

# Leveraging Advanced Machine Learning for DDoS Attack Detection

<sup>1</sup>Dr G K V Narasimha Reddy, <sup>2</sup>Bandi Amrutha

<sup>1</sup>Professor, Department of Computer Science and Engineering,

<sup>2</sup>Students, Department of Computer Science and Engineering,

St.Johns College of Engineering and Technology, Yemmiganr, Andhra Pradesh, India

<sup>1</sup>[gkvnreddy@gmail.com](mailto:gkvnreddy@gmail.com)

<sup>2</sup>[12amrutha04@gmail.com](mailto:12amrutha04@gmail.com)

## Abstract

With the rapid expansion of cloud computing and online services, Distributed Denial of Service (DDoS) attacks have become a major cybersecurity threat, causing severe service disruptions and financial losses. Traditional machine learning-based DDoS detection methods, such as SVM, KNN, and Logistic Regression, exhibit limitations in accuracy, scalability, and computational efficiency. To overcome these challenges, this project proposes a Deep Neural Network (DNN)-based detection system that automatically extracts and learns network traffic patterns to distinguish between legitimate and malicious activities. The proposed DNN model is trained on the SDN dataset, a large-scale dataset containing various DDoS attack types, and achieves an accuracy of 99.38%, significantly outperforming conventional hybrid models with a 96% accuracy. The model demonstrates superior performance in key evaluation metrics, including precision, recall, and F1-score, ensuring a highly reliable detection mechanism. Additionally, hyperparameter tuning optimizes the learning process, further improving accuracy and reducing false positives. This study highlights the effectiveness of deep learning in mitigating DDoS threats by enabling real-time attack detection with minimal computational overhead. Future enhancements could involve integrating advanced architectures like CNNs or RNNs for better feature extraction and implementing real-time adaptive mechanisms to counter evolving cyber threats. The proposed model provides a scalable and robust solution for enhancing network security and protecting cloud-based infrastructures from sophisticated DDoS attacks.

**Keywords**— DDoS attack, Deep Neural Network (DNN), Machine Learning, Cyber security, Cloud Computing.

## 1. INTRODUCTION

### 1.1 Overview

With the growing popularity of e-commerce, and online trading, the majority of businesses rely on cloud computing. To reduce costs and make better use of resources, the company utilizes a hybrid cloud, which is a combination of public and private clouds. In a distributed cloud environment, DDoS (distributed denial of service) attacks are common. The services are inaccessible to legitimate users because of the massive amount of traffic generated, leading to financial losses. The attacker takes control of large numbers of devices before attacking the server. Because of the large population in India, it is a prime target for cybercriminals and financial espionage. Today, security administrators don't have a good answer on how to protect the environment from DDoS attacks. The malicious traffic generated by DDoS attacks depletes the cloud resources. The attackers typically use botnets to steal information in formation. There may be barriers, but

several researchers have developed various mitigation measures. The threats in the cloud environment lead to asset loss.

DDoS (Distributed Denial-of-Service) attacks are known to be very hard to defend against because they are distributed. A DDoS attack is basically a zombie command to flood the target with fake traffic. The goal of a DDoS attack is to deny legitimate users access to servers. This can have a huge impact on any online activity and cause long-term damage. In real assault networks, the number of devices is significantly higher. The main purpose of this type of attack is to disrupt networks, consume network resources and prevent genuine users from accessing them. In a DDoS attack, one or more attackers perform a denial of service (DoS) attack on a target system. The main attack vectors used in a DDoS attack are: UDP flood, HTTP flood, SYN flood, ICMP flood, DNS.

Detecting DDoS attacks is one of the most important DDoS defense mechanisms. However, DDoS attack detection is difficult to automate because attack traffic is very similar to normal traffic and attackers often try to imitate flash crowds. An attack activity with low traffic can be seen as a legitimate attack in the early stages. Some researchers try statistical machine learning methods. Machine learning methods identify DDoS attacks using statistical features and perform better than statistical methods. However, there are several drawbacks to this approach

- 1) it requires extensive network expertise and DDoS experiments to select the appropriate statistical features;
- 2) it is very difficult to maintain and update the hybrid model;
- 3) it requires updating the model and more computational resources compared to a single algorithm
- 4) the possibility that the error could spread through the other components of the hybrid system, potentially impacting the overall accuracy.

A DDoS assault employs a large number of computers to conduct a coordinated DOS attack against one or more targets. By using the resources of numerous un aware participant computers, which act as attack platforms, the perpetrator is able to greatly increase the efficacy of the DOS.

In conclusion, the limitations and lack of research in previous work highlight the need for a real-time detection scheme that effectively identifies and mitigates M-DOS attacks in cloud computing environments.

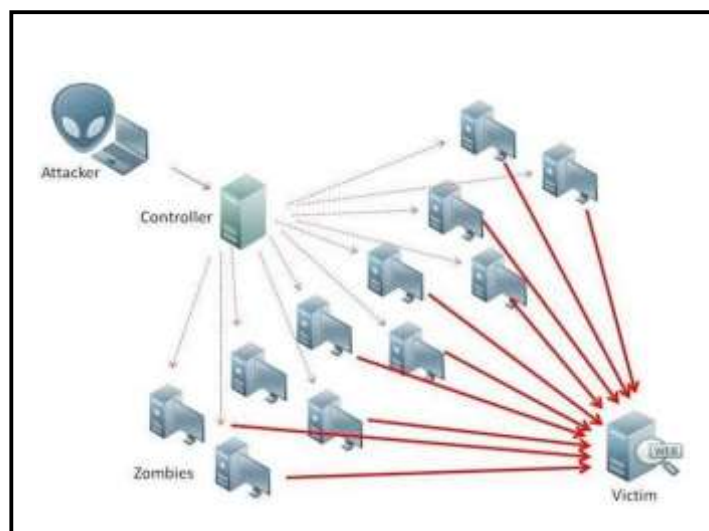


Fig 1.1 Conceptual diagram of DDoS

## 1.2 DDoS Attack Introduction

A real attacker deploys daemon attack programs in multiple host computers, and deploys a master program, that controls and coordinate the daemons, in another host computer. When the real attacker wants to launch an attack, an execute command is sent to the control master program which will then execute all the daemons under its control. After that, the daemons will attack the victim. The four main components of an attack are:

1. A real attacker
2. A control master program
3. Attack daemon agents
4. The victim

The general structure of a DDoS attack is shown in Figure 1.2 below.

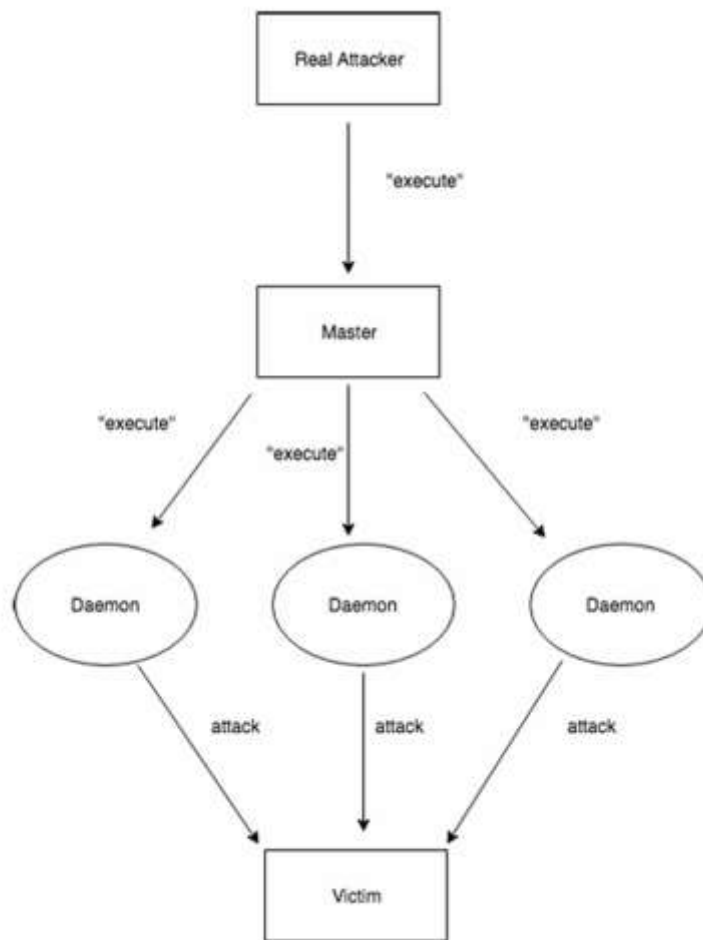


Fig 1.2 General structure of DDoS Attack

### 1.2.1 Types of Attacks

There are various types of attack which target different services, such as

- Flooding system traffic which leads to service denial to legitimate users
- Connection disruption between two machines, thereby preventing access to a service
- Preventing a particular system or user from accessing a service

### 1.2.2 DDoS Attack Taxonomy

DDoS attacks are different in terms of how they are started and the impact they have on the target server. But also, they all share the same target of interfering with legitimate traffic. IoT networks and devices are different in nature. As a result, there will be a different threats focused specifically on them.

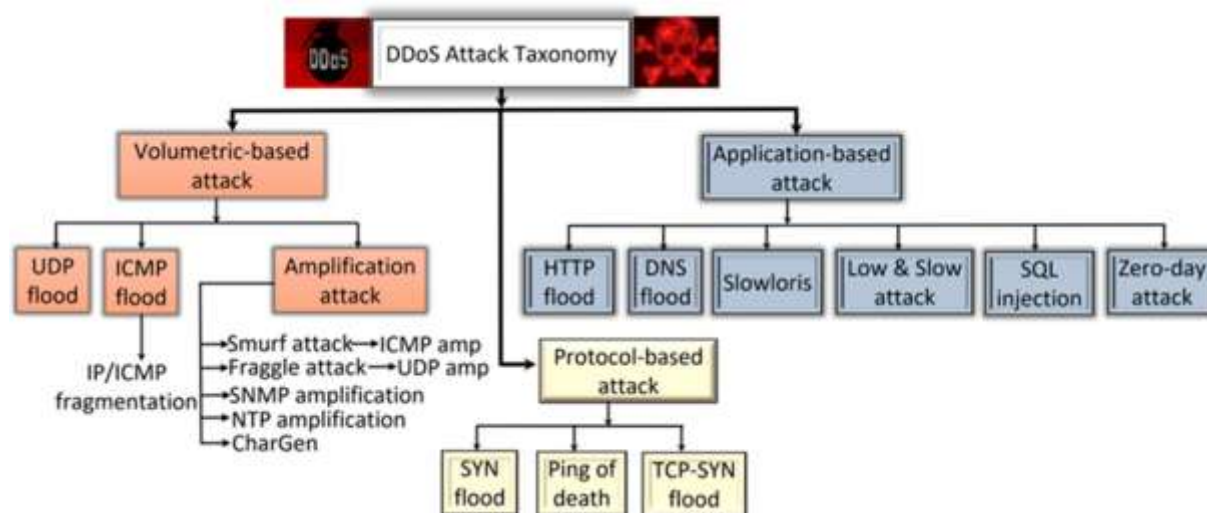


Fig 1.3 Major categories of DDoS attacks

### 1.2.3 Volumetric-Based DDoS Attacks

Volumetric DDoS attack floods the target network's available bandwidth with huge data packets, by overwhelming it. The attack overwhelms the targeted network with abnormally more amounts of malicious traffic to deny service to authorized users. Any server that cannot handle the more traffic volume can be brought down instantly by such attacks. The general structure of a conventional volumetric based DDoS attack is illustrated in Figure 1.4.

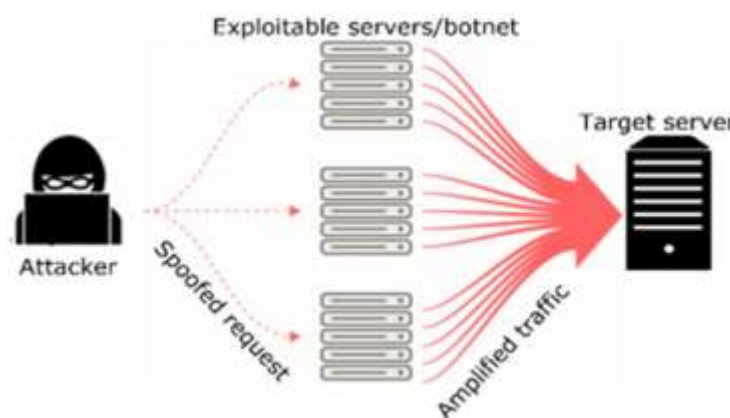


Fig 1.4 Conventional structure of a volumetric-based DDoS attack

The attackers' goal, as depicted in Figure 1.4, is to overwhelm the bandwidth of a target site by sending as much traffic as they can. Attackers mostly use exploited amplification strategies, which involve sending brief true requests to a

domain name server (DNS) with a spoof source IP address of the target to overwhelm the target server. Volumetric attacks are identified by a more amount of traffic (100 GB/s or more). A reflection medium can be used to produce gigabits of traffic from a little amount of traffic. Examples of volumetric-based DDoS attacks include, network time protocol (NTP), internet control message protocol (ICMP) flooding, simple network message protocol (SNMP) amplifications, user datagram protocol (UDP) flooding, character generator (CharGen) among others.

### 1.2.4 DDoS attack detection methodologies

**Traditional methods:** These focuses on measuring the traffic volume. When the measured traffic volume is more than a predetermined level, a DDoS attack is found.

**Signature-based detection:** This method uses the data that are kept in a database to find attacks that is attack signatures. It involves finding traffic patterns and comparing them to pre- existing signatures. Any differences from the previously recorded patterns indicate fake traffic. It says that only attacks whose signatures have been previously stored in the database can be detected. The method has high accuracy in finding known attacks, provided the database is updated. However, any deviation from the attack signatures or novel attack pattern cannot be found.

**Anomaly-based detection:** This method involves collecting typical traffic behavior over a fixed period and creating a baseline profile. Any incoming pattern that is outside the scope of the baseline viewed as an anomaly, which suggests that attacks have taken place.

## 1.3 Objectives

- **Streamlining Complexity:** Deep learning-based approaches aim to simplify network architectures by providing more efficient and integrated solutions, reducing the need for complex systems and easing deployment and maintenance processes.
- **Adaptability to Evolving Threats:** Deep learning algorithms offer the flexibility to address the changing nature of cyber threats, particularly DDoS attacks. Leveraging deep learning's adaptability, these approaches aim to stay ahead of evolving threats and maintain robust defenses against DDoS attacks.
- **Reduction of False Positives:** Deep learning algorithms for DDoS detection target the reduction of false positives, ensuring that legitimate traffic is not misclassified as malicious, thereby minimizing unnecessary interventions and optimizing resource allocation.
- **Scalability:** Deep learning models can be designed to scale efficiently, enabling real- time monitoring and analysis of large network traffic volumes in cloud computing environments without sacrificing performance or causing delays.
- **Improvement in Effectiveness, Efficiency & Scalability:** The application of deep learning techniques holds the potential to enhance the overall effectiveness, efficiency, and scalability of DDoS detection systems in complex and dynamic network environment

## 2. SYSTEM ANALYSIS

### 2.1 Existing System

DDoS is very difficult to fight. It is one of the most utilized techniques of attack. The reason for this problem is because the attack appears from several IP address locations throughout the internet at the same time, making it more difficult to pinpoint the source of the attack. DDoS attacks can affect a large number of devices connected to the internet.

The existing system uses hybrid model of conventional machine learning techniques like SVM-KNN-LR achieved an accuracy of 96% comparing to other individual models such as Naive Bayes, Decision Trees. But it is very complex to maintain and update the hybrid model, also requires more computational resources compared to single algorithm. There's a possibility that the error could propagate through the other components, potentially affecting the overall accuracy of the hybrid system.

#### 2.1.1 Disadvantages of Existing System

- **More Complexity:** The existing solution involves multiple stages and mechanisms, which may add complexity to the network architecture and require careful implementation and maintenance.
- **False Positives:** Machine learning algorithms may generate false positives when they classify legitimate traffic as an attack. These false alarms can lead to unnecessary intervention and resource consumption.
- **Scalability:** Cloud computing environments handle large volumes of data and network traffic. Machine learning models need to scale effectively to monitor and analyze this traffic in real-time without causing significant delays.
- **Algorithm:** Support Vector Machine (SVM), K-Nearest Neighbour (KNN), Logistic Regression (LR).

### 2.2 Motivation to the problem

The motivation behind addressing the challenges posed by Distributed Denial of Service (DDoS) attacks lies in their pervasive and detrimental impact on network infrastructure and connected devices. DDoS attacks are notorious for their complexity and ability to disrupt services by flooding targets with malicious traffic originating from various IP addresses across the internet simultaneously. Despite efforts to combat DDoS attacks using a hybrid model of conventional machine learning techniques, inherent complexities in maintaining and updating such systems, coupled with the evolving nature of attack tactics and the potential for false positives, pose significant hurdles. Moreover, as cloud computing environments handle ever-increasing volumes of data and network traffic, the scalability of existing machine learning models becomes a pressing concern. Addressing these challenges is paramount to safeguarding network integrity, preserving service availability, and mitigating the disruptive effects of DDoS attacks on critical systems and infrastructure.

### 2.3 Proposed System

In the proposed system a deep learning based DDoS attack detection approach that can automatically extract high-level features from low-level ones and gain powerful representation and inference. We design a Deep Neural Network (DNN) to learn patterns from sequences of network traffic and trace network attack activities. We train a deep neural network to

classify normal and DDoS attack states by using a carefully chosen set of network statistics as an input signal. The accuracy of our proposed model was observed to be higher than the hybrid model.

### 2.3.1 Advantages of Proposed System

- **Streamlining Complexity:** Deep learning-based approaches aim in reducing the need for complex systems and easing deployment and maintenance processes.
- **Reduction of False Positives:** Deep learning algorithms for DDoS detection target the reduction of false positives, ensuring that legitimate traffic is not misclassified as malicious.
- **Improvement in Effectiveness, Efficiency & Scalability:** The application of deep learning techniques holds the potential to enhance the overall effectiveness, efficiency, and scalability of DDoS detection systems in complex and dynamic network environments.

### 2.3.2 Algorithm of DNN

Input:

Training dataset  $X$ ,  $y$  Test dataset  $X_{\text{test}}$

Output:

Predictions  $y_{\text{pred}}$

**Train the DNN model on the training dataset  $X$ ,  $y$ :**

Initialize the DNN model

Define the architecture of the model.

Set the parameters such as the learning rate, batch size, and number of epochs. Compile the DNN model with appropriate optimizer, loss function, and evaluation metrics.

Fit the DNN model to the training data

**Make predictions on the test dataset  $X_{\text{test}}$ :**

Predict the class labels for the test dataset using the trained DNN model. Store the predicted labels in  $y_{\text{pred}}$

**Evaluate the performance of the DNN model:**

Calculate the accuracy, precision, recall, and F1 score of the model using the predicted labels and the true labels of the test dataset.

## 2.4 Summary

The current way of dealing with DDoS attacks uses a mix of different computer techniques. While it's quite effective, it's also complicated to manage and needs a lot of resources. DDoS attacks are a big problem because they flood websites with fake traffic from lots of different places, making them hard to stop. These attacks can cause big problems for internet-connected devices. So, there's a need to find a better way to deal with them. That's where the idea of using deep learning comes in. Instead of the mix of techniques we use now, deep learning focuses on using one powerful method called Deep Neural Networks (DNN). This new approach is simpler, reduces mistakes, and makes it easier to manage. Plus, it's more effective at stopping DDoS attacks.



### 3. LITERATURE REVIEW

#### 3.1 Related Works

##### 3.1.1 Related Work-1: DDoS Attack Detection Method Based on Improved KNN With the Degree of DDoS Attack in Software-Defined Networks

**Authors :** Shi Dong and Mudar Sarem

The Distributed Denial of Service (DDoS) attack has seriously impaired network availability for decades and still there is no effective defense mechanism against it. However, the emerging Software Defined Networking (SDN) provides a new way to reconsider the defense against DDoS attacks. In this paper, we propose two methods to detect the DDoS attack in SDN. One method adopts the degree of DDoS attack to identify the DDoS attack. The other method uses the improved K- Nearest Neighbors (KNN) algorithm based on MachineLearning (ML) to discover the DDoS attack. The results of the theoretical analysis and the experimental results on datasets show that our proposed method scan better detect the DDoS attack compared with other methods.

**Pros:** 1) Real time detection,

2) Adaptability

**Cons:** 1) Storage requirements,

2) Limited anomaly detection,

3) Accuracy-89%

##### 3.1.2 Related Work-2: DeMi: A Solution to Detect and Mitigate DoS Attacks

**Authors:** Lubna Fayez Eliyan and Roberto Di Pietro

Software-defined networking (SDN) is gaining popularity due to its scalability and flexibility, which simplifies network management and enables innovations in network architecture and protocols. In SDNs, the controller plays a crucial role in managing the entire network and configuring routes. However, the controller is vulnerable to threats such as denial-of-service (DoS) attacks, which can seriously affect SDN operations. To combat this threat, we propose a lightweight method called DeMi to detect and mitigate DoS attacks and a heavy load management module. Our detection method uses a sample entropy approach with adaptive dynamic thresholding based on an exponentially weighted moving average (EWMA). The mitigation method combines proof of work (PoW) with flow rules, while heavy load management is implemented with scheduling in the SDN controller. Our results are impressive: with DeMi implemented, the number of control packets exchanged during the attack scenario is similar to that of the non-attack scenario, while without DeMi the number of control packets increases by a factor of 2.7. Similarly, DeMi achieves a retransmission rate comparable to the no-attack scenario, while without DeMi, the number of retransmitted packets is 3.7 times higher. Importantly, DeMi does not block legitimate traffic, which distinguishes it from other solutions in the literature. The novelty of our approach, the comprehensive end-to-end solution, and the high quality of the experimental results pave the way for further research in this area.



**Pros:** 1) Comprehensive Solution

2) Improved Network Performance

**Cons:** 1) Complex Configuration,

2) Evaluation Context

3) Accuracy-91%

### **3.1.3 Related Work-3:** Low-Rate DDoS Attack Detection Based on Factorization Machine in Software Defined Network

**Authors:** Wuzhijun, Xu Qing, Wang Jingjie, Yue Meng and Liu Liang

Because Software Defined Network (SDN) uses centralized control logic, it is vulnerable to various DDoS attacks. Currently, almost all research focuses on a fast DDoS attack against the SDN control layer. In addition, most of the existing detection methods are effective in detecting a high-speed DDoS attack on the control layer, while a low-speed DDoS attack on the SDN data layer is well hidden and the detection accuracy against it attack is low. In order to improve the detection accuracy of a low-speed DDoS attack against the SDN data layer, this paper investigates the mechanism of such attacks and then proposes a multi-functional DDoS attack detection method based on a Factorization Machine (FM). Features extracted from flow rules are used to detect low-speed DDoS attacks and detect high-speed DDoS attacks based on FM machine learning algorithms. Experimental results show that the method can effectively detect a low-speed DDoS attack against the SDN data layer with a detection accuracy of 95.80 percent. Because the FM algorithm can detect low-speed DDoS attacks, which provides a reliable condition to protect against such attacks. Finally, this paper proposes a protection method based on the dynamic removal of flow rules and conducts experimental simulation and analysis to prove the effectiveness of the protection method and the success rate of conventional packet has reached 97.85 percent.

**Pros:** 1) Effective Defense Strategy

2) Novel Research Area

**Cons:** 1) Limited Exploration of Deep Learning

2) Theoretical Testing

3) Accuracy – 92%

### **3.1.4 Related Work-4:** A Flexible SDN-Based Architecture for Identifying and Mitigating Low-Rate DDoS Attacks Using Machine Learning

**Authors:** Jesús Arturo Pérez-Díaz<sup>1</sup>, Ismael Amezcua Valdovinos, Kim-Kwang Raymond Choo and Dakai Zhu

Although there has been extensive research on Denial of Service (DoS) attacks and DDoS mitigation, mitigating such attacks remains difficult. For example, low-rate DDoS (LR-DDoS) attacks can be difficult to detect, especially in Software-Defined Networking (SDN). Specifically, we train an intrusion detection system (IDS) in our architecture with six machine learning (ML) models (i.e, J48, random tree, REP tree, random forest, multilayer perceptron(MLP), and Support Vector Machines(SVM)) and their performance is evaluated by the Cybersecurity Institute of Canada (CIC) using the DoS dataset. Test results show that our approach gains a detection rate of 95%. Also note that our implementation uses an Open Network Operating System (ONOS) driver running in a Mininet virtual machine to make our simulation environment as close as possible to real production networks. In our test topology, the intrusion

prevention system. mitigates all the IDS system attacks previously detected by This demonstrates the utility of our architecture to detect and mitigate LR-DDoS attacks.

**Pros:** 1) Modularity  
2) Flexibility  
3) Real world deployment

**Cons:** 1) Limited Evaluation on ML Models  
2) Scalability Challenges  
3) Accuracy – 95%

### **3.1.5 Related Work-5: Real-Time Detection Schemes for Memory DoS (M-DoS) Attacks on Cloud Computing Applications**

**Authors:** Umar Islam Muhammadahsan, Abdullah Al-Atawi, Salamah Alwageed, Fuad A. Awwad and Mohamed R. Abonazel

Memory Denial of Service (M-DoS) attacks refer to a set of cyberattacks designed to drain system memory resources and make them available to legitimate users. This type of attacks are especially dangerous in cloud computing environments where multiple users share the same resource. Real-time detection and mitigation of M-DoS attacks is a difficult task because they often involve a large number of low- rate requests, making them difficult to distinguish from legitimate traffic. Several real-time detection systems have existed offered to detect and mitigate M-DoS attacks in cloud computing environments. These systems can be broadly divided into two categories: signature-based and anomaly-based detection. Signature-based detection methods are based on detecting certain patterns or characteristics of known M-DoS attack techniques, while anomaly-based detection methods detect abnormal behavior that deviates from the normal usage pattern. The data set used in this study was collected from various sources and pre-processed to extract the features necessary to detect the attack. The feature selection process was also used to identify the most important attack detection features. The hybrid model achieved 96% accuracy, outperforming other individual models such as SVM, KNN, LR, NaiveBayes, Decision Trees, Extra Trees, Sack Trees and Random Forests. In the Discussion, we looked at the performance of the hybrid model in detecting MDOS attacks and found that it has an accuracy of 0.97. However, the recovery score was lower at 0.87, indicating that the model was unable to detect all MDOS attacks.

**Pros:** 1) Accuracy,  
2) Real time detection,  
3) Adaptability

**Cons:** 1) Very complex to maintain  
2) Requires more computational resources  
3) Accuracy – 96%

## **3.2 Challenges**

- **More Complexity:** The existing solution involves multiple stages and mechanisms, which may add complexity to the network architecture and require careful implementation and maintenance.

- **Attack Evolution:** Attackers are continually evolving their tactics and techniques, which means that the data used for training machine learning models might not adequately cover the latest attack strategies.
- **False Positives:** Machine learning algorithms may generate false positives when they classify legitimate traffic as an attack. These false alarms can lead to unnecessary intervention and resource consumption.
- **Scalability:** Cloud computing environments handle large volumes of data and network traffic. Machine learning models need to scale effectively to monitor and analyze this traffic in real-time without causing significant delays.

### 3.3 Problem Statement

We propose a deep learning-based DDoS attack detection approach to replace the complex and resource-intensive existing hybrid model. Deep learning allows for automatic high-level feature extraction, reducing the risk of error propagation and enhancing accuracy. We utilize a Deep Neural Network (DNN) to learn patterns from network traffic sequences and detect network attack activities.

### 3.4 Summary

After the literature review by studying several research papers we found many limitations in the models of conventional and machine learning. To overcome the challenges like more complexity, increased false positive rate, low efficiency, effectiveness and less scalability we propose a deep learning model i.e., Deep Neural Networks (DNN) to mitigate the DDoS attacks by doing high-level feature extraction, reducing the risk of error propagation and enhancing accuracy. We utilize a DNN to learn patterns from network traffic sequences and detect network attack activities.

## 4. SYSTEM REQUIREMENT SPECIFICATION

### 4.1 Requirement Analysis

Software engineering's requirement analysis task bridges the gap between software allocation at the system level and software design. The purpose of requirement analysis is to identify how software interacts with other system components and their functions. Prerequisite Examination of the framework begins with the detail given by the client. This client required particular could be formal or casual. The user clearly explains the software's purpose in the formal method. The software engineers can use this as a solid foundation for their requirement analysis. The purpose and outputs of an informal method are not clearly specified by the user. The obligation is on the computer programmer to get the reason and results by connecting more the client.

Requirement Analysis of the system starts with the specification given by the user. This user-required specification could be formal or informal. In normal method, the user clearly states the purpose of the software. This acts as the good basis for the software engineers for the requirement analysis.

### 4.2 Software Requirements Specification (SRS)

A software requirements specification, also known as a requirements specification for a software system, is a comprehensive description of the behavior of the system that is going to be developed. It may also include a collection

of use cases that describe how the user will interact with the software. What's more it additionally contains non-useful prerequisites. The design or implementation is restricted by non-functional requirements. All of the necessary requirements for the project's development are listed in the software requirements specification document.

We need to have a thorough understanding of the products that need to be developed in order to come up with the requirements. This is ready after definite correspondences with the undertaking group and client.

In addition to reducing development costs, a Software Requirements Specification reduces the amount of time and effort required by developers to accomplish desired objectives. In a wide range of real-world scenarios, a good SRS specifies how an application will interact with system hardware, other programs, and human users.

### **4.3 Functional Requirements**

#### **Attack Detection:**

It should be capable of identifying and distinguishing between normal network traffic and malicious DDoS attack traffic using appropriate algorithms and techniques.

#### **Mitigation Strategies:**

It should have mechanisms in place to mitigate the impact of detected DDoS attacks, such as rerouting traffic, blocking suspicious IP addresses, or implementing rate limiting measures.

#### **Scalability:**

The system should be able to handle large volumes of network traffic and scale effectively to accommodate growing network infrastructure and traffic loads.

### **4.4 Non-Functional Requirements**

#### **Performance:**

The system should be capable of efficiently processing and analyzing large volumes of network traffic in real-time to detect and mitigate DDoS attacks without significant delays or performance degradation.

#### **Reliability:**

The system should be highly reliable, with minimal downtime and a low probability of false positives or false negatives in DDoS attack detection.

#### **Security Requirements:**

The system itself should be secure, with appropriate measures in place to prevent unauthorized access, tampering, or exploitation by malicious actors. Additionally, it should not introduce vulnerabilities into the network infrastructure it is designed to protect.

**Performance Metrics Requirement:**

The system should include mechanisms for monitoring and measuring key performance metrics, such as detection accuracy, response time, and resource utilization, to evaluate its effectiveness and identify areas for improvement.

**Software Quality Attributes:**

Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability.

**4.5 Software Requirements**

- Software: Jupyter Notebook
- Programming Language: Python 3.9
- Dataset: Mendeley Data
- Operating System: Windows 11

**4.6 Hardware Requirements**

- Hard Disk Drive: 10GB
- Processor: Intel Core i3
- Main Memory: 128MB

**4.7 Summary**

Overall, the system uses a combination of dataset that is downloaded from Mendely Data website, machine learning and deep learning algorithms for real time detection of DDoS attacks under challenging patterns of traffic packets. The syatem is designed in such a way that it can train the new data and detect the attack. It is designed to mitigate the DDoS attacks and drop the fake traffic so that the server doesn't go down the safety and efficiency of the website is improved.

## 5. SYSTEM DESIGN

### 5.1 System Architecture

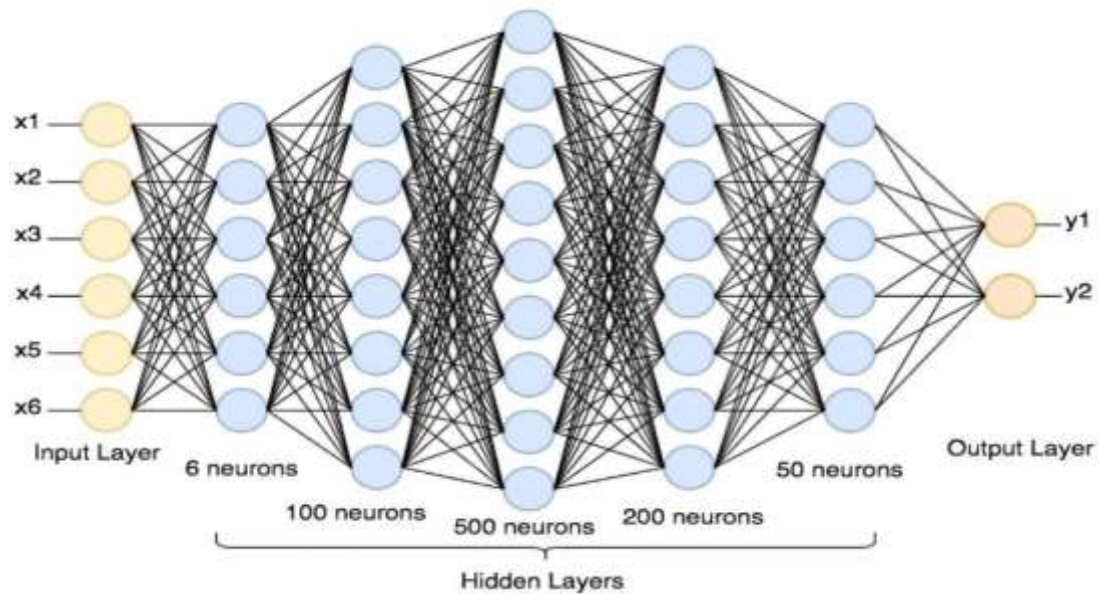


Fig 5.1 DNN Architecture

### 5.2 Data flow diagram

- The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
- The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system
- DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
- DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

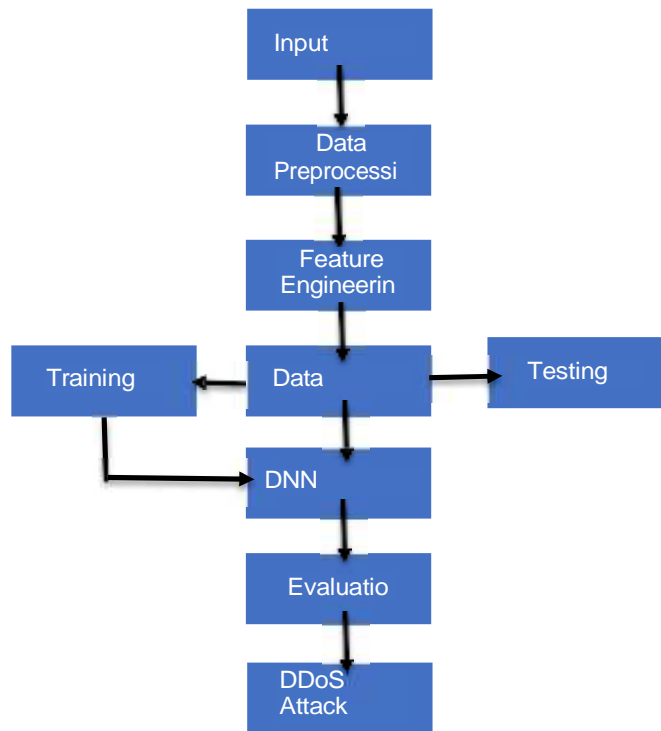


Fig 5.2 Proposed Work flow diagram

### 5.3 UML (Unified modeling Language)

UML stands for Unified Modeling Language which is used in object oriented software engineering. Although typically used in software engineering it is a rich language that can be used to model an application structures, behavior and even business processes. which is designed to provide a standard way to visualize the design of a system.

It was created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software during 1994–95 with further development led by them through 1996. In 1997 it was adopted as a standard by the Object Management Group (OMG), and has been managed by this organization ever since. In 2000 the Unified Modeling Language was also accepted by the International Organization for Standardization (ISO) as an approved ISO standard.

#### 5.3.1 Goals

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.



- Integrate best practices.

### 5.3.2 Class Diagram

Class diagrams are arguably the most used UML diagram type. It is the main building block..1 of any object oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between each class. In most modeling tools a class has three parts, name at the top, attributes in the middle and operations or methods at the bottom.

### 5.3.3 Component Diagram

A component diagram displays the structural relationship of components of a software system. These are mostly used when working with complex systems that has many components. Components communicate with each other using interfaces . The interfaces are linked using connectors.

### 5.3.4 Deployment Diagram

A deployment diagrams shows the hardware of your system and the software in those hardware. Deployment diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration.

### 5.3.5 Object Diagram

Object Diagrams, sometimes referred as Instance diagrams are very similar to class diagrams. As class diagrams they also show the relationship between objects but they use real world examples. They are used to show how a system will look like at a given time.

### 5.3.6 Use Case Diagram

Most known diagram type of the behavioral UML diagrams, Use case diagrams gives a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted. It's a great starting point for any project discussion because you can easily identify the main actors involved and the main processes of the system.

### 5.3.7 Activity Diagram

Activity diagrams represent workflows in an graphical way. They can be used to describe business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine diagrams.

### 5.3.8 State Machine Diagram

State machine diagrams are similar to activity diagrams although notations and usage changes a bit. They are sometime known as state diagrams or start chart diagrams as well. These are very useful to describe the behavior of objects that act different according to the state they are at the moment.

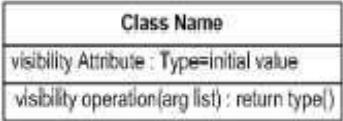
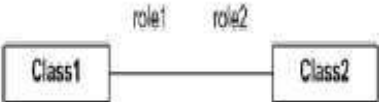






### 5.3.9 Sequence Diagram







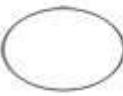




Sequence diagrams in UML shows how object interact with each other and the order those interactions occur. It's important to note that show the interactions for a particular scenario. The processes are represented vertically and interactions are show as arrows.

### 5.3.10 Collaboration Diagram

It is similar to sequence diagrams but the focus is on messages passed between objects. The same information can be represented using a sequence diagram and different objects.

## 5.4 List of UML Notations

S.NO	SYMBOL NAME	SYMBOL	DESCRIPTION
1	Class		Classes represent a collection of similar entities grouped together.
2	Association		Association represents a static relationship between classes.
3	Aggregation		Aggregation is a form of association. It aggregates several classes into single class.
4	Actor		Actors are the users of the system and other external entity that react with the system.
5	Use Case		A use case is an interaction between the system and the external environment.
6	Relation (Uses)		It is used for additional process communication.
7	Communication		It is the communication between various use cases.
8	State		It represents the state of a process. Each state goes through various flows.

9	Initial State		It represents the initial state of the object.
10	Final State		It represents the final state of the object.
11	Control Flow		It represents the various control flow between the states.
12	Decision Box		It represents the decision making process from a constraint.
13	Component		Components represent the physical components used in the system.
14	Node		Deployment diagrams use the nodes for representing physical modules, which is a collection of components.
15	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
16	External Entity		It represent any external entity such as keyboard, sensors etc.
17	Transition		It represents any communication that occurs between the processes.
18	Object Lifeline		Object lifelines represents the vertical dimension that objects communicates.
19	Message		It represents the messages exchanged.

## 5.5 UML Diagrams

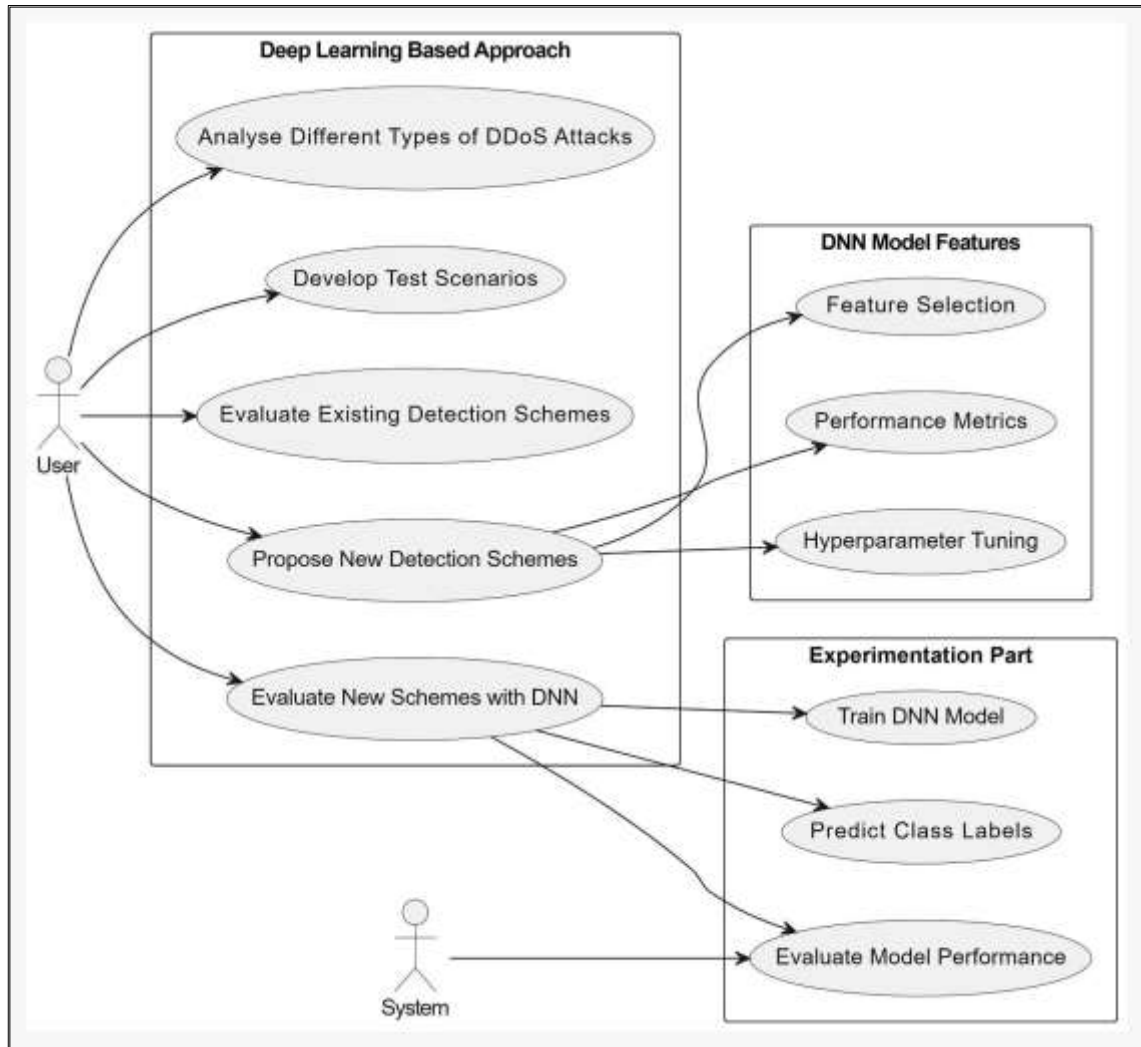
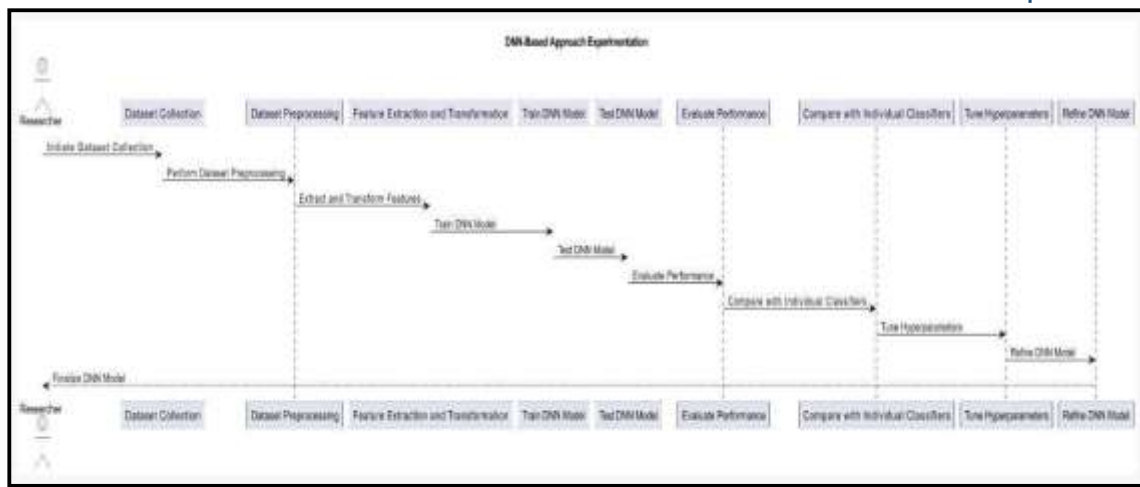


Fig 5.3 Use case Diagram for System and User



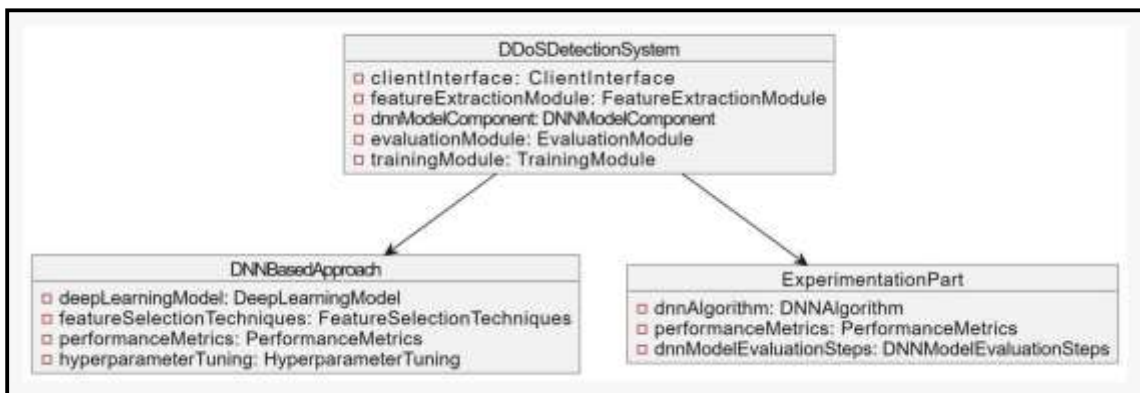
Fig 5.4 Class Diagram for DDoS detection System



```

graph TD
    subgraph TopRow [ ]
        direction LR
        T1[Deep Learning Based Approach]
        T2[Feature Selection Techniques]
        T3[Effective Performance Metrics]
        T4[Hyperparameter Tuning]
        T5[Dataset]
    end
    subgraph BottomRow [ ]
        direction LR
        B1[Deep Learning Based Approach]
        B2[Feature Selection Techniques]
        B3[Effective Performance Metrics]
        B4[Hyperparameter Tuning]
        B5[Dataset]
    end
    T1 -.-> B2
    T1 -.-> B3
    T1 -.-> B4
    T1 -.-> B5

```



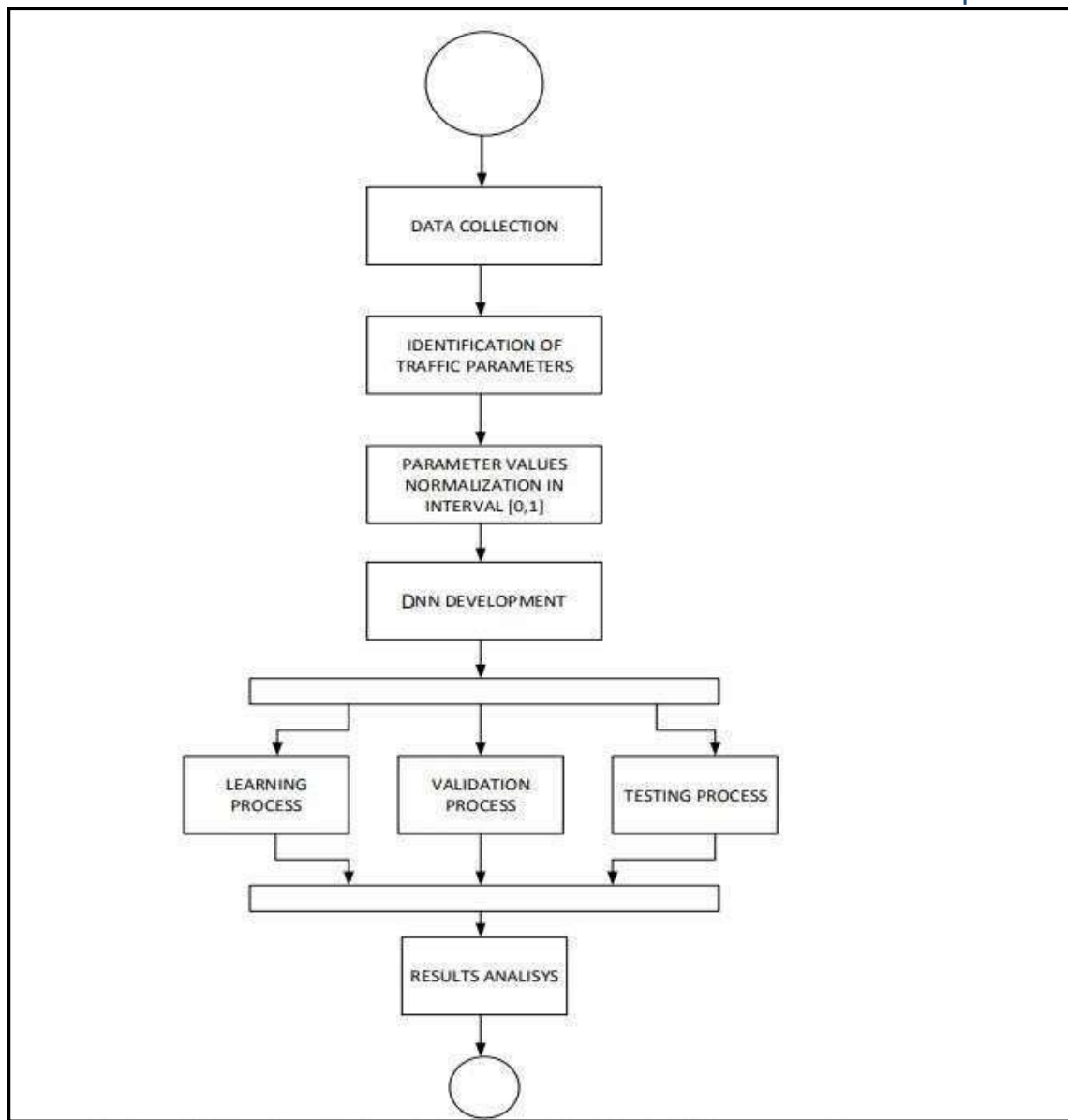


Fig 5.8 Activity Diagram for Proposed Model

## 6. SYSTEM IMPLEMENTATION

### Modules

1. Preprocessing Module
2. Feature Extraction and Transformation
3. Model Implementation
4. Results

### 6.1 Preprocessing Module

**Dataset Preprocessing:** Dataset pre-processing refers to the steps taken to clean, transform, and prepare raw data for analysis. In the context of M-DoS attack detection, the dataset pre-processing stage involves transforming the raw SDN dataset into a suitable format for use with machine learning algorithms. This typically involves removing irrelevant or duplicate data points, normalizing the data to a consistent scale, and encoding categorical variables into numerical

representations.

a: Data Cleaning

Let  $D = \{d_1, d_2, \dots, d_n\}$  be the raw dataset consisting of  $n$  data points. The cleaning process involves identifying and removing any missing or irrelevant data points. This can be represented as:

$D' = \{d'_1, d'_2, \dots, d'_m\}$ , where  $m \leq n$ . b: Data Normalization

Normalization involves transforming the data values to a common scale.

## 6.2 Feature Extraction and Transformation

In the context of detecting DDoS attacks using DNN, feature extraction plays a crucial role in preparing the input signal for the neural network.

1. **Numerical Features:** Numerical features are essential components of datasets and play a crucial role in data analysis, modelling, and machine learning tasks by providing quantitative information that can be used to derive insights and make informed decisions. Some example numerical features might include 'pktcount' (number of packets), 'bytecount' (number of bytes), 'dur' (duration), 'pktrate' (packet rate), 'tx\_bytes' (transmitted), 'rx\_bytes' (received bytes) etc.,
2. **Categorical Features:** Categorical features allow for the grouping and categorization of data into distinct classes or groups based on shared attributes or characteristics. These are often encoded or transformed into numerical representations before being used as input for machine learning models. Attributes like source addresses, destination addresses, and communication protocols provide valuable insights into the behaviour and patterns of network communication.
3. **Continuous Features:** Continuous features are numerical variables that can represent measurements or quantities that can take on any value within a specified range. They allow for more precise and detailed analysis of data compared to discrete features often exhibit a smooth and uninterrupted distribution when visualized. This includes measurements like dt, pktcount, bytecount, dur, pktrate, etc. The unique numerical features are shown in the table below which are used to visualize the ddos attacks from the dataset.



Feature	Unique values
dt	858
switch	10
pktcount	9044
bytecount	9270
dur	840
dur_nsec	1000
tot_dur	4183
flows	15
packetins	168
pktperflow	2092
byteperflow	2793
pktrate	446
Pairflow	2
port_no	5
tx_bytes	12257
rx_bytes	11623
tx_kbps	1800
tot_kbps	2259
label	2

Table 6.1 Numerical Features

## 6.3 Model Implementation

### 6.3.1 Python (3.9.0)

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file –

Live Demo

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows –

```
$ python test.py
```

This produces the following result – Hello, Python!

### **6.3.2 Libraries Tensorflow**

TensorFlow enables developers to build dataflow graphs, which are structures describing how data flows through a graph or a sequence of processing nodes, with each node representing a mathematical operation. Every node in a dataflow graph is a tensor, and each node's connection or edge is a multi-dimensional data array.

The nodes and tensors of TensorFlow are Python objects. TensorFlow apps also are Python applications. The actual mathematical operations are done in Python, but the libraries of transformations that are available through TensorFlow are written as high - performance C++ binaries. Python manages the traffic between nodes and tensors, but provides the abstractions that connect them together.

### **Numpy**

Numpy in Python is a general-purpose package for dealing with arrays. It provides a multi- dimensional array object with excellent capabilities and speed for dealing with these arrays. Python NumPy is the foundation of the scientific computing module. It is also an open- source library.

Numpy is a powerful multi-dimensional data container with many non-science applications. Numpy can declare any data type, making it easy for it to interact with different databases. Python supports arrays with different number of dimensions, making data management efficient and versatile. Broadcasting is a powerful and reliable method for executing operations on arrays of different sizes and shapes.

### **Pandas**

Pandas is a Python library for data analysis that was created by Wes McKinney in 2008. It was inspired by McKinney's desire to create a powerful and versatile tool for quantitative analysis. Pandas is one of the most popular Python libraries, with an active community of contributors.

Pandas is based on matplotlib, which is used to visualize data, and numpy, which is used for mathematical operations. Pandas acts as a wrapper for both of these libraries. It provides many of the methods found in matplotlib in a much shorter amount of code. For instance, the `plot().plot()` method in Pandas combines multiple matplotlib methods into a single method, allowing you to plot a chart in just a few lines.

### **Matplotlib**

Matplotlib (or Matplotlib) is a Python-based plotting library. It is one of the most popular Python visualization packages. Matplotlib is extremely fast at a wide range of operations.

It is also capable of exporting visualizations to a wide range of image formats, such as PDFs, SVG's, JPGs, PNGs, BMPs, and GIFs.

It can also be used to create 3D charts. Matplotlib, the open-source Python plotting toolkit, is the most popular Python plotting library. It was founded in 2002 by John Hunter, a post- doctoral student in Neurobiology, with the goal of visualizing data from epilepsy patients.

### Seaborn

Seaborn contains a data visualization library created over matplotlib and integrated with pandas ds in Python. It helps you to make attractive charts with short code. The main part of Python Seaborn is visualization which helps in understanding and exploring data.

**Categorical Plots:** These kinds of plots cater to categorical variables. They also aid to visualize how graphs may be plotted.

**Regression Plots:** The plot is primarily used to include a visual to the data. It also assists in highlighting the patterns of the dataset while analyzing it in the process of data analyses. **Sklearn**

Sklearn is the most powerful and powerful library for Machine Learning in Python. Sklearn provides a variety of powerful tools for Machine Learning and Statistical modeling such as Classification, Cross Validation, Feature Extraction, Feature Selection, etc. Sklearn is mostly written in Python and is based on the popular Python algorithms such as Linear Regression, Support Vector Machine, Decision Tree etc.

This library is open source and can be used commercially under the BSD license. The main features of this library are:

1. Cluster
2. Cross Validation
3. Feature Extraction
4. Feature Selection
5. Unsupervised Learning

## 6.4 Sample Code

- Data Preprocessing

```
print("This Dataset has { } rows and { } columns".format(df.shape[0], df.shape[1])) df.info()
```

```
# Concise summary of dataset
```

```
df.describe() # Descriptive statistics of dataset df.isnull().sum()
```

```
#Count of null values df.dropna(inplace=True) #Drop rows of null values
```

- Numeric features

```
numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O'] print("The number of numerical features is",len(numerical_features),"and they are : \n",numerical_features)
```

- Categorical features

```
categorical_features = [feature for feature in df.columns if df[feature].dtypes == 'O'] print("The number of categorical features is",len(categorical_features),"and they are : \n",categorical_features)
```

- Discrete numerical features

```
discrete_feature = [feature for feature in numerical_features if df[feature].nunique()<=15 and feature != 'label'] print("The number of discrete features is",len(discrete_feature),"and they are : \n",discrete_feature)
```

- Continuous features

```
continuous_feature=[feature for feature in numerical_features if feature not in discrete_feature + ['label']] print("The number of continuous_feature features is",len(continuous_feature),"and they are : \n",continuous_feature)
```

- Separating input and output attributes x = df.drop(['label'], axis=1)

```
y = df['label']
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3) print(X_train.shape, X_test.shape)
```

- Define and compile DNN model model = keras.Sequential()

```
model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
```

```
model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2")) model.add(Dense(1 , activation="sigmoid" , name="Output_Layer")) opt = keras.optimizers.Adam(learning_rate=0.01)
```

```
model.compile( optimizer=opt, loss="binary_crossentropy", metrics=['accuracy']) model.summary()
```

- fit model history\_org = model.fit(

```
X_train, y_train, batch_size=32,
```

```
epochs=100, verbose=2, callbacks=None, validation_data=(X_test,y_test), shuffle=True, class_weight=None,
```

```
sample_weight=None, initial_epoch=0)
```

- Model evaluation

```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print('Accuracy of Deep neural Network : %.2f' % (accuracy*100)) Classifier_accuracy.append(accuracy*100)
```

- Hyperparameter tuning def model\_builder(hp):

```
model = keras.Sequential()
```

```
model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
```

```
model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2")) model.add(Dense(1 , activation="sigmoid" , name="Output_Layer")) opt = keras.optimizers.Adam(learning_rate=0.01)
```

```
model.compile(optimizer=keras.optimizers.Adam(Choice('learning_rate',[1e-2, 1e-3, 1e-4])),
loss='binary_crossentropy', metrics=['accuracy'])
```

```
return model
```

- Classification Report `print(classification_report(y_test, y_pred, target_names = labels))`
- Plotting confusion matrix from `itertools` import `product`

```
def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=plt.cm.Blues):
```

```
plt.figure(figsize=(10,10)) plt.grid(False)
```

```
plt.imshow(cm, interpolation='nearest', cmap=cmap) plt.title(title)
```

```
plt.colorbar()
```

```
tick_marks = np.arange(len(classes)) plt.xticks(tick_marks, classes, rotation=45) plt.yticks(tick_marks, classes)
```

```
cm1 = cm
```

```
if normalize:
```

```
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
cm = np.around(cm, decimals=2) cm[np.isnan(cm)]
```

```
thresh = cm.max() / 2.
```

```
for i, j in product(range(cm.shape[0]), range(cm.shape[1])): plt.text(j, i, str(cm1[i, j])+ " (" + str(cm[i, j]*100)+"%)",
```

```
horizontalalignment="center",
```

```
color="white" if cm[i, j] > thresh else "black") plt.tight_layout()
```

```
plt.ylabel("True label") plt.xlabel("Predicted label")
```

## 6.5 Database table

dt	switch	src	dst	pktscount	bytescount	dur	dur_nsec	tot_dur	flows	packetsize	pktprio	bytespersec	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps	tot_kbps	label	
2	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	3	1.44E+08	1817	0	0	0	0
3	11405	1	10.0.0.1	10.0.0.8	126395	1.35E+08	280	7.34E+08	2.81E+01	2	2940	13531	14424046	451	0	UDP	4	1842	2038	0	0	0	0
4	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	1	1795	1242	0	0	0	0
5	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	2	1888	1492	0	0	0	0
6	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	3	1413	1865	0	0	0	0
7	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	1	1795	1402	0	0	0	0
8	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	4	1865	1413	0	0	0	0
9	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	1	1775	1492	0	0	0	0
10	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	2	1845	1402	0	0	0	0
11	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	4	3.55E+08	4295	16578	0	16578	0
12	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	1	1775	1242	0	0	0	0
13	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	2	1413	1865	0	0	0	0
14	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	1	4047	1.44E+08	0	0	0	0
15	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	4	1865	1413	0	0	0	0
16	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	2	5.85E+08	2586	17594	0	17594	0
17	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	4	1865	1413	0	0	0	0
18	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	2	1795	1402	0	0	0	0
19	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	2	1795	1492	0	0	0	0
20	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	2	4047	1.93E+08	0	6307	6307	0
21	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	1	1879	44410560	0	1818	1818	0
22	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	2	4205	98422684	0	1818	1818	0
23	11425	1	10.0.0.1	10.0.0.8	45304	48294064	130	7.36E+08	2.81E+01	3	2943	13535	14428303	451	0	UDP	3	1413	1865	0	0	0	0
24	11425	1	10.0.0.2	10.0.0.8	90133	96294078	200	7.44E+08	2.81E+01	3	2943	13534	14427244	451	0	UDP	1	1570	1492	0	0	0	0
25	11405	1	10.0.0.2	10.0.0.8	103886	1.11E+08	130	7.47E+08	2.81E+01	3	2943	13531	14426178	451	0	UDP	1	1775	1242	0	0	0	0
26	11405	1	10.0.0.1	10.0.0.8	126395	1.35E+08	280	7.34E+08	2.81E+01	2	2943	13531	14424046	451	0	UDP	3	4768	1.53E+08	0	6400	6400	0
27	11511	1	10.0.0.1	10.0.0.8	85676	91330101	190	7.26E+08	2.81E+01	3	2943	13505	14324186	443	0	UDP	2	1795	1492	0	0	0	0

dataset\_sch

+

Fig 6.1 Database Table

## 6.6 Screenshots

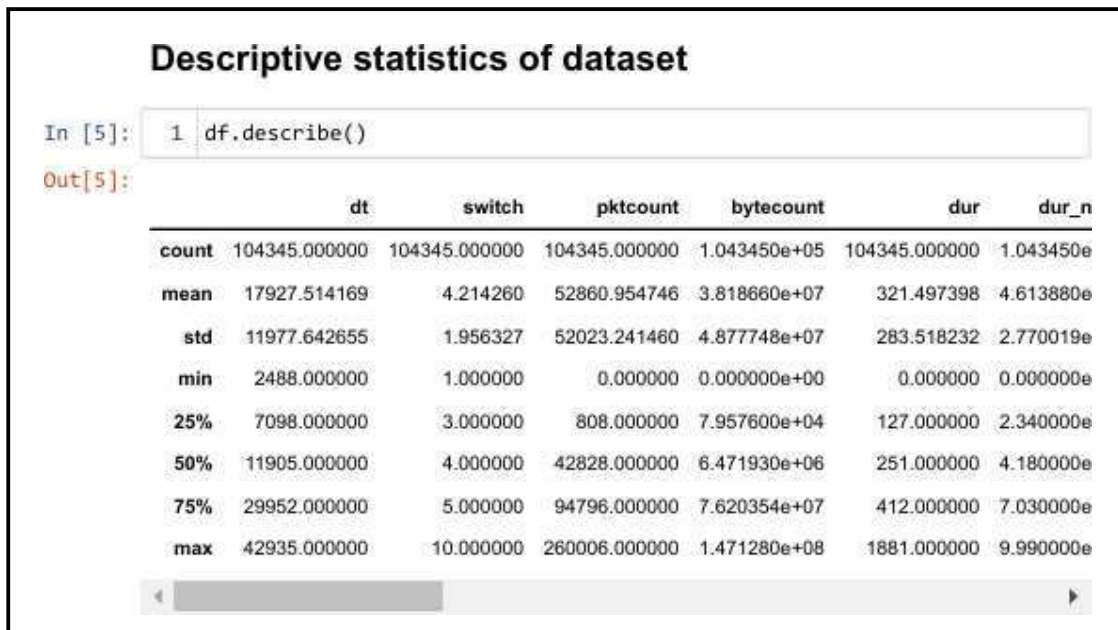


Fig 6.2 Descriptive statistics of dataset

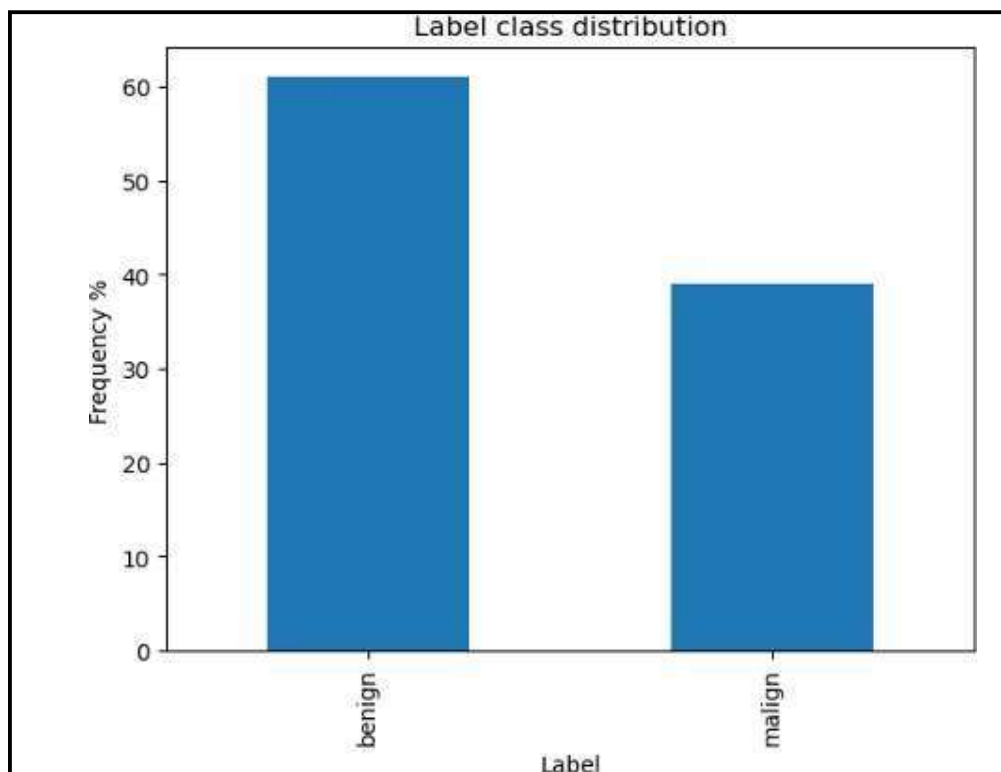


Fig 6.3 Barplot of target class

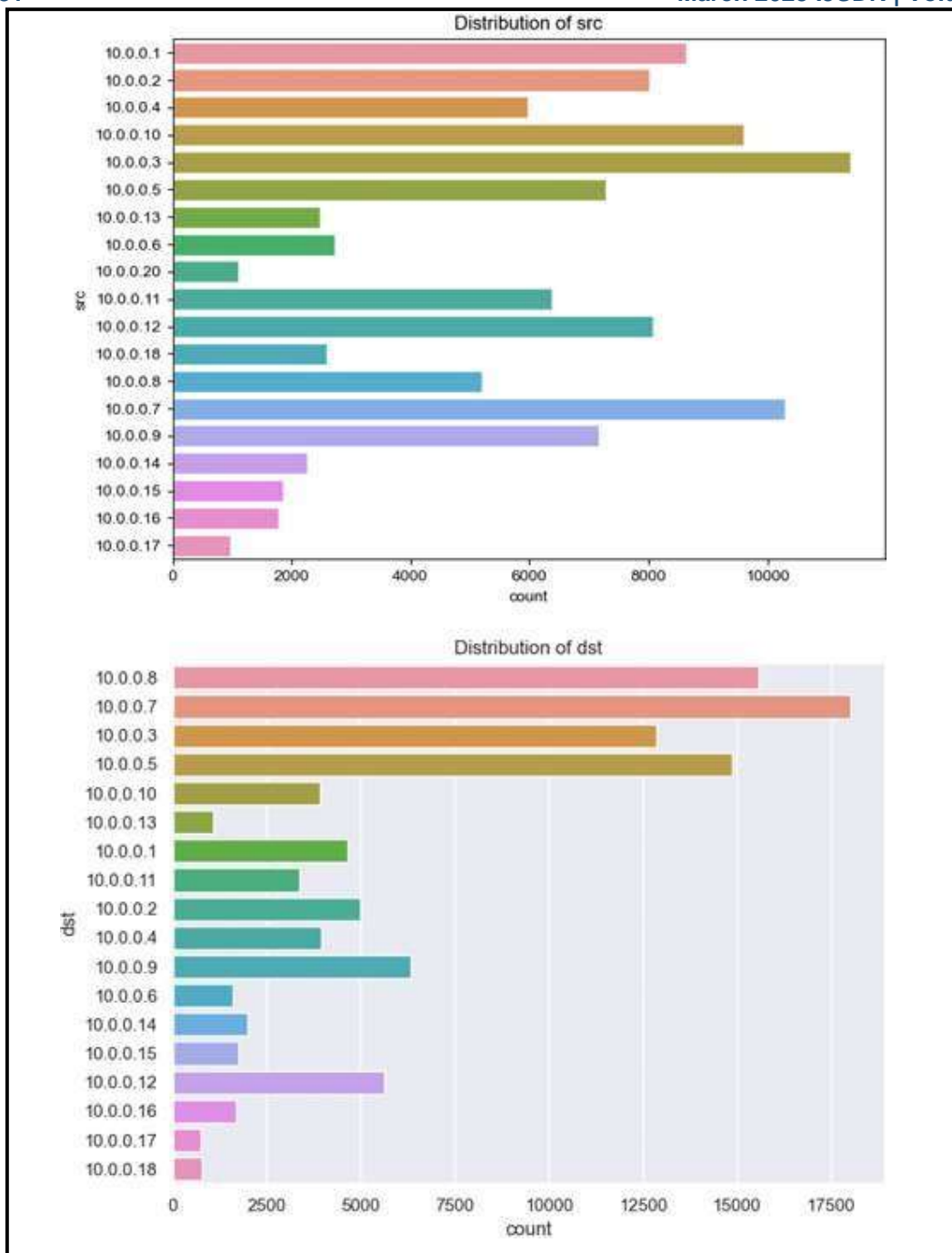


Fig 6.4 Distribution of categorical features



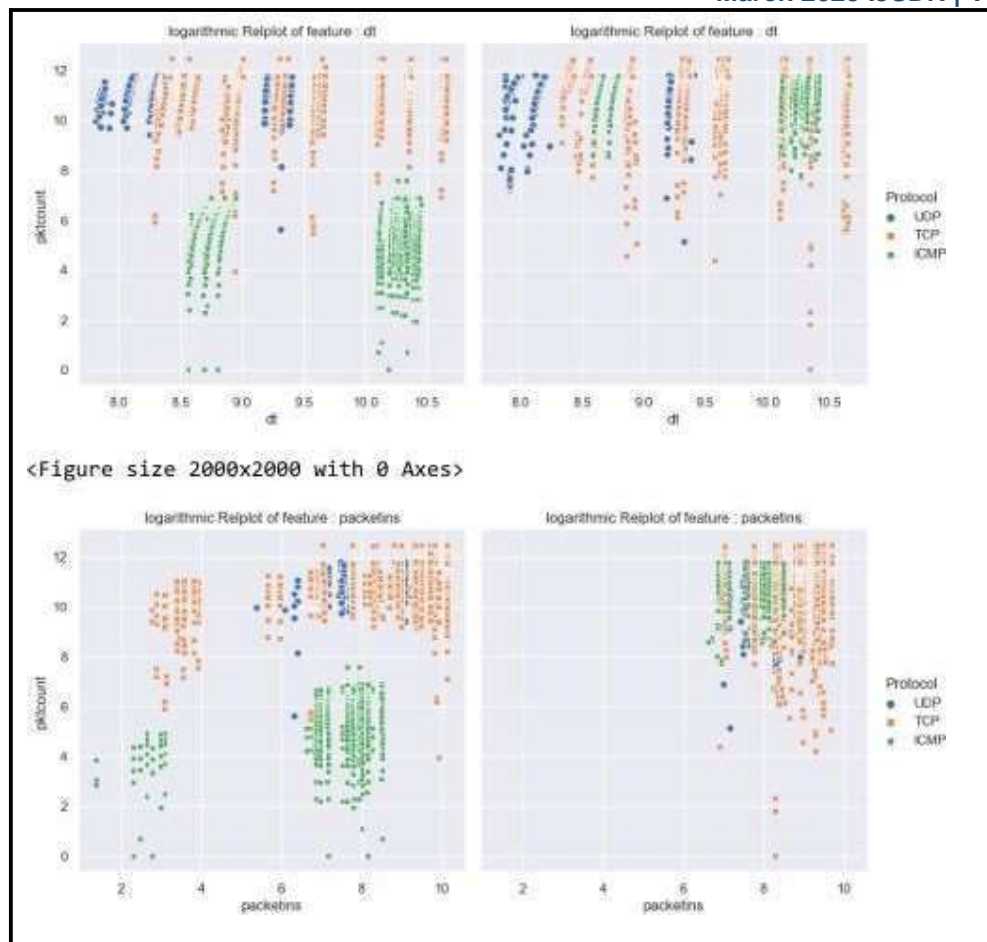


Fig 6.5 Visualize the distribution of continous features wrt packet count,protocol and type of attack

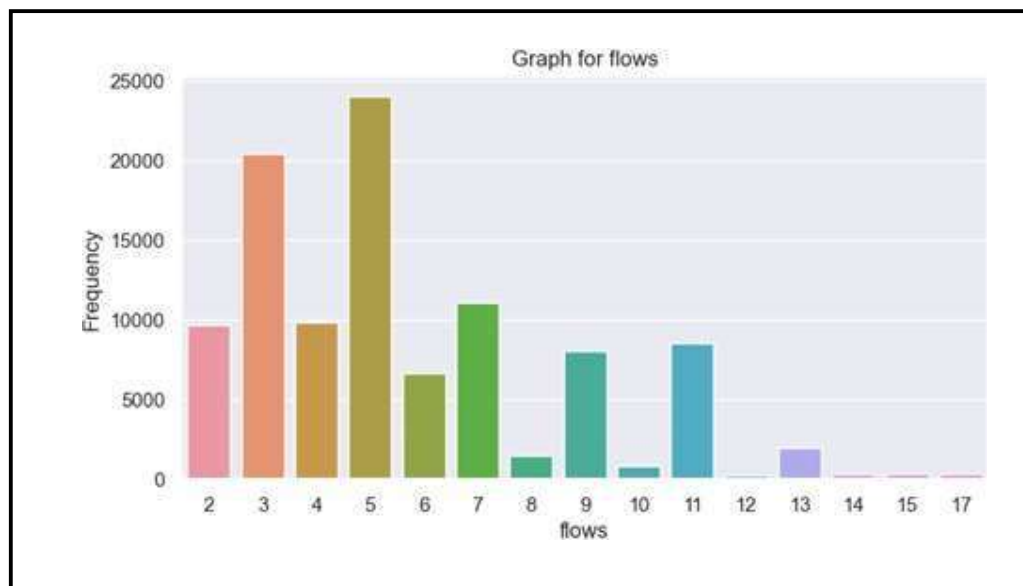


Fig 6.6 Visualize the distribution of numerical discrete features

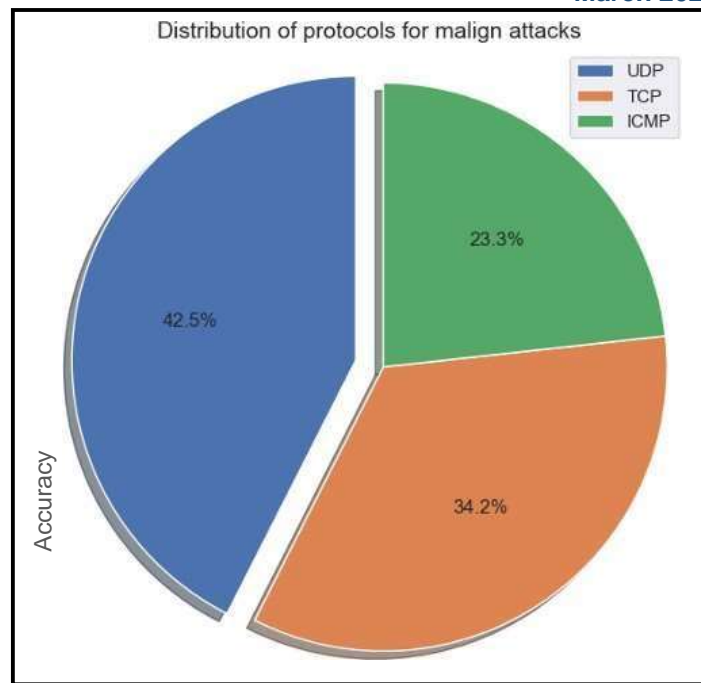


Fig 6.7 Distribution of protocols for malign attacks

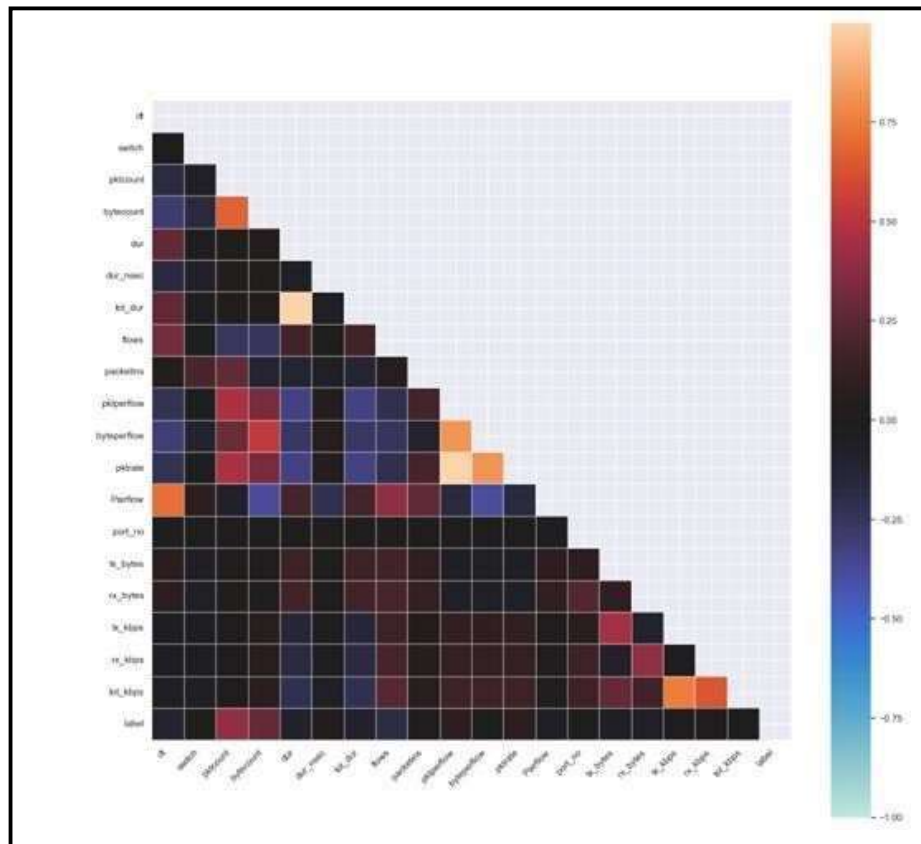


Fig 6.8 Heat map of correlation of features

## 6.7 Results

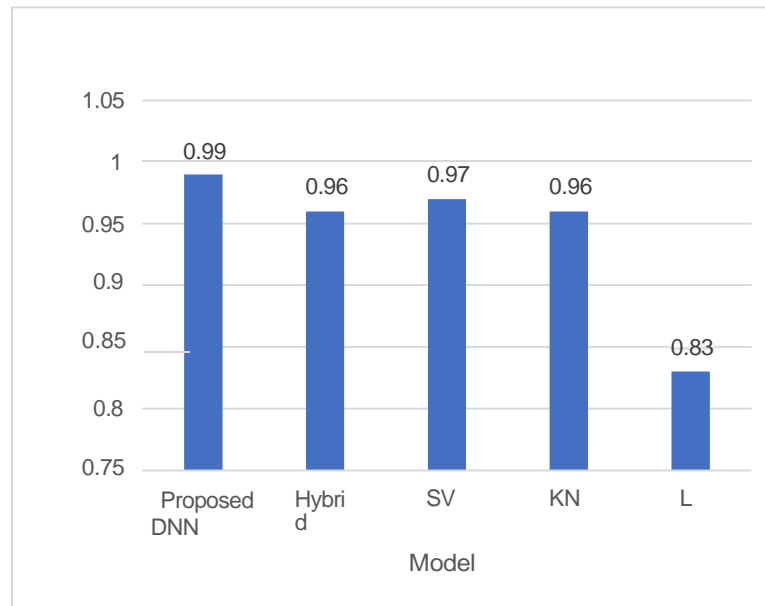


Fig 6.9 Comparison of accuracies with DNN model

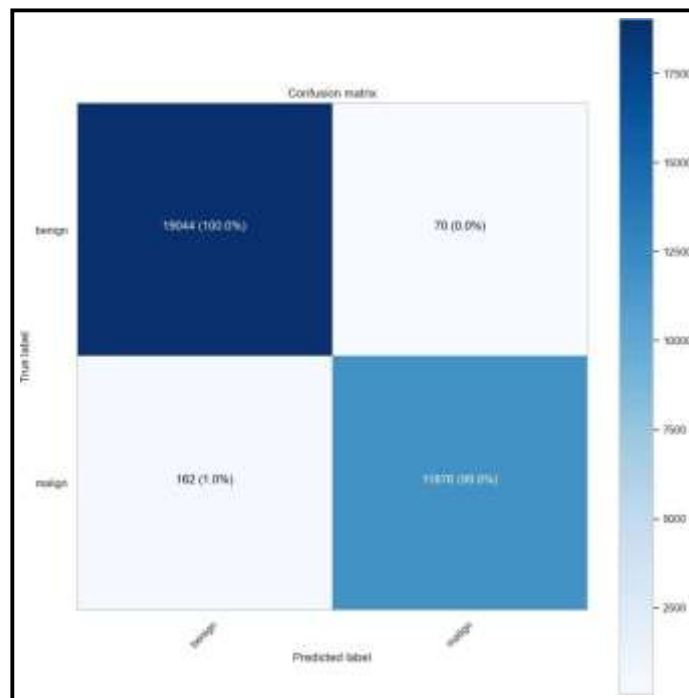
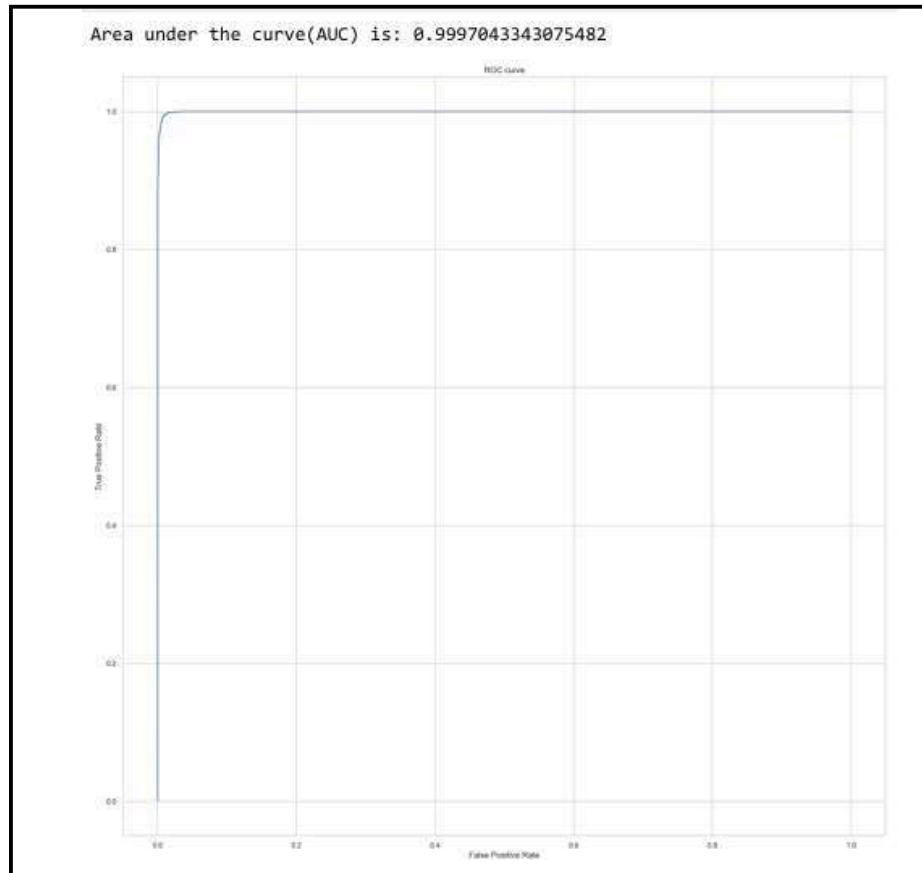


Fig 6.10 Confusion matrix of DNN model

Performance Metric	Score
Accuracy	0.99
Precision	0.99
Recall	0.99
F1 Score	0.99

**Table 6.3 Model Performance****Fig 6.11 ROC-AUC Curve**

## 7. SYSTEM TESTING

### 7.1 Test case Description

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later on during maintenance also.

#### Need of Testing:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system

meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test.

Each test type addresses a specific testing requirement.

### **Testing Objectives:**

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say,

- Testing is a process of executing a program with the intent of finding an error.
- A successful test is one that uncovers an as yet undiscovered error.
- A good test case is one that has a high probability of finding error, if it exists.
- The tests are inadequate to detect possibly present errors.
- The software more or less confirms to the quality and reliable standards.

### **Levels of Testing:**

In order to uncover the errors present in different phases we have the concept of levels of testing.

#### **System Testing:**

The philosophy behind testing is to find errors. Test cases are devised with this in mind. A strategy employed for system testing is code testing.

#### **Code Testing:**

This strategy examines the logic of the program. To follow this method we developed some test data that resulted in executing every instruction in the program and module i.e. every path is tested. Systems are not designed as entire nor are they tested as single systems. To ensure that the coding is perfect two types of testing is performed or for that matter is performed or that matter is performed on all systems.

## **7.2 Types of Testing**

### **7.2.1 Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

### **7.2.2 Link Testing**

Link testing does not test software but rather the integration of each module in system. The primary concern is the compatibility of each module. The Programmer tests where modules are designed with different parameters, length, type etc.

### **7.2.3 Integration Testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of

components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

### **7.2.4 Functional Testing**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted. Invalid Input : identified classes of invalid input must be rejected. Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised. Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

### **7.2.5 System Testing**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### **7.2.6 Acceptance Testing**

Acceptance Test is performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here is focused on external behavior of the system; the internal logic of program is not emphasized. Test cases should be selected so that the largest number of attributes of an equivalence class is exercised at once. The testing phase is an important part of software development. It is the process of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied.

### 7.2.7 White Box Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### 7.2.8 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

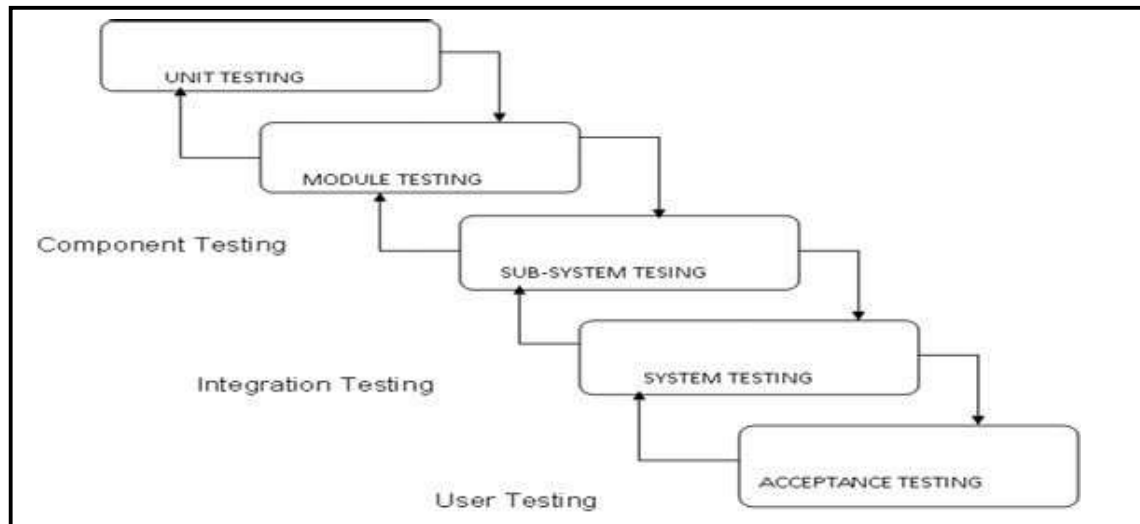


Fig 7.1 Testing Cycle

## 8. CONCLUSION

In this project, we proposed a Deep Learning-based approach for detecting Distributed Denial of Service (DDoS) attacks using a Deep Neural Network (DNN). Our model effectively addresses the limitations of conventional machine learning techniques, such as high computational complexity, difficulty in feature selection, and lower accuracy in hybrid models. By leveraging deep learning, our method automatically extracts high-level features from network traffic data, improving detection capabilities.

Experimental results demonstrate that our DNN-based model outperforms existing approaches, achieving a significantly higher accuracy of 99.38%, compared to the 96% accuracy of the hybrid SVM-KNN-LR model. The proposed system reduces error propagation and computational overhead while enhancing real-time attack detection. Performance metrics, including precision, recall, and F1-score, confirm the model’s robustness and efficiency in identifying malicious traffic with minimal false positives.



For future work, integrating more advanced architectures such as Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN) could further improve detection accuracy. Additionally, incorporating real-time monitoring and adaptive learning mechanisms can enhance the model's responsiveness to evolving cyber threats. With the continuous rise of cyber attacks, our approach provides a scalable and efficient solution to safeguard cloud environments and network infrastructures from DDoS threats.

## 9. FUTURE ENHANCEMENT

In the future improvements of the project, several improvement and extensibility possibilities could be explored to improve the efficiency and robustness of the DDoS detection system. First, the integration of advanced anomaly detection techniques, such as recurrent neural networks (RNNs) or autoencoders, can enable the detection of more subtle and sophisticated DDoS attacks that can evade traditional detection methods. For example, RNNs excel at capturing temporal dependence in sequential data, making them well suited for detecting network traffic patterns over time that are indicative of DDoS attacks. In addition, using ensemble learning techniques such as boosting or bagging can combine the strengths of multiple detection algorithms to create a more flexible and accurate detection system. By combining the outputs of different models, ensemble methods can mitigate the bias of individual models and improve the overall detection performance, which improves the system's ability to detect various DDoS attacks with high accuracy.

In addition, the scalability and adaptability of the system to evolving cyber threats may be the focus of future development. This can include optimizing system architecture and algorithms to effectively manage large network environments and dynamically adapting to new DDoS attack strategies in real time. Implementing a distributed detection framework that can be seamlessly scaled across distributed computing resources can improve system scalability and responsiveness, enabling it to effectively monitor and protect against DDoS attacks in high-traffic and geographically dispersed networks. In addition, incorporating threat intelligence streams and machine learning-based threat detection capabilities can enable the system to proactively identify and mitigate new DDoS attack vectors before they pose a significant threat to network security. By adapting to the ever-evolving and changing threat environment, an advanced DDoS detection system can better protect network infrastructures and ensure uninterrupted service availability for organizations in the face of evolving cyber threats.

## BIBLIOGRAPHY

- [1] S. Dong and M. Sarem, "DDoS attack detection method based on improved KNN with the degree of DDoS attack in software defined networks," IEEE Access, vol. 8, pp. 5039–5048, 2020, doi:10.1109/ACCESS.2019.2963077.
- [2] Lubna Fayez Eliyan and Roberto di Pietro, "DeMi: A Solution to Detect and Mitigate DoS Attacks in SDN", IEEE Access, August 2023, Digital Object Identifier 10.1109/ACCESS.2023.3301994.
- [3] Wu Zhijun, Xu Qing, Wang Jingjie, Yue Meng and Liu Liang, "Low-Rate DDoS Attack Detection Based on Factorization Machine in Software Defined Network," IEE Access, January 2020, Digital Object Identifier 10.1109/ACCESS.2020.2967478.
- [4] Jesus Arturo, Ismael Amezcua Valdovinos, Kim-Kwang Raymond Choo and Dakai Zhu "A Flexible SDN-Based Architecture for Identifying and Mitigating Low-Rate DDoS Attacks Using Machine Learning," in IEEE Access, September 2020, Digital Object Identifier 10.1109/ACCESS.2020.3019330.

- [5] Umar Islam, Abdullah Al-Atawi, H.S.Alwageed, M.Ahsan, Fuad A.Awwad and M.R.Abonazel, “Real-Time Detection Schemes for Memory DoS (M-DoS) Attacks on Cloud Computing Applications,” in IEEE Access, July. 25, 2023, Digital Object Identifier 10.1109/ACCESS.2023.3290910.
- [6] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón, and D. Siracusa, “LUCID: A practical, lightweight deep learning solution for DDoS attack detection,” IEEE Trans. Netw. Service Man age., vol. 17, no. 2, pp. 876–889, Jun. 2020, doi: 10.1109/TNSM. 2020.2971776.
- [7] T. Jili and N. Xiao, “DDoS detection and protection based on cloud computing platform,” J. Phys., Conf. Ser., vol. 1621, no. 1, Aug. 2020, Art. no. 012005, doi: 10.1088/1742- 6596/1621/1/012005.
- [8] A.Amjad, T.Alyas, U.Farooq, and M.A.Tariq, “Detection and mitigation of DDoS attack in cloud computing using machine learning algorithm,” EAI Endorsed Trans. Scalable Inf. Syst., vol. 6, no. 23, pp. 1–8, 2019, doi: 10.4108/eai.29-7-2019.159834.
- [9] Xiaoyong Yuan, Chuanhuang Li, Xiaolin Li “DeepDefense: Identifying DDoS Attack via Deep Learning” IEEE 2017 doi: 978-1-5090-6517-2/17/\$31.00.
- [10] Sandeep K, Reyana A, Ankit Vidyarthi, “SDMTA: Attack Detection and Mitigation Mechanism for DDoS Vulnerabilities in Hybrid Cloud Environment” IEEE Transactions on Industrial Informatics, VOL 18,No.9,September2022.