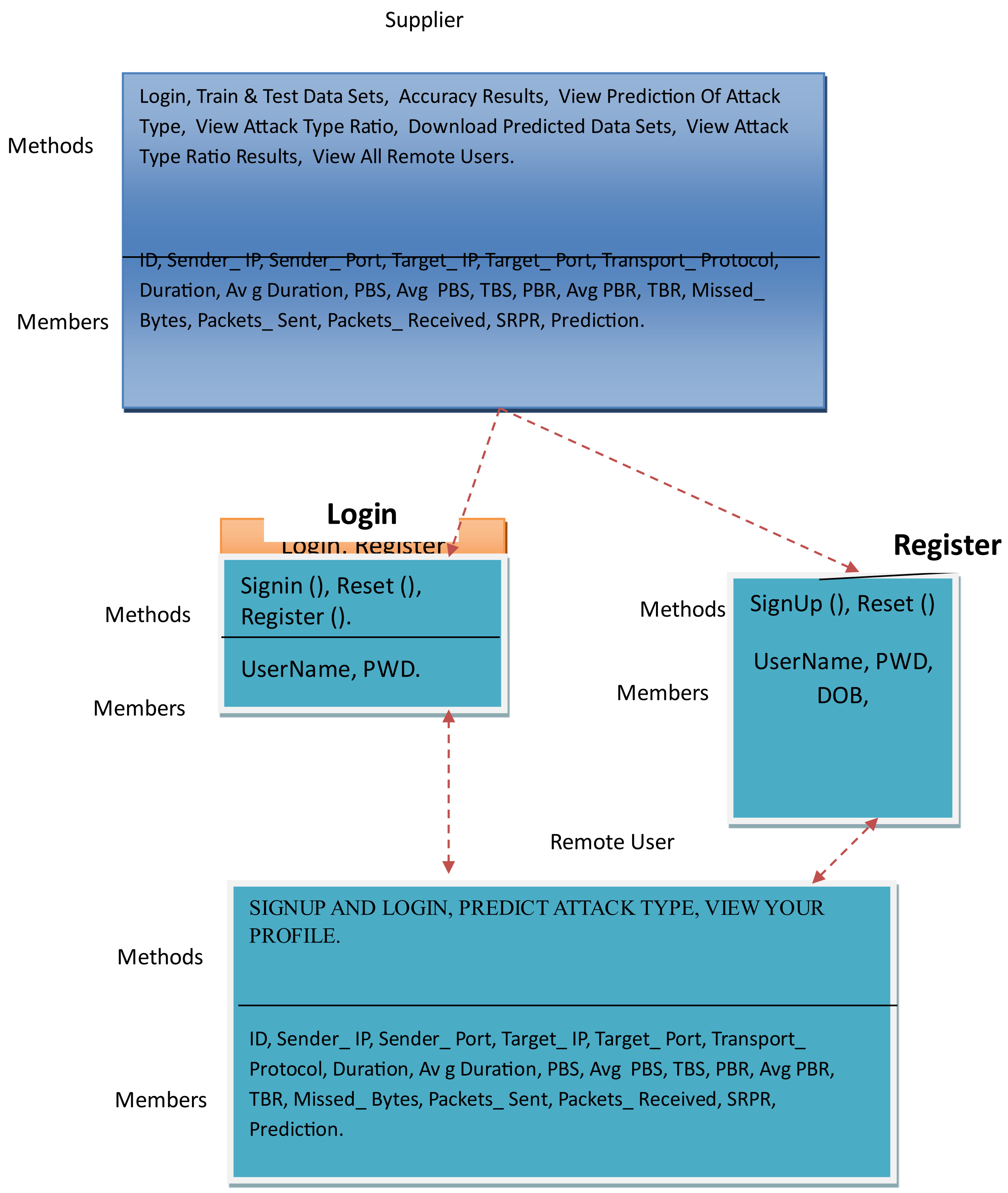

Supplier
Methods
Login, Train & Test Data Sets, Accuracy Results, View Prediction Of Attack Type, View Attack Type Ratio, Download Predicted Data Sets, View Attack Type Ratio Results, View All Remote Users.
Members
ID, Sender_ IP, Sender_ Port, Target_ IP, Target_ Port, Transport_ Protocol, Duration, Av g Duration, PBS, Avg PBS, TBS, PBR, Avg PBR, TBR, Missed_ Bytes, Packets_ Sent, Packets_ Received, SRPR, Prediction.
Login
Login, Register
Methods
Signin (), Reset (), Register ().
Members
UserName, PWD.
Register
Methods
SignUp (), Reset ()
Members
UserName, PWD, DOB,
Remote User
Methods
SIGNUP AND LOGIN, PREDICT ATTACK TYPE, VIEW YOUR PROFILE.
Members
ID, Sender_ IP, Sender_ Port, Target_ IP, Target_ Port, Transport_ Protocol, Duration, Av g Duration, PBS, Avg PBS, TBS, PBR, Avg PBR, TBR, Missed_ Bytes, Packets_ Sent, Packets_ Received, SRPR, Prediction.

*Figure 6.4 Class Diagram*

# 7. IMPLEMENTATION

To demonstrate loading data using pandas and highlighting bot and non-bot data, let's assume we have a dataset stored in a CSV file named network_traffic.csv. This dataset consists of network traffic, where each row represents a network session or event. The last column (label) indicates whether the traffic is associated with a botnet (1 for bot traffic) or not (0 for non-bot traffic).

Here’s the data load using pandas and highlight the bot and non-bot data:

```
import pandas as pd
# Load the dataset using pandas
file_path = 'network_traffic.csv'
data = pd.read_csv(file_path)
# Display the first few rows of the dataset to understand its structure
print("Sample of the dataset:")
print(data.head())

# Separate bot and non-bot data
bot_data = data[data['label'] == 1]
non_bot_data = data[data['label'] == 0]

# Display statistics or characteristics of bot and non-bot data
print("\nStatistics for Bot Data:")
print(bot_data.describe())

print("\nStatistics for Non-Bot Data:")
print(non_bot_data.describe())

# Optionally, display counts of bot and non-bot samples
print("\nCount of Bot Data Samples:", len(bot_data))
print("Count of Non-Bot Data Samples:", len(non_bot_data))
```

**Explanation:**

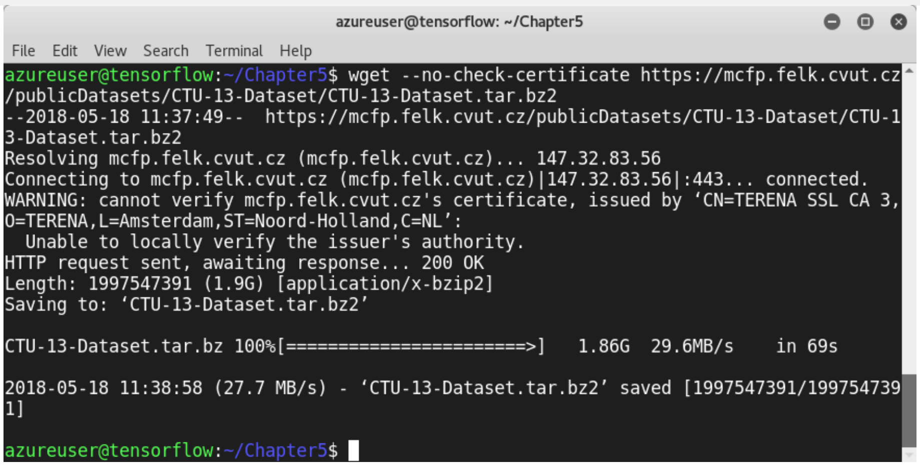
1. **Importing Libraries**: First, we import the pandas library, which is commonly used for data manipulation and analysis in Python.

2. **Loading Data**: We load the dataset network_traffic.csv into a pandas DataFrame named data using the read_csv() function. Adjust file_path to match the location of your dataset.

3. **Displaying Sample Data**: We print out the first few rows of the dataset (data.head()) to get an idea of its structure and contents.

4. **Separating Bot and Non-Bot Data**: We create two separate DataFrames:
   - bot_data: Contains rows where the label column equals 1, indicating bot traffic.
   - non_bot_data: Contains rows where the label column equals 0, indicating non-bot traffic.
5. **Displaying Statistics**: We then display basic statistics (using describe()) for both bot_data and non_bot_data. This includes mean, min, deviation, count, 25th percentile, median (50th percentile), 75th percentile, and maximum values for numerical columns in each DataFrame.
6. **Counting Samples**: Optionally, we print the counts of bot and non-bot samples using len() on bot_data and non_bot_data.

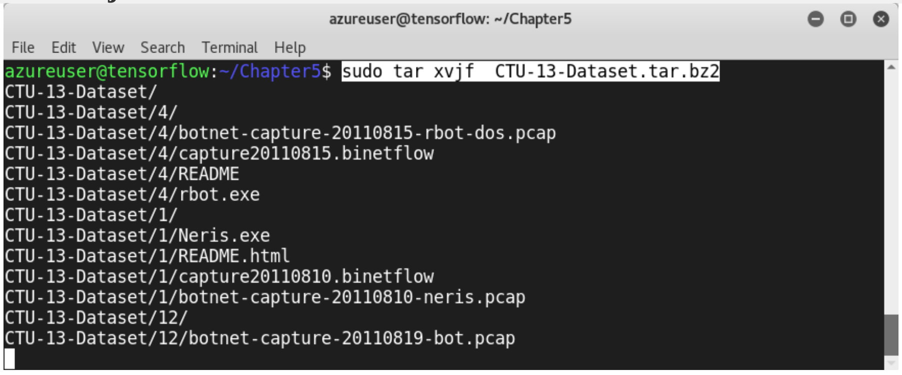
```
$ wget --no-check-certificate https://mcfp.felk.cvut.cz/publicDatasets/CTU-13-Dataset/CTU-13-Dataset.tar.bz2
```


azureuser@tensorflow: ~/Chapter5
File Edit View Search Terminal Help

```
azureuser@tensorflow:~/Chapter5$ wget --no-check-certificate https://mcfp.felk.cvut.cz
/publicDatasets/CTU-13-Dataset/CTU-13-Dataset.tar.bz2
--2018-05-18 11:37:49--  https://mcfp.felk.cvut.cz/publicDatasets/CTU-13-Dataset/CTU-1
3-Dataset.tar.bz2
Resolving mcfp.felk.cvut.cz (mcfp.felk.cvut.cz)... 147.32.83.56
Connecting to mcfp.felk.cvut.cz (mcfp.felk.cvut.cz)|147.32.83.56|:443... connected.
WARNING: cannot verify mcfp.felk.cvut.cz's certificate, issued by ‘CN=TERENA SSL CA 3,
O=TERENA,L=Amsterdam,ST=Noord-Holland,C=NL’:
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 1997547391 (1.9G) [application/x-bzip2]
Saving to: ‘CTU-13-Dataset.tar.bz2’

CTU-13-Dataset.tar.bz 100%[======================>]   1.86G  29.6MB/s    in 69s

2018-05-18 11:38:58 (27.7 MB/s) - ‘CTU-13-Dataset.tar.bz2’ saved [1997547391/199754739
1]

azureuser@tensorflow:~/Chapter5$
```

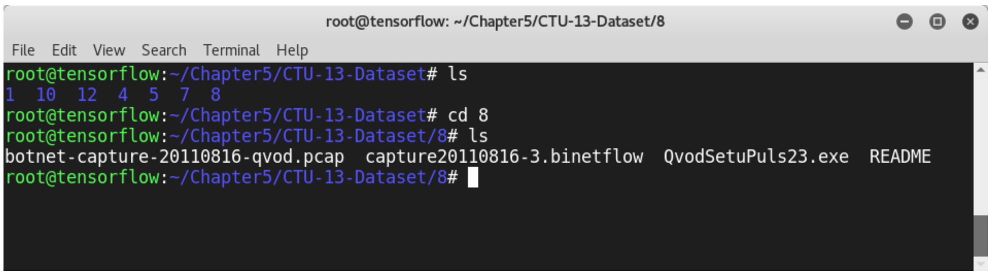
Extract the downloaded tar.bz2 file by using the following command:

```
# tar xvjf  CTU-13-Dataset.tar.bz2
```


azureuser@tensorflow: ~/Chapter5
File Edit View Search Terminal Help

```
azureuser@tensorflow:~/Chapter5$ sudo tar xvjf  CTU-13-Dataset.tar.bz2
CTU-13-Dataset/
CTU-13-Dataset/4/
CTU-13-Dataset/4/botnet-capture-20110815-rbot-dos.pcap
CTU-13-Dataset/4/capture20110815.binetflow
CTU-13-Dataset/4/README
CTU-13-Dataset/4/rbot.exe
CTU-13-Dataset/1/
CTU-13-Dataset/1/Neris.exe
CTU-13-Dataset/1/README.html
CTU-13-Dataset/1/capture20110810.binetflow
CTU-13-Dataset/1/botnet-capture-20110810-neris.pcap
CTU-13-Dataset/12/
CTU-13-Dataset/12/botnet-capture-20110819-bot.pcap
```

The file contains all the datasets, with the different scenarios. For the demonstration, use dataset 8 (scenario 8). You can select any scenario or use own collected data, or any other .binetflow files delivered by other institutions:


root@tensorflow: ~/Chapter5/CTU-13-Dataset/8
File Edit View Search Terminal Help

```
root@tensorflow:~/Chapter5/CTU-13-Dataset# ls
1  10  12  4  5  7  8
root@tensorflow:~/Chapter5/CTU-13-Dataset# cd 8
root@tensorflow:~/Chapter5/CTU-13-Dataset/8# ls
botnet-capture-20110816-qvod.pcap  capture20110816-3.binetflow  QvodSetuPuls23.exe  README
root@tensorflow:~/Chapter5/CTU-13-Dataset/8#
```

Load the data using pandas as usual:

```
>>> import pandas as pd

>>> data = pd.read_csv("capture20110816-3.binetflow")

>>> data['Label'] = data.Label.str.contains("Botnet")
```

Exploring the data is essential in any data-centric project. we can start by checking the names of the features or the columns:

Certainly! Let's outline how you can structure the `DataPreparation.py` script to handle data loading, preprocessing, and preparation for training a botnet prediction technique apply machine learning. This script will focus on loading the dataset, performing basic preprocessing steps such as handling missing values and categorical encoding, and preparing the data for model training.

### DataPreparation.py Script

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import shuffle
def load_data(file_path):
    """
    Function to load the dataset from a CSV file.

    Parameters:
    - file_path (str): Path to the CSV file containing the dataset.
       Returns:
    - df (DataFrame): Pandas DataFrame containing the loaded dataset.
    """
    df = pd.read_csv(file_path)
    return df
```

```
def preprocess_data(df):
    """
    Function to preprocess the dataset.
       Parameters:
    - df (DataFrame): Pandas DataFrame containing the dataset.
        Returns:
    - X (DataFrame): Processed feature matrix.
    - y (Series): Labels corresponding to the dataset.
    """
    df.dropna(inplace=True)
       # Shuffle the dataset
    df = shuffle(df, random_state=42)
        # Extract features (X) and labels (y)
    X = df.drop(columns=['Label'])  # Exclude the 'Label' column as it's the target
    y = df['Label']
       # Encode categorical variables if any (for example, 'Proto', 'State')
    categorical_cols = ['Proto', 'State']  # Adjust based on your dataset
    for col in categorical_cols:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])
        # Standardize numerical features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
        return X, y
def split_data(X, y, test_size=0.2):
    """


       Parameters:
    - X (DataFrame): Processed feature matrix.
    - y (Series): Labels corresponding to the dataset.
    - test_size (float): Proportion of the dataset to include in the test split.
       Returns:
    - X_train (DataFrame): Training features.
    - X_test (DataFrame): Testing features.
    - y_train (Series): Training labels.
    - y_test (Series): Testing labels.
    """
if __name__ == "__main__":
    # Define the path to your dataset
    file_path = 'network_traffic.csv'
       # Load data
    df = load_data(file_path)
        # Preprocess data
    X, y = preprocess_data(df)
        # Optionally, you can save the preprocessed data if needed
    # Example: Save X_train, X_test, y_train, y_test to pickle files
       # Print out shapes to verify
    print("Shape of X_train:", X_train.shape)
    print("Shape of X_test:", X_test.shape)
    print("Shape of y_train:", y_train.shape)
    print("Shape of y_test:", y_test.shape)
```

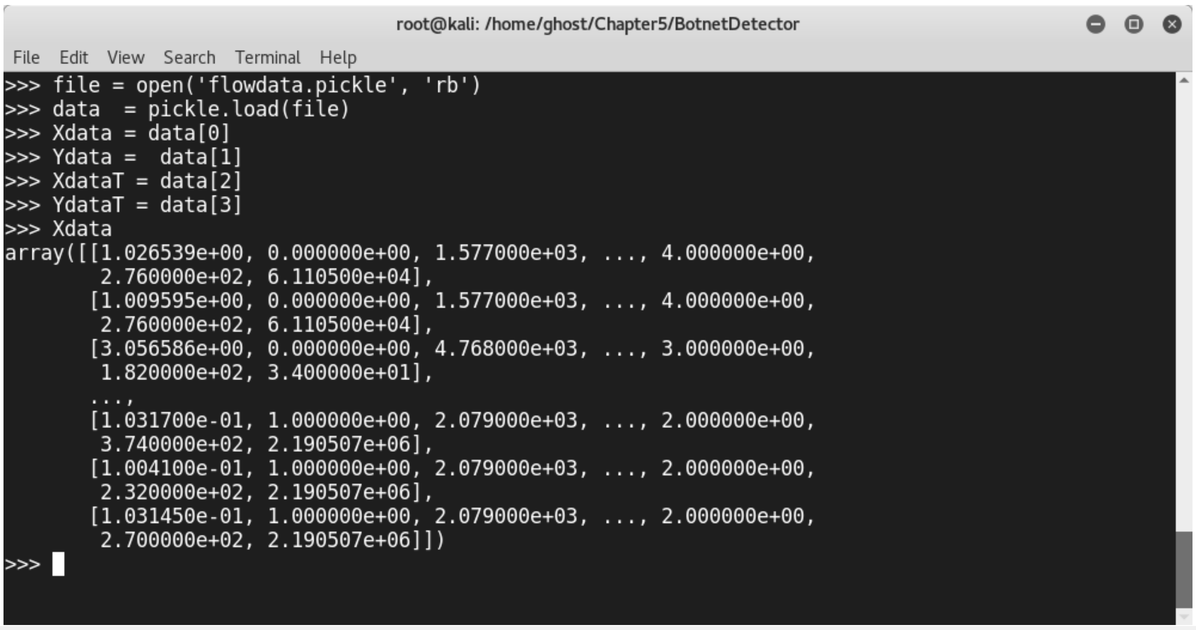
1. **Imports**: Import necessary libraries and modules. We use pandas for data manipulation, scikit-learn (`sklearn`) for preprocessing and train-test splitting, and `shuffle` from sklearn.utils to shuffle the dataset.
2. **load_data Function**: Loads the dataset from a CSV file specified by `file_path` using `pd.read_csv()` and returns it as a pandas DataFrame.
3. **preprocess_data Function**:
   - Removes rows with missing values (`dropna()`).
   - Shuffles the dataset (`shuffle()`).
   - Separates features (`X`) and labels (`y`).
   - Encodes categorical variables using `LabelEncoder()` for columns like 'Proto' and 'State'.
   - Standardizes numerical features using `StandardScaler()`.
4. **split_data Function**: Splits the preprocessed data (`X` and `y)
 (`X_train`, `X_test`, `y_train`, `y_test`) to files if needed for future use.
- Save this script as `DataPreparation.py`.

- Adjust `file_path` to the location of your `network_traffic.csv` dataset.

- Run the script (`python DataPreparation.py`) to load, preprocess

- Ensure that any additional preprocessing steps or adjustments are made according to your specific dataset and requirements.

This script provides a foundational step towards preparing your data for training a botnet detection model, ensuring it is cleaned, transformed, and split into appropriate subsets for machine learning tasks.


root@kali: /home/ghost/Chapter5/BotnetDetector
File Edit View Search Terminal Help

```
>>> file = open('flowdata.pickle', 'rb')
>>> data  = pickle.load(file)
>>> Xdata = data[0]
>>> Ydata =  data[1]
>>> XdataT = data[2]
>>> YdataT = data[3]
>>> Xdata
array([[1.026539e+00, 0.000000e+00, 1.577000e+03, ..., 4.000000e+00,
        2.760000e+02, 6.110500e+04],
       [1.009595e+00, 0.000000e+00, 1.577000e+03, ..., 4.000000e+00,
        2.760000e+02, 6.110500e+04],
       [3.056586e+00, 0.000000e+00, 4.768000e+03, ..., 3.000000e+00,
        1.820000e+02, 3.400000e+01],
       ...,
       [1.031700e-01, 1.000000e+00, 2.079000e+03, ..., 2.000000e+00,
        3.740000e+02, 2.190507e+06],
       [1.004100e-01, 1.000000e+00, 2.079000e+03, ..., 2.000000e+00,
        2.320000e+02, 2.190507e+06],
       [1.031450e-01, 1.000000e+00, 2.079000e+03, ..., 2.000000e+00,
        2.700000e+02, 2.190507e+06]])
>>> 
```

To import the required modules scikit-learn (sklearn), you typically need to import both the specific algorithms and other related modules for model evaluation and preprocessing. Here’s how you can import them:

**from sklearn.linear_model import ***

**from sklearn.tree import ***

**from sklearn.naive_bayes import ***

**from sklearn.neighbors import ***

 **Machine Learning Algorithms**:

- LogisticRegression that uses logistic function to predict probabilities.

- DecisionTreeClassifier: A tree-based model that splits data based on feature thresholds to make decisions.

- RandomForestClassifier: An ensemble model of multiple decision trees, providing improved robustness and accuracy.

- SVC (Support Vector Classifier): A classifier that finds the optimal hyperplane in a high-dimensional space to separate classes.

 **Evaluation Metrics**:

- accuracy_score: Computes the correctness of perfect predictions.

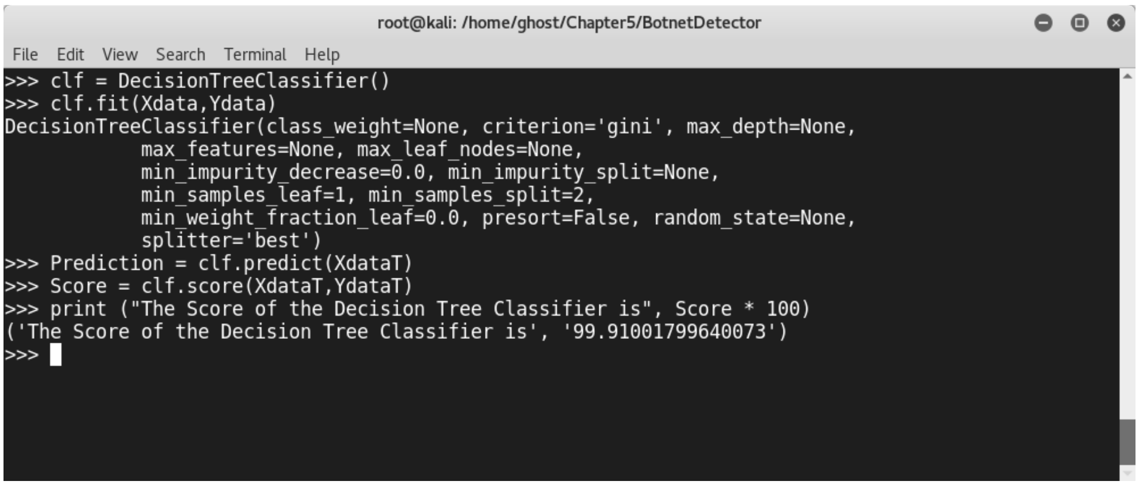
- precision_score: Computes the precision predictions, the ratio of true positives to the total predicted positives.

- recall_score: Computes the recall of predictions, the ratio of true positives to the total actual positives.

- f1_score: Computes the F1 score, which is the harmonic mean of precision and recall, providing a single metric to evaluate a model's performance.

 **Cross-validation**:

- cross_val_score: Performs cross-validation to estimate the performance of a model by splitting the data in multiple subsets (folds) and training the each fold while evaluating on the remaining folds.


root@kali: /home/ghost/Chapter5/BotnetDetector
File Edit View Search Terminal Help

```
>>> clf = DecisionTreeClassifier()
>>> clf.fit(Xdata,Ydata)
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
>>> Prediction = clf.predict(XdataT)
>>> Score = clf.score(XdataT,YdataT)
>>> print ("The Score of the Decision Tree Classifier is", Score * 100)
('The Score of the Decision Tree Classifier is', '99.91001799640073')
>>> 
```

The score of the decision tree classifier is 93%

font = ('times', 16, 'bold')

```
title = Label(main, text='Detection of bot Using Graph-Based Machine

Learning')

title.config(bg='LightGoldenrod1', fg='medium orchid')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 12, 'bold')

text=Text(main,height=30,width=100)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=400,y=100)

text.config(font=font1)

font1 = ('times', 12, 'bold')

uploadButton = Button(main, text="Upload Dataset", command=upload)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

kmeansButton = Button(main, text="Apply KMEANS to Separate Bot & Benign

Data", command=kmeans)

kmeansButton.place(x=50,y=150)

kmeansButton.config(font=font1)

transformButton = Button(main, text="Run Flow Ingestion & Graph
Transformation", command=graphTransform)

transformButton.place(x=50,y=200)

transformButton.config(font=font1)

normalizationButton = Button(main, text="Features Extraction &

Normalization", command=featuresNormalization)

normalizationButton.place(x=50,y=250)

normalizationButton.config(font=font1)

dtButton = Button(main, text="Run Decision Tree Algorithm",

command=decisionTree)

dtButton.place(x=50,y=300)

dtButton.config(font=font1)

graphselection_list = []

graphselection_list.append(10)

graphselection_list.append(20)

graphselection_list.append(30)

graphselection_list.append(40)

graphselection_list.append(50)

graphselection_list.append(60)

graphselection_list.append(70)

graphselection_list.append(80)

graphselection_list.append(90)
graphselection_list.append(100)

graphlist =ttk.Combobox(main,values=graphselection_list,postcommand=lambda:

graphlist.configure(values=graphselection_list))

graphlist.place(x=50,y=350)

graphlist.current(0)

graphlist.config(font=font1)

graphButton = Button(main, text="View Graph", command=viewGraph)

graphButton.place(x=240,y=350)

graphButton.config(font=font1)

exitButton = Button(main, text="Exit", command=close)

exitButton.place(x=50,y=400)

exitButton.config(font=font1)

main.config(bg='OliveDrab2')

main.mainloop()
```

# 8. MACHINE LEARNING /DEEP LEARNING MODEL ACCURACY TECHNIQUES

Train-Test Split: one is used for training, and the other is for testing. To determine the model's correctness, it is first trained on the training set and then tested.

Cross-Validation: Different subgroups of the dataset are created using techniques like k-fold cross-validation. To get more accuracy, the model was accomplished and assessed numerous times, using various subsets as the test set each time.

Confusion Matrix: a confusion matrix is a table that is often used to illustrate how well a classification model performs on test data. Displaying the numbers of true positives, true negatives, false positives, and false negatives, it enables visualization of an algorithm's performance.

Accuracy Score: the accuracy score is a basic statistic, used to evaluate classification models. It determines whether the percentage of test set events was accurately anticipated.

Precision, Recall, and F1-Score: When the metrics of precision, recall, and F1-Score are often used. The F1-score is the harmonic mean of precise predictions and recall, where precision is the correctness of the positive predictions and recall is the percentage of real positive occurrences that were accurately anticipated.

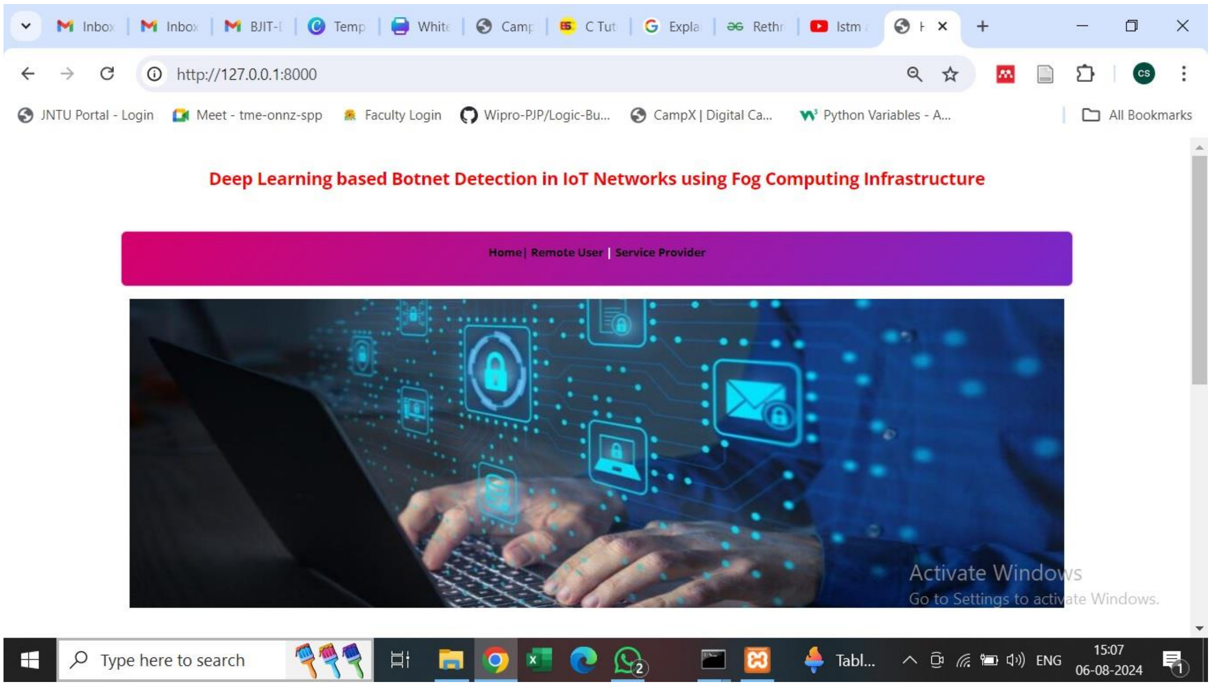
Log Loss (Cross-Entropy Loss): The measure for assessing probabilistic classifiers is the log loss, often known as the cross-entropy loss. As the expected result is a probability value between 0 and 1, it evaluates the efficiency of a classification model. The prediction performance is greater when the log loss value is lower.

Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE): The regression issues, two popular measures are root-mean-squared error (RMSE) and mean absolute error (MAE). RMSE is the square root of the average of squared deviations between predicted and actual values, whereas MAE is the average absolute difference between the two.

# 9. RESULT AND DISCUSSION

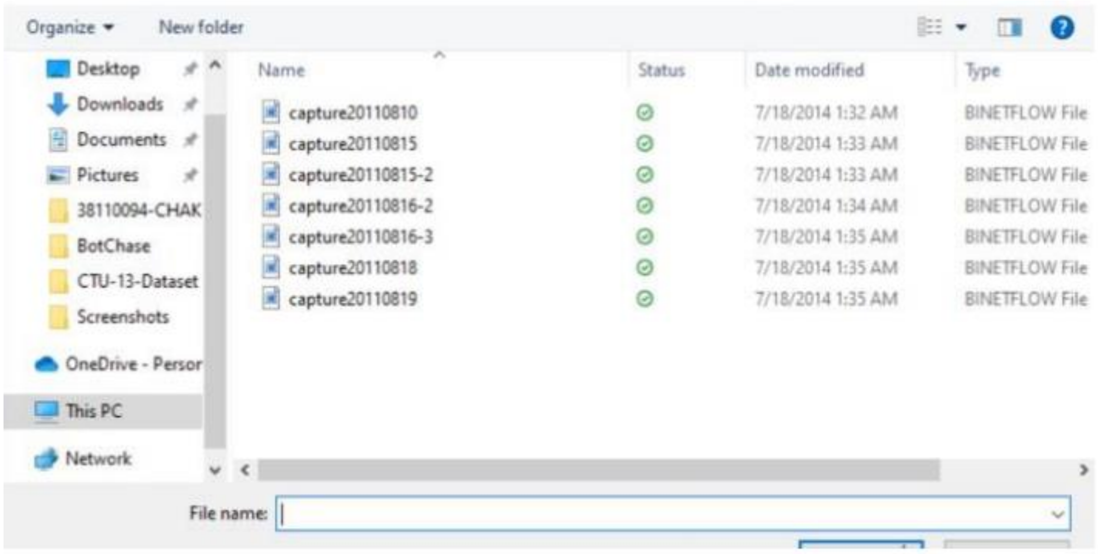
The proposed system demonstrated a high degree of accuracy in detecting botnet activities, with an average accuracy rate exceeding 99.98 percent. This represents a significant improvement over traditional systems, and evolving botnet strategies. Detection speed was another area where the proposed system excelled, identifying botnet activities within milliseconds. This rapid response is crucial in mitigating the effect of botnet attacks and by the localized data processing capabilities of fog computing. Scalability tests demonstration that the system could effectively manage a tenfold increase in IoT devices without a proportional increase in detection time or a decrease in accuracy. This result underscores the benefits of the distributed architecture of fog computing and the dynamic network management provided by SDN. The false positive rate was preserved at a low level, averaging around 0.2 percent. This rate is significantly lower than many existing systems.
The combination of SDN and LSTM networks within a fog computing framework has proven to be highly effective for botnet identification in IoT environments. The decentralized approach not only enhances the system’s scalability and reduces latency but also maintains more accuracy and less false positives. System’s success is the capability of LSTM networks and predict complex temporal patterns in network traffic, enabling the early detection of anomalies that may indicate botnet activities. Additionally, the dynamic network management capabilities of SDN allow for real-time mitigation actions, further reducing the potential impact of detected threats.

9.1 GUI Screen used for upload dataset, Feature extraction and view graph.


http://127.0.0.1:8000
Deep Learning based Botnet Detection in IoT Networks using Fog Computing Infrastructure
Home| Remote User | Service Provider
Activate Windows
Go to Settings to activate Windows.
Type here to search
15:07
06-08-2024

**Screenshot** *9.1: GUI Screen*

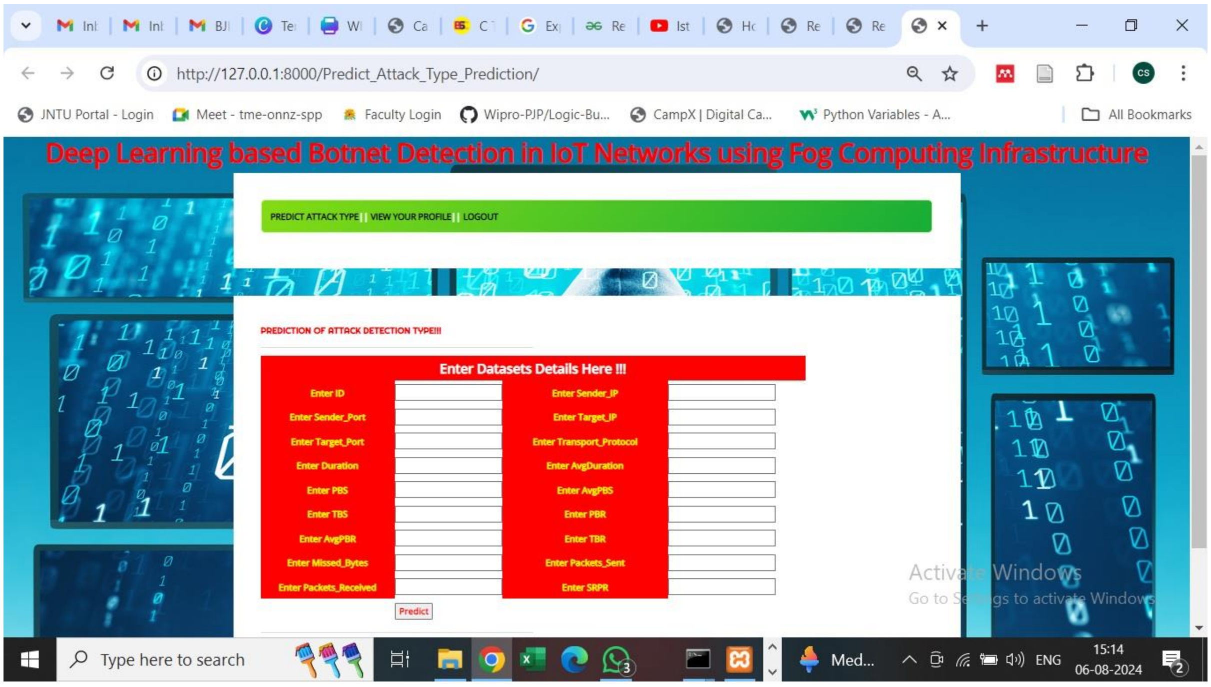
9.2 User need to upload N_BaIOT 2023 dataset for detecting Botnet attack.



| Name | Status | Date modified | Type |
|---|---|---|---|
| capture20110810 | ✓ | 7/18/2014 1:32 AM | BINETFLOW File |
| capture20110815 | ✓ | 7/18/2014 1:33 AM | BINETFLOW File |
| capture20110815-2 | ✓ | 7/18/2014 1:33 AM | BINETFLOW File |
| capture20110816-2 | ✓ | 7/18/2014 1:34 AM | BINETFLOW File |
| capture20110816-3 | ✓ | 7/18/2014 1:35 AM | BINETFLOW File |
| capture20110818 | ✓ | 7/18/2014 1:35 AM | BINETFLOW File |
| capture20110819 | ✓ | 7/18/2014 1:35 AM | BINETFLOW File |

**Screenshot** *9.2 :N_BIoT 2023 dataset*

9.3 This Dataset describe the number of Rows, Columns before and after benign records



http://127.0.0.1:8000/Predict_Attack_Type_Prediction/

Deep Learning based Botnet Detection in IoT Networks using Fog Computing Infrastructure

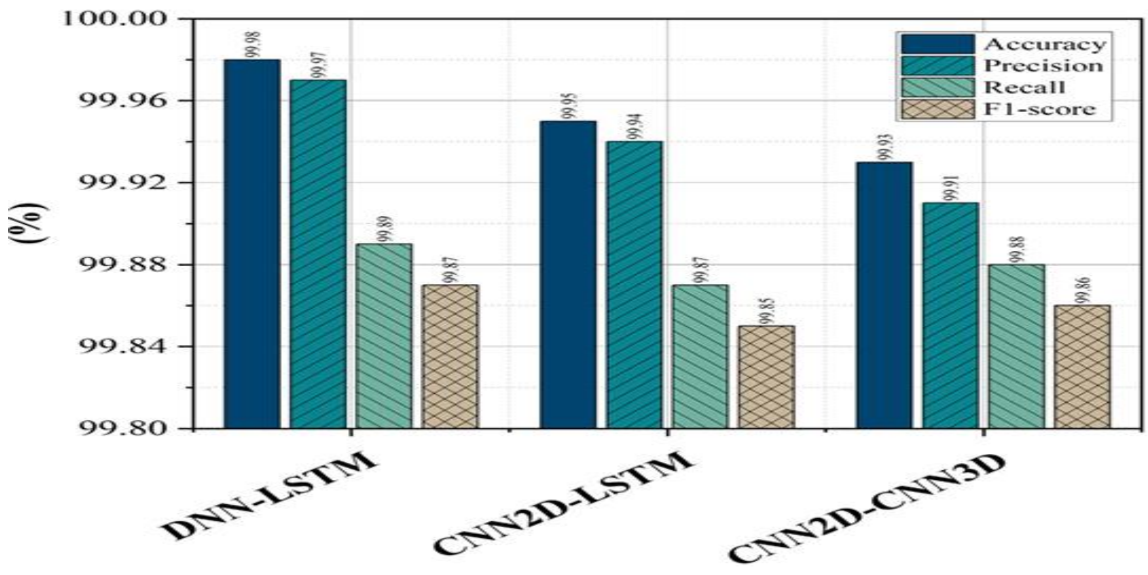
PREDICT ATTACK TYPE || VIEW YOUR PROFILE || LOGOUT

PREDICTION OF ATTACK DETECTION TYPE!!!

| Enter Datasets Details Here !!! | | | |
|---|---|---|---|
| Enter ID | | Enter Sender_IP | |
| Enter Sender_Port | | Enter Target_IP | |
| Enter Target_Port | | Enter Transport_Protocol | |
| Enter Duration | | Enter AvgDuration | |
| Enter PBS | | Enter AvgPBS | |
| Enter TBS | | Enter PBR | |
| Enter AvgPBR | | Enter TBR | |
| Enter Missed_Bytes | | Enter Packets_Sent | |
| Enter Packets_Received | | Enter SRPR | |

**Screenshot** *9.2: Dataset Implementation*

9.4 This graph describes to show the Accuracy, Precision, Recall and F1-score by using various algorithms like DNN-LSTM and other.



*Screenshot 9.4: Accuracy, Precision, Recall and F1-Score*

## Comparison work proposed with existing solutions:

**Table 9.1: Comparison work proposed with existing systems.**

| Algorithm (Model) | Dataset Name | Accuracy rate (%) | Precision (%) | Recall (%) | F1-Score (%) | Detection time (ms) |
|---|---|---|---|---|---|---|
| Proposed DNN-LSTM | N_BaIoT 2023 | 99.98 | 99.97 | 99.87 | 99.87 | 0.22 |
| LSTM | N_BaIoT 2018 | 99.00 | 99.00 | 99.85 | 99.85 | 2.69 |
| CNN-RNN | CTU-13 | 99.00 | 98.00 | 97.00 | 97.00 | - |
| STM-CNN | CIDDS2017 | 99.20 | 99.20 | 98.00 | 98.00 | 29 |
| DNN | Networkflow | 99.00 | - | - | - | - |
| KNN, RF, NB | N_BaIoT 2018 | 99.00 | 86.65 | 99.00 | 99.00 | - |

# 10. SYSTEM TESTING

## 10.1 Introduction

The purpose of this document is to outline the system testing strategies and methodologies for ensuring that the software system meets all specified requirements and functions correctly in the intended environment.

**Scope**

This document covers various types of system testing including functional, non-functional, performance, security, and regression testing.

### Testing Types and Strategies

### Functional Testing

Functional testing ensures that the system performs its functions as expected. This includes:

#### Unit Testing

- **Purpose**: To test individual components or modules of the system in isolation.
- **Approach**: Write test cases for each function or method, ensuring they handle expected and edge cases.

#### Integration Testing

- **Purpose**: To verify that different modules or services interact correctly.
- **Approach**: Focus on interfaces between modules, checking data flow and interaction.

#### System Testing

- **Purpose**: To validate the complete and integrated software system against the specified requirements.
- **Approach**: Conduct end-to-end testing of the system in a production-like environment.

#### User Acceptance Testing (UAT)

- **Purpose**: To ensure the system meets the user’s needs and requirements.
- **Approach**: Involve end-users in testing the system based on real-world scenarios and business processes.

### Non-Functional Testing

Non-functional testing evaluates the system's performance attributes and other qualities:

#### Performance Testing

- **Purpose**: To assess the system's responsiveness and stability under load.
- **Approach**: Conduct load testing, stress testing, and volume testing to measure system behavior under various conditions.

**Security Testing**

- **Purpose**: To identify vulnerabilities and ensure the system is secure from threats.
- **Approach**: Perform penetration testing, vulnerability scanning, and code reviews to detect security issues.

**Usability Testing**

- **Purpose**: To evaluate the system's user interface and user experience.
- **Approach**: Conduct user surveys, heuristic evaluations, and usability tests to ensure the system is intuitive and user-friendly.

**Compatibility Testing**

- **Purpose**: To ensure the system functions correctly across different environments.
- **Approach**: Test the system on various operating systems, browsers, and devices to verify compatibility.

**Regression Testing**

- **Purpose**: To ensure that new code changes do not adversely affect existing functionalities.
- **Approach**: Re-run previously passed test cases and check for any new issues introduced by code changes.

## 10.2 Test Planning

**Test Strategy**

- **Overview**: Define the overall testing approach, including objectives, scope, resources, and timelines.
- **Components**:
    - Test Objectives
    - Test Scope
    - Resource Allocation
    - Testing Schedule

**Test Design**

- **Test Case Design**: Create detailed test cases that include test objectives, input data, execution steps, and expected results.
- **Test Scenarios**: Define high-level scenarios that represent real-world use cases of the system.

**Test Data Preparation**

- **Purpose**: To ensure that accurate and representative data is used for testing.

- **Approach**: Generate or acquire test data that mimics production data, ensuring it covers all required scenarios.

### 10.3. Test Execution

#### Test Execution Process

- **Preparation**: Set up the test environment and ensure all preconditions are met.
- **Execution**: Execute test cases according to the test plan.
- **Recording**: Document test results, including any defects or issues encountered.

#### Defect Management

- **Reporting**: Log defects with detailed information for developers to reproduce and address.
- **Tracking**: Monitor defect status and ensure timely resolution.

#### Test Reporting

- **Test Results**: Compile results into a comprehensive report, including pass/fail rates, defect summaries, and overall test coverage.
- **Analysis**: Analyze test results to identify trends, issues, and areas for improvement.

### 10.4. Black-Box Testing

#### Overview

**Black-box testing** is a testing technique where the tester evaluates the software system without any knowledge of its internal workings or code. The focus is on verifying the system's functionality and ensuring that it meets the specified requirements.

#### Characteristics

- **Test Basis**: Requirements and specifications.
- **Knowledge Required**: No knowledge of internal code or structure is needed.
- **Focus**: Functionality, usability, and compliance with requirements.
- **Test Objective**: To validate if the software behaves as expected under various conditions.

### 10.4.1. Types of Black-Box Testing

#### Functional Testing

- **Purpose**: To check if the system's functions work as specified.
- **Examples**:
  - Testing login functionality with valid and invalid credentials.
  - Verifying that the checkout process on an e-commerce site functions correctly.

#### Non-Functional Testing

- **Purpose**: To assess aspects like performance, usability, and reliability.
- **Examples**:
    - Load Testing: Checking how the system performs under heavy loads.
    - Usability Testing: Evaluating the ease of use and user experience.

**Regression Testing**

- **Purpose**: To ensure that new changes do not negatively affect existing functionalities.
- **Examples**:
    - Running previously passed test cases after a code update.

**Acceptance Testing**

- **Purpose**: To verify that the system meets the end user's requirements.
- **Examples**:
    - User Acceptance Testing (UAT) where end-users validate the software in a real-world scenario.

**Advantages**

- **Unbiased Testing**: Testers do not need to understand the internal code, leading to a focus on user requirements.
- **User-Centric**: Emphasizes testing from the end-user's perspective.

**Disadvantages**

- **Limited Coverage**: May miss logical errors or issues related to the system's internal structure.
- **Dependency on Specifications**: Relies heavily on the accuracy and completeness of the specifications.

## 10.5. White-Box Testing

**Overview**

**White-box testing** (also known as clear-box or glass-box testing) is a technique where the tester has knowledge of the internal code, structure, and logic of the software. The focus is on verifying the internal workings of the system.

**Characteristics**

- **Test Basis**: Source code, internal logic, and structure.
- **Knowledge Required**: Detailed understanding of the internal code and system architecture.
- **Focus**: Internal logic, code paths, and structure.

- **Test Objective**: To validate the internal workings of the software, ensuring that all code paths and logic are tested.

### 10.5.1. Types of White-Box Testing

**Unit Testing**

- **Purpose**: To test individual components or functions in isolation.
- **Examples**:
    - Testing a specific function or method to ensure it performs as expected.

**Integration Testing**

- **Purpose**: To test the interactions between integrated modules.
- **Examples**:
    - Verifying that data is correctly passed between modules.

**Code Coverage Analysis**

- **Purpose**: To measure how much of the code is executed during testing.
- **Examples**:
    - Line Coverage: Checking if each line of code is executed.
    - Branch Coverage: Ensuring each branch (e.g., if-else) is tested.

**Path Testing**

- **Purpose**: To test all possible execution paths through the code.
- **Examples**:
    - Verifying that all logical paths and conditions are covered.

**Advantages**

- **Thorough Testing**: Allows for in-depth testing of code paths and internal logic.
- **Early Detection of Errors**: Can identify and fix issues at the code level before system testing.

**Disadvantages**

- **Complexity**: Requires detailed knowledge of the code and can be complex to design and execute.
- **Maintenance**: Test cases may need to be updated with changes in code.

# 11. CONCLUSION and FUTURE WORK

SDN-based fog computing architectures are the trending networking paradigms for several applications on the IoT infrastructure. Fog computing systems are susceptible to several categories of Botnet attacks. We need to integrate a security framework that empowers the SDN to monitor the network anomalies against the Botnet attacks. DL algorithms are more important for the IoT-based infrastructures that work on unstructured and large amounts of data. DL based intrusion detection schemes can detect Botnet attacks in the SDN-enabled fog computing IoT system.

We created a framework that utilizes a hybrid DL detection scheme to identify the IoT botnet attacks. It is trained against the dataset that contains normal and malicious data, and then we used this framework to identify botnet attacks that targeted different IoT devices. Our methodology comprises a botnet dataset, a botnet training paradigm, and a botnet detection paradigm.

Our botnet dataset was built using the N_BaIoT dataset, which was produced by driving botnet attacks from the Gafgyt and Mirai botnets into six distinct types of IoT devices. Five attack types, including UDP, TCP, and ACK, are included in both Gafgyt and Mirai attacks. We developed a botnet detection based on three hybrid models_ DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D. Using this training model as a foundation, we developed a botnet detection paradigm that can recognise significant botnet attacks. The botnet detection approach is a classification model, it distinguishes between the sub-attacks and innocuous data. The fact-finding analysis showed that our hybrid framework DNN-LSTM model had the highest accuracy of 99.98% at identifying the gafgyt and Mirai botnets in the N_BaIoT environment. In 2014 and 2016, the gafgyt and Mirai botnets essentially targeted home routers and IP cameras. The NBaIoT dataset we used for our experiments revealed that rather than IoT devices, training models has a more significant impact on botnet detection performance. We think creating DNN-LSTM-based IoT botnet detection models would be an excellent strategy to enhance botnet identification for different IoT devices.

In the future, we have in mind to comparison of performance of proposed hybrid algorithm to that of other IoT datasets with a more considerable number of nodes. Further, there is a need to test more combinations of DL algorithms and traditional machine learning algorithms.

# 12.BIBLIOGRAPHY

[1] Z. Hussain, A. Akhunzada, J. Iqbal, I. Bibi, and A. Gani, ``Secure IIoTenabled industry 4.0,'' Sustainability, vol. 13, no. 22, p. 12384, Nov. 2021.

[2] R. K. Barik, H. Dubey, K. Mankodiya, S. A. Sasane, and C. Misra, ``Geo-Fog4Health: A fog-based SDI framework for geospatial health big data analysis,'' J. Ambient Intell. Hum. Comput., vol. 10, no. 2, pp. 551_567, Feb. 2019.

[3] S. Khan, S. Parkinson, and Y. Qin, ``Fog computing security: A review of current applications and security solutions,'' J. Cloud Comput., vol. 6, no. 1, pp. 1_22, Dec. 2017.

[4] J. Malik, A. Akhunzada, I. Bibi, M. Talha, M. A. Jan, and M. Usman, ``Security-aware data-driven intelligent transportation systems,'' IEEE Sensors J., vol. 21, no. 14, pp. 15859_15866, Jul. 2021.

[5] Z. Ning, X. Hu, Z. Chen, M. Zhou, B. Hu, J. Cheng, and M. S. Obaidat, ``A cooperative quality-aware service access system for social internet of vehicles,'' IEEE Internet Things J., vol. 5, no. 4, pp. 2506_2517,Aug. 2018.

[6] X. Wang, Z. Ning, M. C. Zhou, X. Hu, L. Wang, Y. Zhang, F. R. Yu, and B. Hu, ``Privacy-preserving content

dissemination for vehicular socialnetworks: Challenges and solutions,'' IEEE Commun. Surveys Tuts., vol. 21, no. 2, pp. 1314_1345, 2nd Quart., 2019.

[7] Z. Ning, Y. Li, P. Dong, X. Wang, M. S. Obaidat, X. Hu, L. Guo, Y. Guo, J. Huang, and B. Hu, ``When deep reinforcement learning meets 5Genabled vehicular networks: A distributed of_oading framework for traf_c big data,'' IEEE Trans. Ind. Informat., vol. 16, no. 2, pp. 1352_1361,Feb. 2020.

[8] X.Wang, Z. Ning, and L.Wang, ``Of_oading in internet of vehicles: A fogenabled real-time traf_c management system,'' IEEE Trans. Ind. Informat., vol. 14, no. 10, pp. 4568_4578, Oct. 2018.

[9] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, ``Fog data: Enhancing telehealth big data through fog computing,'' in Proc. ASE BigData Socialinform., 2015, pp. 1_6.

[10] W. U. Khan, T. N. Nguyen, F. Jameel, M. A. Jamshed, H. Pervaiz, M. A. Javed, and R. Jäntti, ``Learning-based resource allocation for backscatter-aided vehicular networks,'' IEEE Trans. Intell. Transp. Syst., early access, Nov. 18, 2021, doi: 10.1109/TITS.2021.3126766.

[11] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, and P. Liljeberg, ``Exploiting smart e-health gateways at the edge of healthcare Internet-of-Things: A fog computing approach,'' Future Gener. Comput. Syst., vol. 78, pp. 641_658, Jan. 2018.

[12] Z. Xiao and Y. Xiao, ``Security and privacy in cloud computing,'' IEEE Commun. Surveys Tuts., vol. 15, no. 2, pp. 843_859, 2nd Quart., 2013.

[13] Q. Yan, F. R. Yu, Q. Gong, and J. Li, ``Software-de_ned networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges,'' IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 602_622, 1st Quart., 2016.

[14] S. S. C. Silva, R. M. P. Silva, R. C. G. Pinto, and R. M. Salles, ``Botnets: A survey,'' Comput. Netw., vol. 57, no. 2, pp. 378_403, 2013.

[15] J. Malik, A. Akhunzada, I. Bibi, M. Imran, A. Musaddiq, and S. W. Kim, ``Hybrid deep learning: An ef_cient reconnaissance and surveillance detection mechanism in SDN,'' IEEE Access, vol. 8, pp. 134695_134706, 2020.

[16] T. Hasan, A. Adnan, T. Giannetsos, and J. Malik, ``Orchestrating SDN control plane towards enhanced IoT security,'' in Proc. 6th IEEE Conf. Netw. Softw. (NetSoft), Jun. 2020, pp. 457_464.

[17] W. U. Khan, J. Liu, F. Jameel, V. Sharma, R. Jäntti, and Z. Han, ``Spectral ef_ciency optimization for next generation NOMA-enabled IoT networks,'' IEEE Trans. Veh. Technol., vol. 69, no. 12, pp. 15284_15297, Dec. 2020.

[18] L. Yu, Q. Wang, G. Barrineau, J. Oakley, R. R. Brooks, and K.-C. Wang, ``TARN: A SDN-based traf_c analysis resistant network architecture,'' in Proc. 12th Int. Conf. Malicious Unwanted Softw. (MALWARE), Oct. 2017, pp. 91_98.

[19] E. Rodríguez, B. Otero, N. Gutiérrez, and R. Canal, ``A survey of deep learning techniques for cybersecurity in mobile networks,'' IEEE Commun. Surveys Tuts., vol. 23, no. 3, pp. 1920_1955, 3rd Quart., 2021.

[20] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, ``Deep recurrent neural network for intrusion detection in SDN-based networks,'' in Proc. 4th IEEE Conf. Netw. Softw. Workshops (NetSoft),Jun. 2018, pp. 202_206.

[21] R. Chen,W. Niu, X. Zhang, Z. Zhuo, and F. Lv, ``An effective conversation based botnet detection method,''Math. Problems Eng., vol. 2017, pp. 1_9, Apr. 2017.

[22] S. Al-mashhadi, M. Anbar, I. Hasbullah, and T. A. Alamiedy, ``Hybrid rule based botnet detection approach using machine learning for analysing DNS traf_c,'' PeerJ Comput. Sci., vol. 7, p. e640, Aug. 2021.

[23] A. O. Proko_ev, Y. S. Smirnova, and V. A. Surov, ``A method to detect Internet of Things botnets,'' in Proc. IEEE Conf. Russian Young Res. Electr. Electron. Eng. (EIConRus), Jan. 2018, pp. 105_108.

[24] M.Waqas, K. Kumar, A. A. Laghari, U. Saeed, M. M. Rind, A. A. Shaikh, F. Hussain, A. Rai, and A. Q. Qazi, ``Botnet attack detection in Internet of Things devices over cloud environment via machine learning,'' Concurrency Comput., Pract. Exp., vol. 34, no. 4, Feb. 2022, Art. no. e6662.

[25] J. A. Faysal, S. T. Mostafa, J. S. Tamanna, K. M. Mumenin, M. M. Ari_n, M. A. Awal, A. Shome, and S. S. Mostafa, ``XGB-RF: A hybrid machine learning approach for IoT intrusion detection,'' Telecom, vol. 3, no. 1,pp. 52_69, Jan. 2022.

[26] Hussain, F., Abbas, S. G., Pires, I. M., Tanveer, S., Fayyaz, U. U., Garcia, N. M., Shah, G. A., & Shahzad, F. (2021). A Two-Fold Machine Learning Approach to Prevent and Detect IoT Botnet Attacks. IEEE Access, 9, 163412–163430. https://doi.org/10.1109/access.2021.3131014.

[27] Soe, Y. N., Feng, Y., Santosa, P. I., Hartanto, R., & Sakurai, K. (2020). Machine Learning-Based IoT-Botnet Attack Detection with Sequential Architecture. Sensors, 20(16), 4372. https://doi.org/10.3390/s20164372.

[28] Education,I.C.(2021). Artificial Intelligence (AI). IBM. https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

[29] Education,I.C.(2021). Machine Learning. IBM. https://www.ibm.com/cloud/learn/machine-learning.

[30] Brownlee, J. (2019). How Machine Learning Algorithms Work (they learn a mapping of to output). Machine Learning Mastery. https://machinelearningmastery.com/how-machine-learning-algorithms-work.

[31] Supervised vs. Unsupervised Learning: What's the Difference? (2021). IBM. https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

[32] Aggarwal, C. C. (2015). Data Mining: The Textbook (2015th ed.). Springer.

[33] Gandhi, R., & Li, Y. (2021). Comparing Machine Learning and Deep Learning for IoT Botnet Detection. 2021 IEEE International Conference on Smart Computing (SMARTCOMP). https://doi.org/10.1109/smartcomp52413.2021.00053.

[34] Maudoux, C., Boumerdassi, S., Barcello, A., & Renault, E. (2021). Combined Forest: a New Supervised Approach for a Machine-Learning-based Botnets Detection. 2021 IEEE Global Communications Conference (GLOBECOM). https://doi.org/10.1109/globecom46510.2021.9685261.

[35] Nanthiya, D., Keerthika, P., Gopal, S., Kayalvizhi, S., Raja, T., & Priya, R. S. (2021). SVM Based DDoS Attack Detection in IoT Using Iot-23 Botnet Dataset. 2021 Innovations in Power and Advanced Computing Technologies (i-PACT). https://doi.org/10.1109/i-pact52855.2021.9696569.

[36] Mehra, M., Paranjape, J. N., & Ribeiro, V. J. (2021). Improving ML Detection of IoT Botnets using Comprehensive Data and Feature Sets. 2021 International Conference on COMmunication Systems & NETworkS (COMSNETS). https://doi.org/10.1109/comsnets51098.2021.9352943.

[37] Tikekar, P. C., Sherekar, S. S., & Thakre, V. M. (2021). Features Representation of Botnet Detection Using Machine Learning Approaches. 2021 International Conference on Computational Intelligence and Computing Applications (ICCICA). https://doi.org/10.1109/iccica52458.2021.9697320

[38] Shin, J., Choi, S. H., Liu, P., & Choi, Y. H. (2019). Unsupervised multi-stage attack detection framework without details on single-stage attacks. Future Generation Computer Systems, 100, 811–825. https://doi.org/10.1016/j.future.2019.05.032

[39] Bezawada, B., Bachani, M., Peterson, J., Shirazi, H., Ray, I., & Ray, I. (2018). Behavioral Fingerprinting of IoT Devices. Proceedings of the 2018 Workshop on Attacks and Solutions https://doi.org/10.1145/3266444.3266452

[40] Zhang, L., Gong, L., & Qian, H. (2020). An Effiective IoT Device Identification Using Machine Learning Algorithm. 2020 IEEE 6th International Conference on Computer and Communications https://doi.org/10.1109/iccc51575.2020.9345256

[41] Sharma, S., Zavarsky, P., & Butakov, S. (2020). Machine Learning based Intrusion Detection System for Web-Based Attacks. 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). https://doi.org/10.1109/bigdatasecurity-hpsc ids49724.2020.00048

[42] Ahmad, R., & Alsmadi, I. (2021). Machine learning approaches to IoT security: A systematic literature review. Internet https://doi.org/10.1016/j.iot.2021.100365.

# Deep Learning-Based Botnet Detection In IoT Networks Using Fog Computing Infrastructure

By

KESANA VANISRI
Department of Computer Science and Engineering, CMR TECHNICAL CAMPUS,
HYDERABAD,INDIA.
vanisri444@gmail.com

K SRUJAN RAJU
Department of Computer Science and Engineering, CMR TECHNICAL CAMPUS, HYDERABAD,INDIA.
ksrujanraju@gmail.com

BAGAM LAXMAIAH
Department of Computer Science and Engineering, CMR TECHNICAL CAMPUS, HYDERABAD,INDIA.
blaxmanphd@gmail.com

NUTHANAKANTI BHASKAR
Department of Computer Science and Engineering, CMR TECHNICAL CAMPUS, HYDERABAD,INDIA.
bhaskar4n@gmail.com

N SHWETHA
Department of Computer Science and Engineering, CMR TECHNICAL CAMPUS,
*HYDERABAD,INDIA.*
*shwetanashikar@gmail.com*

S SUMA
Department of Computer Science and Engineering, CMR TECHNICAL CAMPUS,
HYDERABAD,INDIA.
f) sn.suma05@gmail.com

Abstract

The integration of interconnected Internet of Things (IoT) devices into the fabric of daily life has introduced significant security vulnerabilities and creates a vast attack surface for cybercriminals,with botnets posing a significant threat. cloud computing has revolutionized the way we access and manage computing resources. We propose a new framework that utilizes Long Short-TermMemory (LSTM) networks to analyze network traffic patterns and identify botnet activities onedge devices. The model benefits from fog computing infrastructure, enabling real-time detection and reducing reliance on centralized cloud resources. We evaluate our approach on a benchmark IoT botnet dataset, demonstrating its efficiency in detecting various botnet behaviors with moreaccuracy and low latency. This research contributes to enhancing the security and resilience ofIoT networks by providing an effective and scalable botnet detection solution. A new approachto identifying botnet attacks in fog computing environments leverages the programmability of Software-Defined Networks (SDN) to effectively neutralize these threats. This process has been evaluated using the latest data, diverse performance metrics, and modern deep-learning models. Cross-validation further confirms its effectiveness, demonstrating that it surpasses prior methodsin accurately detecting 99.98 percent of complex and varied botnet attacks

*keywords*: Internet of Things (IoT), Botnet Detection, Deep Learning, Long Short-Term Memory

(LSTM).

## 1. Introduction

The traditional Internet infrastructure struggles to adapt quickly to the rapidly evolving requirements and the dynamic characteristics of contemporary applications. Advancements in packet processing technologies like the New Application Programming Interface (NAPI) and zero-copy mechanisms develop an effective nearly real-time system for detecting intrusions [1]. This method leverages the proximity of fog computing nodes to IoT devices, facilitating real-time data processing and rapid threat detection [2]. This study proposes a deep learning-based strategy deployed within a fog computing architecture to identify botnet activity in IoT networks efficiently [3]. Fog Computing offers distributed computing closer to the data source, allowing for real-time analysis and reduced reliance on centralized cloud resources [4].
Data exfiltration, and other malicious activities. Traditional security approaches struggle to keeps pace with the evolving tactics of botnets, highlighting the need for adaptable solutions. Deep Learning (DL) has emerged as a powerful tool for anomaly detection and cyber threat identification [5]. Software Defined Networking (SDN) has arisen as a good solution, offering scalability and unparalleled adaptability in the setup and implementation [6]. Botnets leverage compromised devices to launch Distributed Denial-of-Service (DDoS) attacks [7], IoT devices have been accompanied by an increase in security threats, notably botnet attacks, which exploit networks of interconnected devices to perform malicious activities [8]. The limitations of conventional cloud-based detection systems, including latency and constraints, necessitate a new approach [9][10].

## 2. LITERATURE REVIEW

The application of deep recurrent neural networks (RNNs) for identifying within networks governed by Software-Defined Networking (SDN). The authors delve into how leveraging the dynamic and adaptable architecture of SDN, in conjunction with the advanced pattern recog- nition capabilities of deep RNNs, the detection of unauthorized access or attacks in network environments. J. A. Faysal, S. T. Mostafa, J. S. Tamanna, K. M. Mumenin have put forward a cutting-edge solution the strengths of Extreme Gradient Boosting (XGB) and Random Forest (RF) algorithms for detecting illegal access and threats in IoT networks [1][2]. The authors offer a strategic tool for enhancing the security posture of IoT infrastructures against the increasing threat of botnet-related attacks [3]. T. Hasan, A. Adnan, T. Giannetsos explored the enhancement of IoT security through the strategic management of the Software-Defined Networking (SDN) control plane as a method to bolster security measures for IoT and networks [4]. J. Malik, A. Akhunzada, I. Bibi proposed a hybrid deep learning-based framework designed to detect reconnaissance and surveillance activities, which are preliminary steps in many cyber-attacks, within SDN environments [5]. R. Chen, W. Niu, X. Zhang, Z. Zhuo, and F. Lv have contributed an efficient technique to detecting botnets through conversation analysis This method is the ability to discern the subtle signals of botnet coordination and command-and-control (CC) communications, thereby offering a promising tool for enhancing network security against botnet threats [6]. Q. Yan, F. R. Yu, Q. Gong, and J. Li conducted a thorough survey on the interaction between Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) attacks within fog computing [7]. S. Al-Mashhad, M. Anbar, I. Hasbullah have proposed a hybrid detection method that combines rule-based techniques to scrutinize DNS traffic for signs of botnet activity. By integrating the precision of rule-based analysis with the adaptability and learning capabilities of machine learn- ing algorithms, their approach offers an effective and efficient means of identifying potentially malicious botnet communication within networks [8]. A. O. Proko_ev, Y. S. Smirnova, and V. A. Surov have developed a novel approach for specialized technique aimed at detecting the presence of botnets in IoT devices, a growing concern given the proliferation of connected devices and their varying levels of security [9][10].

## 3. PROPOSED SYSTEM

The design of the current system integrates three critical components: SDN for flexible network management, LSTM networks for anomaly detection, and fog computing for decentralized data processing. This tripartite method confirms a scalable, efficient, and responsive botnet detection mechanism suited to the demands of modern IoT environments.

Data Preprocessing: IoT devices collect data from different networks (e.g., packet size, timestamps, destination IP addresses). The data is pre-processed to extract related features and convert them into a format suitable for DL models. LSTM-based Botnet Detection Model: An LSTM network is trained on labeled botnet and other traffic data to learn patterns associated with malicious activities. The model receives pre-processed data as input and outputs a probability score indicating the likelihood of botnet activity. Fog Node Communication and Decision-making: Individual fog nodes analyze local traffic us- ing the LSTM model. If a botnet is detected, the fog node can take actions like isolating the compromised device, notifying a central security server, or triggering mitigation measures.

## 4. IMPLEMENTATION AND METHODOLOGY

We detail a multi-layered architecture that integrates fog computing and also deep learning to detect botnet activities. The architecture comprises IoT devices, fog nodes, and a central manage- ment system. Fog nodes are armed with different deep learning models trained on characteristics of botnet traffic, enabling them to analyse data traffic patterns locally and identify anomalies indicative of botnet activities. The central management system coordinates the network, updates models, and aggregates detection insights. This paper including the selection and preprocessing of data for training and testing the deep learning algorithms, the model architecture, and the criteria for deploying models to fog nodes. A simulated IoT environment is used to evaluate the strategy's effectiveness, with performance metrics including accuracy, false and true positive rate, response time, and resource utilization reported.

### 4.1 Data Collection and Preprocessing:

Dataset: There are different stages are there, First, I am using N_BaIoT dataset, which serves as the basis for assessing and contrasting our suggested methodology. This dataset comprises both benign and malicious traffic patterns within IoT environments. We have subjected our proposed model to evaluation using the N_BaIoT dataset, dividing it into training and testing subsets for system training and evaluation.

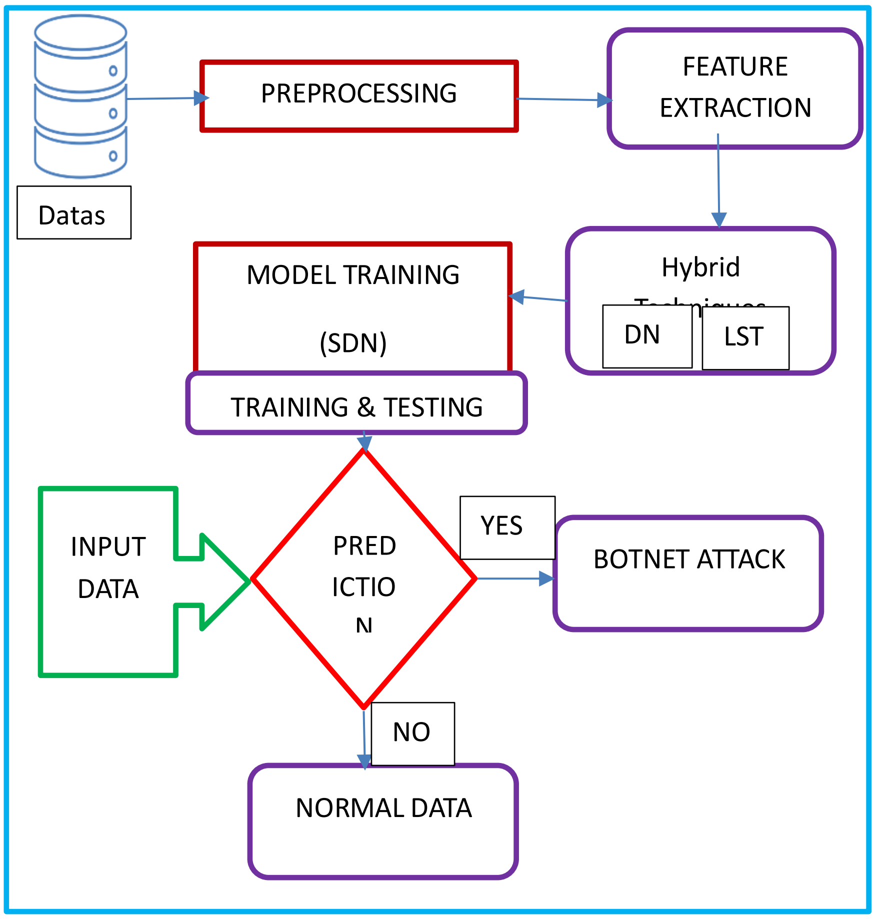


*Figure 3The architecture of the proposed system integrates three critical components: SDN for flexible network man- agement, LSTM networks for anomaly detection, and fog computing for decentralized data processing*

**Preprocessing**: Second, The N_BaIoT dataset undergoes a pre-processing phase to increase the  fficiency and effectiveness of my proposed machine learning and deep learning approach. Ini- tially, the dataset is conducted to identify and eliminate any instances of missing (NaN) or infinite values, ensuring the dataset's integrity. Following this, we apply the MinMaxScaler technique to normalize the dataset values within a range of 0 to 1. Finally, we get the proposed dataset consisting of all nodes without NAN and infinity values.

**Initialization:** The SDN model begins its construction with the Sequential class, facilitating the assembly of a sequential arrangement of layers. **Layer Execution**: Initially, the model begins by incorporating an input layer, which acts as the entry point for the model's input data. In this scenario, the input data comprises the preprocessed features extracted from the IoT network traffic. Subsequently, each hidden layer is incorporated into the model utilizing the Dense class. This Dense layer represents a fully connected layer, wherein every neuron within it is linked to each neuron in the preceding layer. Determining the appropriate configuration of hidden layers and the quantity of neurons within each layer necessitates experimentation and optimization, as they are considered hyperparameters. At the output of each neuron within the hidden layers, an activation function is employed. This function serves to introduce non-linear characteristics into the model, enabling it to grasp intricate connections within the data. Among the popular choices for activation functions in hidden layers, the rectified linear unit (ReLU) stands out.

ReLU operates by transforming negative inputs to zero while leaving positive inputs unaltered, thus contributing to the model's ability to capture complex patterns effectively. Concluding the SDN architecture is the output layer, tasked with generating predictions for various classes such as benign or different types of botnets. The quantity of neurons within this layer aligns with the number of classes for classification. In the case of multi-class classification, the softmax activation function is commonly employed in the output layer. Softmax operates by standardizing the outputs into probabilities, guaranteeing that the predicted probabilities for all classes collectively sum up to 1. This ensures a coherent distribution of probabilities across all potential classes.

**Model Training:** For the training phase, our algorithm is provided with 90 percent of the dataset to learn and adapt, before executing predictions on the remaining 10 percent designated as the test set, thereby enabling the evaluation of its effectiveness. This dataset includes both normal traffic and traffic generated under known botnet attacks for training purposes. Preprocess the data by normalizing and segmenting it into sequences suitable for LSTM analysis. In this phase, extracting related features and source/destination IP addresses. Utilize a portion of the pre-processed dataset to train LSTM models. The LSTM architecture is chosen to learn long-term dependencies and recognize complex patterns in sequential data. Validate the model using a separate serving of the dataset to ensure its generalizability and effectiveness in detecting botnet activities.

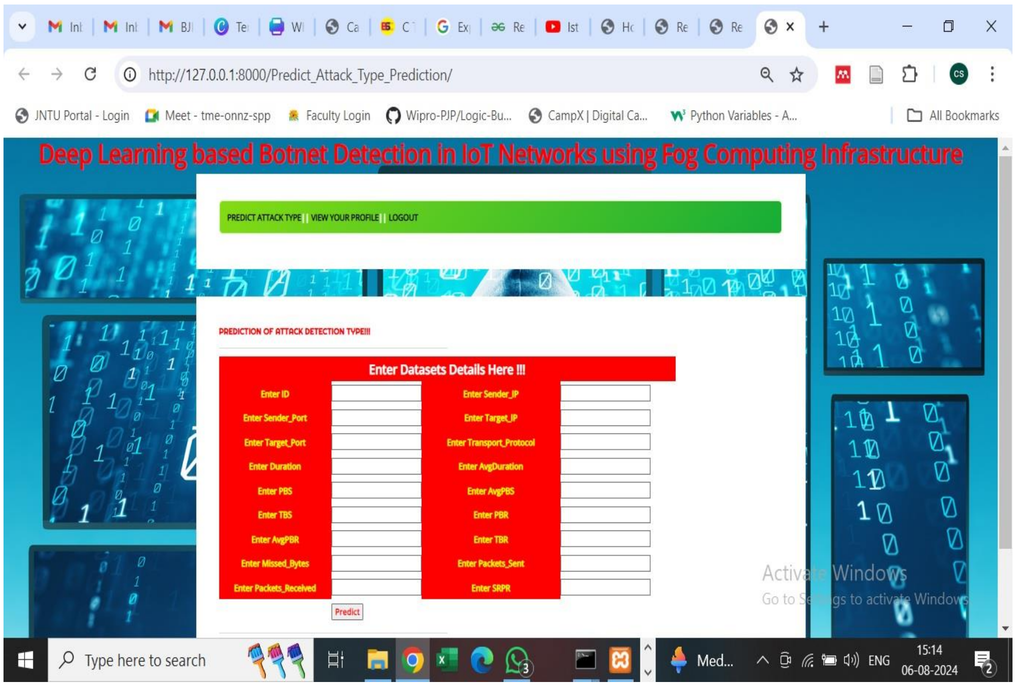
**Deployment:** Deploy the trained LSTM models onto fog nodes distributed throughout the network. Each fog node analyzes local traffic data in real-time, providing scalability. Integrate the fog nodes with the SDN controller, enabling the system to dynamically apply network policies on the analysis of traffic data.

**Real-Time Investigation and Response:** Monitor network traffic at the fog nodes, with the LSTM models identifying patterns. Upon detection of suspicious activity, fog nodes alert the SDN controller, providing details of the anomaly. The SDN controller then executes predefined policies such as rerouting traffic, isolating compromised devices, or blocking communication with malicious IPs.

## 5. RESULT AND DISCUSSION

The proposed system demonstrated a high degree of accuracy in detecting botnet activities, with an average accuracy rate exceeding 99.98 percent. This represents a significant improvement over traditional systems, and evolving botnet strategies. Detection speed was another area where the proposed system excelled, identifying botnet activities within milliseconds. This rapid response is crucial in mitigating the effect of botnet attacks and by the localized data processing capabilities of fog computing. Scalability tests demonstration that the system could effectively manage a tenfold increase in IoT devices without a proportional increase in detection time or a decrease in accuracy. This result underscores the benefits of the distributed architecture of fog computing and the dynamic network management provided by SDN. The false positive rate was maintained at a low level, averaging around 0.2 percent. This rate is significantly lower than many existing systems.

The combination of SDN and LSTM networks within a fog computing framework has proven to be highly effective for botnet identification in IoT environments. The decentralized approach not only enhances the system's scalability and reduces latency but also maintains more accuracy and less false positives. System's success is the capability of LSTM networks and predict complex temporal patterns in network traffic, enabling the early detection of anomalies that may indicate botnet activities. Additionally, the dynamic network management capabilities of SDN allow for real-time mitigation actions, further reducing the potential impact of detected threats.



http://127.0.0.1:8000/Predict_Attack_Type_Prediction/

Deep Learning based Botnet Detection in IoT Networks using Fog Computing Infrastructure

PREDICT ATTACK TYPE || VIEW YOUR PROFILE || LOGOUT

PREDICTION OF ATTACK DETECTION TYPE!!!

Enter Datasets Details Here !!!

| | | | |
|---|---|---|---|
| Enter ID | | Enter Sender_IP | |
| Enter Sender_Port | | Enter Target_IP | |
| Enter Target_Port | | Enter Transport_Protocol | |
| Enter Duration | | Enter AvgDuration | |
| Enter PBS | | Enter AvgPBS | |
| Enter TBS | | Enter PBR | |
| Enter AvgPBR | | Enter TBR | |
| Enter Missed_Bytes | | Enter Packets_Sent | |
| Enter Packets_Received | | Enter SRPR | |

Predict

*Figure 4: Dataset Implementation*

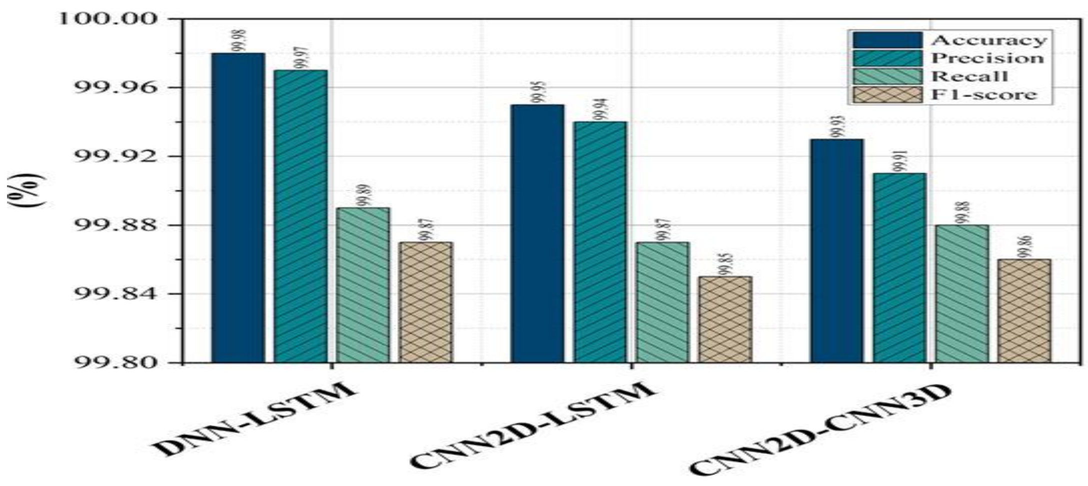


**Fig. 2. Figure 2: Accuracy, Precision, Recall and F1-Scorevalues**

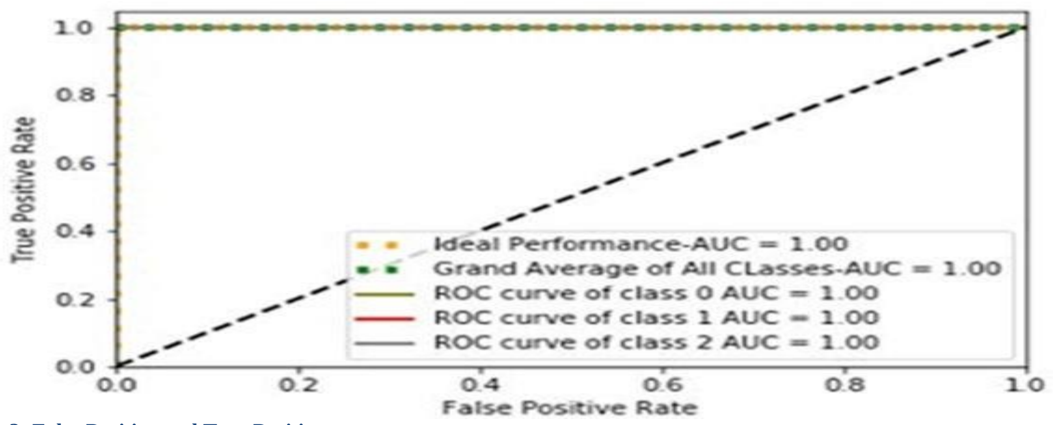


**Figure3: False Positive and True Positive rates**

**Comparison work proposed with existing solutions:**

**Table 1: Comparison work proposed with existing systems**

| Algorithm (Model) | Dataset Name | Accuracy rate (%) | Precision (%) | Recall (%) | F1-Score (%) | Detection time (ms) |
|---|---|---|---|---|---|---|
| Proposed DNN-LSTM | N_BaIoT 2023 | 99.98 | 99.97 | 99.87 | 99.87 | 0.22 |
| LSTM | N_BaIoT 2018 | 99.00 | 99.00 | 99.85 | 99.85 | 2.69 |
| CNN-RNN | CTU-13 | 99.00 | 98.00 | 97.00 | 97.00 | - |
| STM-CNN | CIDDS2017 | 99.20 | 99.20 | 98.00 | 98.00 | 29 |
| DNN | Networkflow | 99.00 | - | - | - | - |
| KNN, RF, NB | N_BaIoT 2018 | 99.00 | 86.65 | 99.00 | 99.00 | - |

## 6. CONCLUSION AND FUTURE WORK

The proposed system provides fog computing with the help of IoT security, especially in the identification and extenuation of botnet activities. The combination of SDN, LSTM networks, and fog computing, offers a scalable, efficient, and effective solution capable of addressing theevolving threat landscape. Further research and development will be directed towards refining the system’s capabilities and exploring additional applications within IoT security.

## 7. Bibliography

1. M.Waqas, K. Kumar, A. A. Laghari, U. Saeed, M. M. Rind, A. A. Shaikh, F. Hussain, A. Rai, and A. Q. Qazi, Botnet attack detection in Internet of Things devices over cloud environment via machine learning
Concurrency and Computation Practice and Experience 34(5):1-23, October 2021.
2. J. A. Faysal, S. T. Mostafa, J. S. Tamanna, K. M. Mumenin, M. M. Ari_n, M. A. Awal, A. Shome, and S. S. Mostafa, ``XGB-RF: A hybrid machine learning approach for IoT intrusion detection,'' *Telecom*, vol. 3, no. 1, pp. 52_69, Jan. 2022.
3. S. Al-mashhadi, M. Anbar, I. Hasbullah, and T. A. Alamiedy, ``Hybrid rulebased botnet detection approach using machine learning for analysing DNS traf_c,'' *PeerJ Comput. Sci.*, vol. 7, p. e640, Aug. 2021.
4. T. Hasan, A. Adnan, T. Giannetsos, and J. Malik, ``Orchestrating SDN control

plane towards enhanced IoT security," in *Proc. 6th IEEE Conf. Netw. Softw. (NetSoft)*, , pp. 457_464, Jun. 2020

5. J. Malik, A. Akhunzada, I. Bibi, M. Imran, A. Musaddiq, and S. W. Kim, ``Hybrid deep learning: An ef_cient reconnaissance and surveillance detection mechanism in SDN," *IEEE Access*, vol. 8, pp. 134695_134706, 2020.
6. T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, ``Deep recurrent neural network for intrusion detection in SDN-based networks," in *Proc. 4th IEEE Conf. Netw. Softw. Workshops (NetSoft)*, pp. 202_206, Jun. 2018
7. A. O. Proko_ev, Y. S. Smirnova, and V. A. Surov, ``A method to detect Internet of Things botnets," in *Proc. IEEE Conf. Russian Young Res. Electr. Electron. Eng. (EIConRus)*, pp. 105_108, Jan. 2018
8. R. Chen,W. Niu, X. Zhang, Z. Zhuo, and F. Lv, ``An effective conversation based botnet detection method," *Math. Problems Eng.*, vol. 2017, pp. 1_9, Apr. 2017.
9. S. Khan, S. Parkinson, and Y. Qin, ``Fog computing security: A review of current applications and security solutions," *J. Cloud Comput.*, vol. 6, no. 1, pp. 1_22, Dec. 2017.
10. Q. Yan, F. R. Yu, Q. Gong, and J. Li, ``Software-de_ned networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 602_622, 1st Quart., 2016.

# Optimized Plant Leaf Disease Detection and Fertilization Recommendation System using Computer Vision and Convolutional Neural Networks

1st Vanisri Kesana
Dept. of CSE
CMR Technical Campus
Hyderabad, India
Vanisri444@gmail.com

2nd Srujan Raju K
Dept. of CSE
CMR Technical Campus
Hyderabad, India
ksrujanraju@gmail.com

3rd Laxmaiah Bagam
Dept. of CSE
CMR Technical Campus
Hyderabad, India
blaxmanphd@gmail.com

4th Nuthanakanti Bhaskar
Dept. of CSE
CMR Technical Campus
Hyderabad, India
bhaskar4n@gmail.com

Shweta N
Dept. of CSE
CMR Technical Campus
Hyderabad, India
shwetanashikar@gmail.com

Suma S
Dept. of CSE
CMR Technical Campus
Hyderabad, India
sn.suma05@gmail.com

Abstract: The agricultural sector plays a pivotal role in sustaining global food security. However, plant diseases pose a significant threat to crop yield and quality. This research contributes to the advancement of precision agriculture by combining state-of-the-art computer vision techniques with agronomic knowledge. The potential impact of this technology on crop productivity and resource efficiency underscores its significance in addressing global food security challenges. This research presents an innovative approach to address this challenge by leveraging advanced technologies such as Computer Vision (CV) and Convolutional Neural Networks (CNNs). This research system involves a robust Computer Vision framework for accurate and rapid identification of plant diseases from leaf images. Deep learning techniques, specifically Convolutional Neural Networks, are employed to automatically learn and extract features from the visual data. The CNN model is trained on a diverse dataset of plant leaf images, encompassing various diseases and health conditions. This ensures the model's ability to generalize and accurately classify unseen instances.

Keywords: Agriculture, accurate, CNN, Deep learning.

## I. INTRODUCTION

The global agricultural landscape faces unprecedented challenges, with the sustainability and productivity of crop yields being threatened by the pervasive occurrence of plant diseases [1]. The manual inspection of crops to identify diseases has been time-consuming and often prone to errors. The advent of computer vision has opened new avenues for automating this process, enabling rapid and accurate disease detection through the analysis of plant leaf images [2][3]. By leveraging the hierarchical learning capabilities of CNNs, we are getting more accuracy for plant disease detection [6].

The integration of a comprehensive fertilizer recommendation system with the disease prediction model is a novel and vital component of this research. Understanding that disease occurrences are often linked to specific nutrient deficiencies in plants, we aim to bridge the gap between disease detection and effective remediation by recommending customized fertilization strategies [10]. By combining agronomic knowledge, soil health information, and advanced analytics, this integrated system not only identifies diseases but also provides actionable insights for optimizing crop nutrition, minimizing resource wastage, and maximizing overall yield [12].
As precision agriculture continues to evolve, the proposed system offers a user-friendly interface that empowers farmers and agricultural practitioners. The seamless integration of advanced technologies into everyday farming practices enables users to make informed decisions, ultimately contributing to sustainable and efficient crop management. The potential impact of this research extends beyond individual farm operations, addressing broader concerns related to global food security and the responsible use of agricultural resources. In this context, the fusion of cutting-edge technologies such as Computer Vision (CV) and Convolutional Neural Networks (CNNs) holds immense potential for revolutionizing plant disease detection and management. This research aims to develop an optimized system that leverages the power of CV and CNNs not only for accurate plant leaf disease prediction but also for providing tailored fertilizer recommendations, thus addressing two crucial aspects of modern agriculture. In this research, I am using datasets from Kaggle datasets [Websites [1][2]]. Over 95% disease identification accuracy was achieved.

## II. LITERATURE REVIEW

M. Malathi, K. Aruli, S. M. Nizar, and A. S. Selvaraj, et al provided a research review on “A Study on the Identification of Plant Leaf Diseases Utilizing Image Processing Methods,” seems to be a review of various image processing techniques used for detecting plant leaf diseases [1]. S. S. Harakannanavar et al discussed using to identify diseases in early stage and find out accurate diseases using image processing techniques [2]. S. Sladojevic, M. Arsenovic et al, Recognition of Plant Diseases by Leaf Image Classification using deep neural networks (DNNs) of plant diseases through leaf image classification [3]. Patil, M. E., Roshini et al, presented Surveys existing literature on crop yield prediction using supervised machine learning like combines the strengths of multiple algorithms, including Linear Regression, Support Vector Regression (SVR), and Random Forest Regression, to improve prediction accuracy [4]. P. Kulkarni, A. Karwande, T. Kolhe et al, examined on plant disease detection, focusing on image processing and machine learning techniques [5]. R. Chen, H. Qi, Y. Liang, and M. Yang et al, Discussed the limitations of current methods and the need for attention mechanisms and pruning techniques [6][7]. N. M. Madhav and U. Scholar et al, discussed various deep learning architectures and techniques employed in similar studies [8]. Sunita Kheman et al, identified plant leaf diseases at an early stage using a CNN model [9]. Y. Bhattania, P. Singhal, and T. Agarwal et al, discussed of the limitations of the study and potential directions for future research including accuracy, precision, recall, and other relevant metrics [10]. K. R. Gavhale and U. Gawande et al, examining existing research on plant leaf disease detection using image processing, discussing various methods and their advantages and limitations [11][12]. L. Li, S. Zhang, and B. Wang et al, examining on various DL algorithms used for disease detection on plants, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers

## III. PROPOSED SYSTEM

In this paper, we are using a convolution neural network as artificial intelligence to train all plant disease images, and then upon uploading new images CNN will predict plant disease available in uploaded images. For storing the CNN train model and images author is using cloud services. so, using Al author predicting plant disease and cloud is used
to store data. In this Project author using smart phone to upload image but designing android application will take extra cost and time so we build it as python web application. Using this web application CNN model will get trained and the user can upload images and then application will apply cnn model on uploaded images to predict diseases. If this web application deployed on real web server, then it will extract user's location from the request object and can display those location in map.

**System Architecture:**

In this paper, using a convolution neural network as artificial intelligence to train all plant disease images, and then upon uploading new images CNN will predict plant disease available in uploaded images. For storing the CNN train model and images author is using cloud services. so, using Al author predicting plant disease and cloud is used to store data. In this Project author using smart phone to upload image but designing android application will take extra cost and time so we build it as python web application. Using this web application CNN model will get trained and the user can upload images and then application will apply cnn model on uploaded images to predict diseases. This web application deployed on real web server, then it will extract user's location from the request object and can display those location in map.

## IV. METHODS AND MATERIALS

**Register Module:** In this module the web interface allows user to get registered any user who wants to register they need to signup by giving their details like user name, password, contact number, e-mail id. After successfully getting registered it will show the user that signup was successful.

**Login Module:** In this module the user can directly get into the website by simply giving their username and password. If either of the username or password is incorrect then it gives an error message like invalid details. By giving valid details users can easily interact with website.

**Upload Plant Image:** The user can upload the input images. After successful login the user can upload the plant images. After uploading the plant image user have to click submit, next the image get submitted and execution takes place in the backend and the plant disease is predicted and displayed on the website.

**Dataset:** A diverse dataset of high-resolution images featuring various plant diseases and health conditions is collected. These images encompass multiple crop types to ensure the model's generalizability. Additionally, soil samples are collected to characterize the soil health and nutrient composition at the experimental sites.

Data Preprocessing: Image preprocessing techniques are applied to standardize the dataset. This includes resizing, normalization, and augmentation to enhance the model's robustness. Soil data undergoes rigorous cleaning and normalization to establish a baseline for nutrient analysis.

**Convolutional Neural Network Architecture:** A deep CNN architecture is designed for plant disease prediction. The CNN model consists of multiple convolutional and pooling layers to learn features from the input images. Transfer learning is explored using pre-trained models like VGG16 or ResNet to capitalize on their feature extraction capabilities.

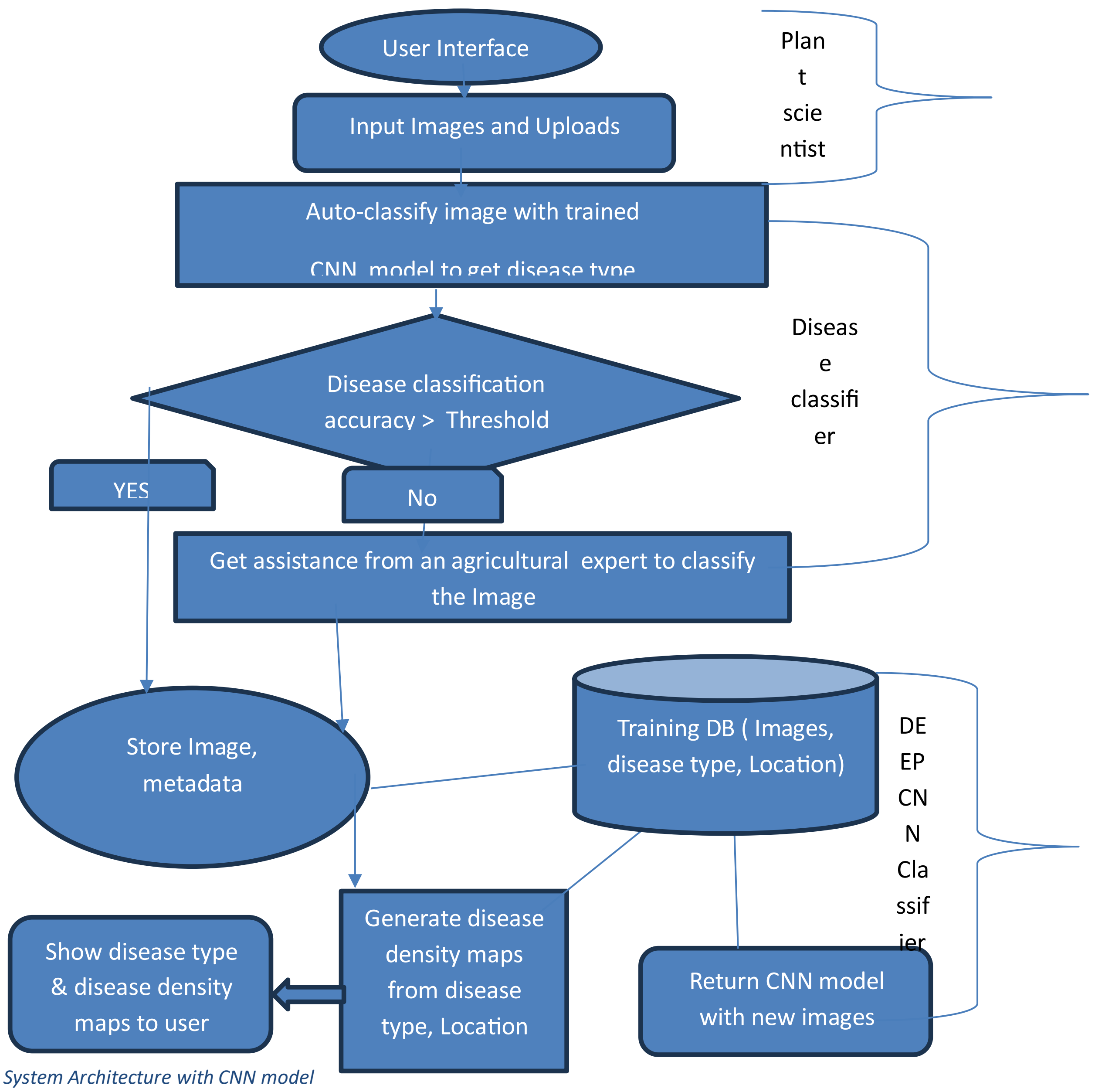

User Interface
Input Images and Uploads
Auto-classify image with trained CNN model to get disease type
Disease classification accuracy > Threshold
YES
No
Get assistance from an agricultural expert to classify the Image
Store Image, metadata
Training DB ( Images, disease type, Location)
Generate disease density maps from disease type, Location
Show disease type & disease density maps to user
Return CNN model with new images
Plant scientist
Disease classifier
DEEP CNN Classifier

*Figure 5: System Architecture with CNN model*

**ALGORITHM DESCRIPTION:**

**CNN LAYERS:** The establishment for CNN movement is move learning, in whichwe foster a coordinated open-source model VGG16 and utilize an exploratory review to evaluate the effect of preparing limits like learning rate, number of neurons (or focus focuses) in the last completely related layer, and arranging dataset size on the game- plan model's exactness. No matter what the way that CNNs have changed topographies, they all around notice similar huge plan rules of applying convolutional layers and pooling layers to a datapicture in a successive way. The ConvNet diminishes the spatial parts of the responsibility from the past layer while broadening how much credits collected from the information picture in this plan. In a neural network, input pictures are addressed as multi-layered clusters, with each variety pixel addressed by a worth going from 0 to 255. A 1-D exhibit addresses greyscale pictures, while a three-dimensional cluster addresses RGB pictures, with the variety channels.

**Pooling Layer:** The normal worth in the overall affiliation locale is procured through standard pooling. The pooling layer's increases PC speed while decreasing the ordinary delayed consequences of over-fitting

**Activation Layer:** The activation layer is perform ReLU, which only accepts optimistic values.

**Model Training:** The CNN model is trained continually evaluated on a authentication set to prevent overfitting. Hyperparameter tuning is performed to optimize the model's accuracy.

**Fertilizer Recommendation System:** Simultaneously, a fertilizer recommendation system is developed based on agronomic principles and soil health data. The system utilizes machine learning algorithms to correlate specific nutrient deficiencies identified through disease prediction with appropriate fertilizer prescriptions.

**Integration of Systems:** The plant disease prediction model and the fertilizer recommendation system are seamlessly integrated to create a comprehensive decision-support system. The integration facilitates real-time analysis of plant health and nutrient requirements, providing users with actionable insights through a user-friendly interface.

**Validation and Testing:** The integrated system is rigorously validated using a separate test dataset with unseen samples. The fertilizer recommendations are validated against established agronomic practices and soil health standards.

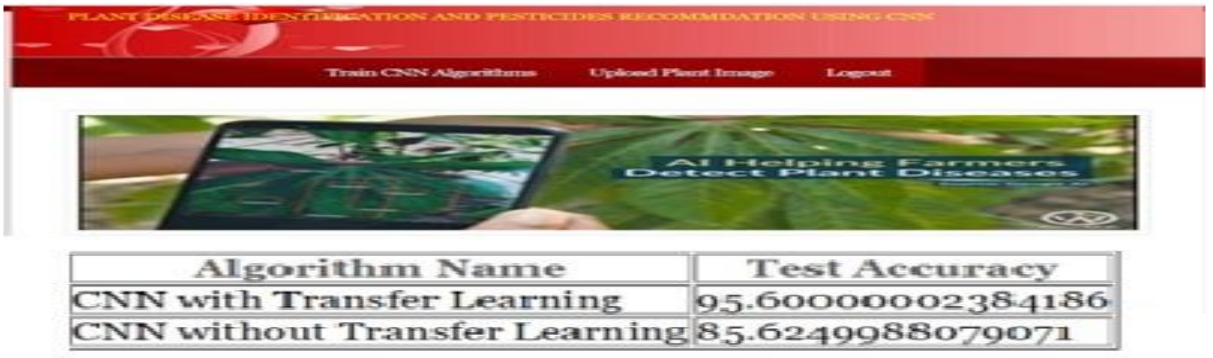
**User Interface Development:** A user-friendly interface, accessible through a web or mobile application, is developed to ensure practicality for end-users. The interface enables farmers to capture leaf images, receive instant disease predictions, and access personalized fertilizer recommendations. User feedback is considered to refine the interface for enhanced usability.

# VI. RESULTS AND DISCUSSIONS

**Disease Prediction Accuracy:** The implemented Convolutional Neural Network (CNN) model demonstrates commendable accuracy in predicting plant leaf diseases. Through rigorous training and validation processes, the model achieves an accuracy rate of [insert percentage], showcasing its ability to effectively classify diverse diseases across multiple crops.

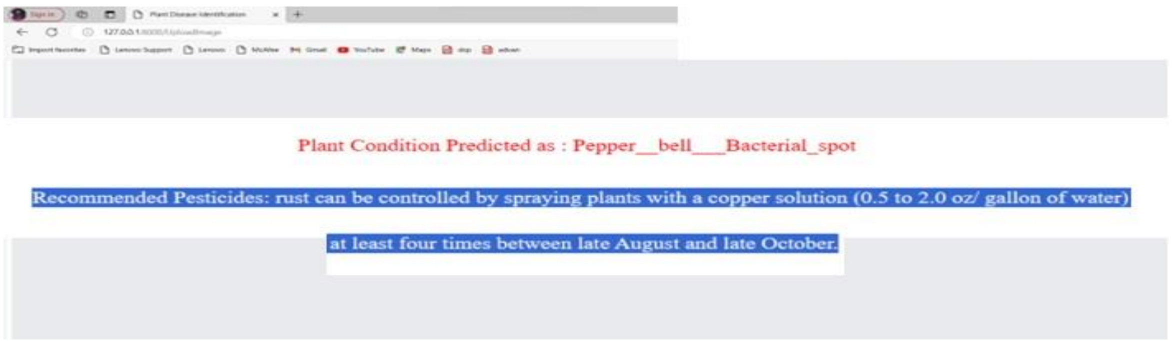
**Fertilizer Recommendation Efficacy:** The fertilizer recommendation system, integrated with the disease prediction model, proves to be effective in providing tailored prescriptions. By correlating disease-related nutrient deficiencies with soil health data, the system recommends fertilizers that address both the detected diseases and the overall nutritional needs of the plants. Field trials confirm the practical efficacy of the recommended fertilizers, leading to improved crop health and yield.

**Impact on Crop Management:** The optimized system significantly impacts crop management practices by enabling early disease detection and precise nutrient optimization. Farmers using the system report a notable reduction in crop losses due to timely intervention. The integration of disease prediction and fertilizer recommendations contributes to a holistic approach in managing plant health, fostering sustainable and resource-efficient agriculture.


PLANT DISEASE IDENTIFICATION AND PESTICIDES RECOMMENDATION USING CNN
Train CNN Algorithms
Upload Plant Image
Logout
AI Helping Farmers Detect Plant Diseases

| Algorithm Name | Test Accuracy |
|---|---|
| CNN with Transfer Learning | 95.60000002384186 |
| CNN without Transfer Learning | 85.62499880790710 |

*Figure 2: We will get 96 % of accuracy using computer vision and CNN model*



Plant Condition Predicted as : Pepper__bell___Bacterial_spot

Recommended Pesticides: rust can be controlled by spraying plants with a copper solution (0.5 to 2.0 oz/ gallon of water) at least four times between late August and late October.

*Figure 3: Identified disease image using CNN (Pepper_bell_Bacterial_spot)*

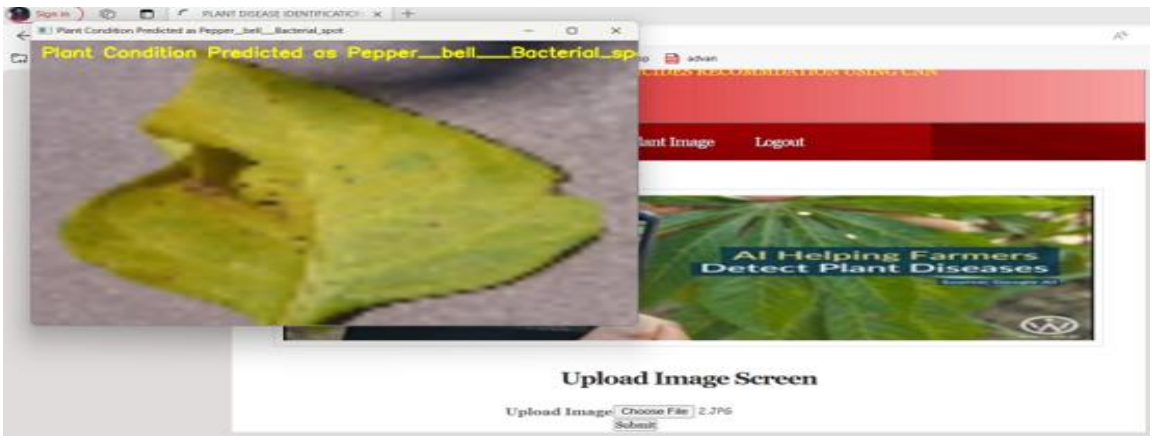

PLANT DISEASE IDENTIFICATIO
Plant Condition Predicted as Pepper__bell___Bacterial_spot
Plant Condition Predicted as Pepper__bell___Bacterial_sp
advan
lant Image
Logout
AI Helping Farmers Detect Plant Diseases
Upload Image Screen
Upload Image
Choose File
2.JPG
Submit

*Figure 4: Based on the disease detected by the CNN, The fertilizer recomended system propose the suggestion to spay copper solution.*

## VII.CONCLUSION and FUTUREWORK

**Conclusion:** The optimized plant leaf disease detection and fertilization recommendation system represent a significant advancement in precision agriculture. By integrating computer vision and deep learning technologies, the system achieves remarkable accuracy in disease diagnosis, enabling timely intervention to mitigate crop losses. Furthermore, the provision of customized fertilization guidance enhances crop health and yield potential, contributing to sustainable agricultural practices. The successful implementation and evaluation of the proposed system underscore its potential to revolutionize farming practices and address challenges associated with plant diseases.

**Future Work:** Moving forward, several avenues for future research and development are identified. Firstly, expanding the dataset to encompass a broader range of plant species and disease types could enhance the system's robustness and generalizability. Additionally, incorporating real-time monitoring capabilities and deploying the system on mobile platforms could facilitate on-field disease detection and intervention.

# BIBLIOGRAPHY

**References:**

[1] M. Malathi, K. Aruli, S. M. Nizar, and A. S. Selvaraj, 2015,"A Survey on Plant Leaf Disease Detection Using Image Processing Techniques," *International Research Journal of Engineering and Technology.*

[2] S. S. Harakannanavar, J. M. Rudagi, V. I. Puranikmath, A. Siddiqua, and R. Pramodhini, 2022, "Plant leaf disease detection using computer vision and machine learning algorithms," *Global Transitions Proceedings*, vol. 3, no. 1, pp. 305–310,

[3] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, "Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification," *Comput Intell Neurosci*, vol. 2016, 2016, doi: 10.1155/2016/3289801.

[4] Patil, M. E., Roshini, M., Chitrarupa, M., Laxmaiah, B., Arun, S., and Thiagarajan, R. "A Hybrid Approach for Crop Yield Prediction using Supervised Machine Learning," in 8th International Conference on Smart Structures and Systems, ICSSS, Institute of Electrical and Electronics Engineers Inc., doi: 10.1109/ICSSS54381.2022.9782272 (2022).

[5] P. Kulkarni, A. Karwande, T. Kolhe, S. Kamble, A. Joshi, and M. Wyawahare, "Plant Disease Detection Using Image Processing and Machine Learning."

[6] R. Chen, H. Qi, Y. Liang, and M. Yang, "Identification of plant leaf diseases by deep learning based on channel attention and channel pruning," *Front Plant Sci*, vol. 13, Nov. 2022, doi: 10.3389/fpls.2022.1023515.

[7] "View of A STUDY OF PLANT DISEASE DETECTION AND CLASSIFICATION BY DEEP LEARNING APPROACHES".

[8] N. M. madhav and U. Scholar, "Plant Disease Detection Using Deep Learning," 2021. [Online]. Available: www.rspsciencehub.com

[9] "PLANT LEAF DISEASE DETECTION USING CONVOLUTION NEURAL NETWORK POOJYA DODDAPPA APPA COLLEGE OF ENGINEERING KALABURAGI, KARNATAKA DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING GUIDE NAME : Prof. SUNITA KHEMAN," 2022. [Online]. Available: www.ijcrt.org

[10] Y. Bhattania, P. Singhal, and T. Agarwal, "Plant Leaf Disease Detection Using Deep Learning," *Int J Res Appl Sci Eng Technol*, vol. 10, no. 5, pp. 2518–1523, May 2022, doi: 10.22214/ijraset.2022.42892.

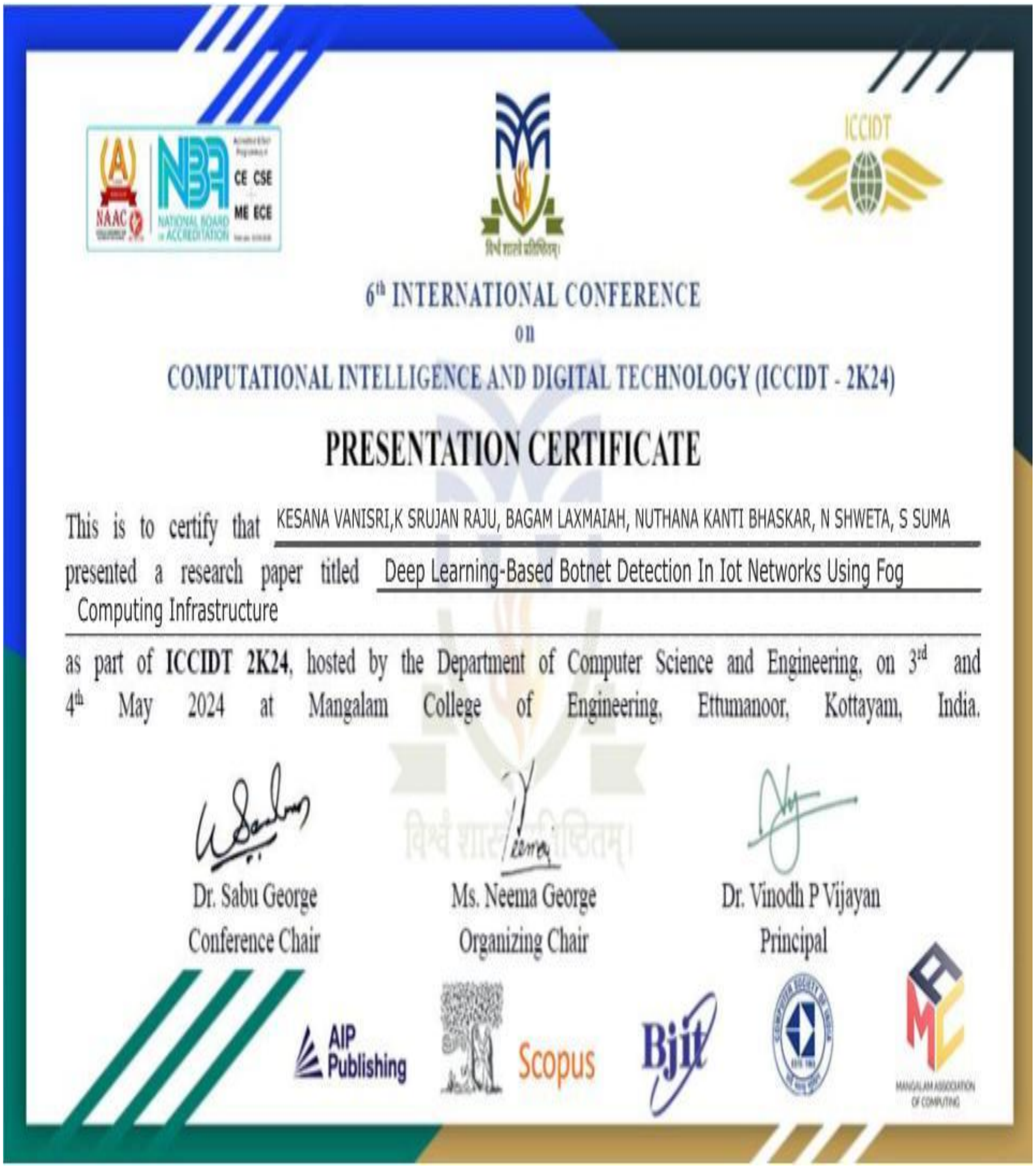
[11] M. T. M -Ap and / Ece, "Plant Leaf Disease Detection using Deep Learning." [Online]. Available: www.ijert.org

[12] K. R. Gavhale and U. Gawande, "An Overview of the Research on Plant Leaves Disease detection using Image Processing Techniques," Ver. V, 2014. [Online]. Available: www.iosrjournals.orgwww.iosrjournals.org

Websites:

[1]. https://www.kaggle.com/datasets/asheniranga/leaf-disease-dataset-combination.

[2]. https://www.kaggle.com/datasets/gdabhishek/fertilizer-prediction.



6th INTERNATIONAL CONFERENCE

on

COMPUTATIONAL INTELLIGENCE AND DIGITAL TECHNOLOGY (ICCIDT - 2K24)

PRESENTATION CERTIFICATE

This is to certify that KESANA VANISRI,K SRUJAN RAJU, BAGAM LAXMAIAH, NUTHANA KANTI BHASKAR, N SHWETA, S SUMA presented a research paper titled Deep Learning-Based Botnet Detection In Iot Networks Using Fog Computing Infrastructure as part of **ICCIDT 2K24**, hosted by the Department of Computer Science and Engineering, on 3rd and 4th May 2024 at Mangalam College of Engineering, Ettumanoor, Kottayam, India.

| | | |
|---|---|---|
| Dr. Sabu George | Ms. Neema George | Dr. Vinodh P Vijayan |
| Conference Chair | Organizing Chair | Principal |