### DoS/DDoS Attack & Detection

***A report submitted for the completion of***

###### Bachelor of Technology in Computer Science and Engineering (Cyber Security)

by

**Arnab Bag (10331721012)**

**Arnab Saha (10331721013)**

#### Kunal Dhibar (10331721023) Rahul Kumar Mishra (10331721043)

*Under the guidance of*

***Asst. Prof. Mr. Kallol Bera***

*(CSE CS)*



###### DEPARTMENT OF CSE (CYBER SECURITY) HALDIA INSTITUTE OF TECHNOLOGY

**HALDIA, PURBA MEDINIPUR, WEST BENGAL, INDIA**

2025

# DECLARATION

We solemnly declare that the work presented in this project report is an original record of our efforts, undertaken as part of the partial fulfilment requirements for the Bachelor of Technology degree in Computer Science and Engineering with a specialization in Cyber Security. We further affirm that this work has not been submitted to any other university or institute for the award of any degree.

Signature of students

………………………….

Arnab Bag (Roll no:10331721012)

………………………….

Arnab Saha (Roll no: 10331721013)

………………………….

Kunal Dhibar (Roll no: 10331721023)

………………………….

Rahul Kumar Mishra (Roll no: 10331721043)

# ACKNOWLEDGEMENT

We wish to extend our deepest gratitude to **Mr. Kallol Bera**, our project mentor, for his invaluable guidance, unwavering encouragement, and steadfast support throughout this project. His expertise, patience, and insightful feedback have played a pivotal role in refining our skills and deepening our understanding during this internship.

We also wish to express our heartfelt thanks to the Department of Computer Science and Engineering, specializing in Cyber Security, for their continuous support and encouragement, which has been instrumental in shaping the success of this project.

Finally, we would like to thank everyone who, directly or indirectly, supported and encouraged us throughout this journey. Their contributions have been indispensable in the successful completion of this endeavor.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **CONTENT** | **Page No.** |
|  | **CHAPTER 1 INTRODUCTION** | **7-10** |
| 1.1 | Overview of the project | 8 |
| 1.2 | Objectives and goals | 9 |
| 1.3 | Target users | 10 |
|  | **CHAPTER 2 LITERATURE REVIEW** | **11-17** |
|  | **CHAPTER 3   METHODOLOGY** | **18-35** |
| 3.1 | Data collection | 18 |
| 3.2 | Data Pre-processing | 21 |
| 3.3 | Tools and Technologies used | 25 |
|  | **CHAPTER 4 ANALYSIS PROCESS AND VISUALIZATION** | **36-42** |
| 4.1 | ANALYSIS PROCESS | 36 |
| 4.2 | VISUALIZATION | 40 |
|  | **CHAPTER 5 RESULT AND DISCUSSION** | **43-51** |
|  | **CHAPTER 6  CONCLUSION AND FUTURE SCOPE** | **52-64** |
|  | **REFERENCE** | **65** |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **FIGURE** | **CONTENT** | **PAGE.NO** |
| 4.1 | Denial of Service Attack | 37 |
| 4.2 | Work Flow Diagram Denial of Service Attack | 39 |
| 5.1 | Confusion matrix for naive Bayes | 43 |
| 5.2 | Confusion matrix for logistic regression | 44 |
| 5.3 | Confusion matrix for Decision matrix | 44 |
| 5.4 | Confusion matrix for random forest | 45 |
| 5.5 | Confusion matrix for SVM | 46 |
| 5.6 | ROC Curve for naive Bayes | 46 |
| 5.7 | ROC Curve for logistic regression | 47 |
| 5.8 | ROC Curve for Decision Tree | 47 |
| 5.9 | ROC Curve for Random Forest Classifier | 48 |
| 5.10 | ROC Curve for SVM | 48 |
| 5.11 | ROC Curve for BERT | 49 |
| 5.12 | Accuracy table for naive Bayes | 50 |
| 5.13 | Accuracy table for logistic regression | 50 |
| 5.14 | Accuracy table for decision tree | 50 |
| 5.15 | Accuracy table for random forest | 51 |
| 5.16 | Accuracy table for SVM | 51 |
| 5.17 | Accuracy table for BERT | 51 |
| 8.1 | Code Snippets | 67-72 |

# ABSTRACT

In recent years, the frequency and complexity of Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks have increased significantly, posing serious risks to the availability, security, and performance of network services. Traditional security tools like firewalls and static intrusion detection systems often struggle to cope with these evolving, large-scale attacks. To address this challenge, our project introduces an intelligent, scalable DDoS Detection and Mitigation System that uses Machine Learning to proactively identify, analyze, and respond to such threats in real time.

The system continuously monitors network traffic using Wireshark, a widely used packet capturing and intrusion detection tool. The captured data is preprocessed and fed into a combination of supervised and unsupervised machine learning algorithms trained to detect unusual traffic patterns, irregular packet flows, and misuse of network protocols — all typical indicators of DoS and DDoS attacks. Additionally, we integrated the Elastic Stack (Elasticsearch, Logstash, Kibana) to enhance data analysis and visualization. This allows administrators to view network activity in real time, quickly identify potential threats, and take immediate action. The entire framework is deployed on cloud-based platforms like GitHub and Render, ensuring flexibility, scalability, and secure access for different types of organizations.

To enhance the analytical and visualization capabilities of the system, the project integrates the **Elastic Stack (Elasticsearch, Logstash, Kibana)**, enabling real-time indexing, filtering, and dynamic dashboard visualizations. This provides users with a comprehensive view of network activity, enabling rapid identification of potential threats, assessing their severity, and initiating appropriate mitigation measures.

Looking ahead, this system has strong potential for future enhancements. Incorporating deep learning models such as Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks could enable more accurate, context-aware detection of complex attack patterns. Deploying the system on enterprise-grade cloud services like AWS, Microsoft Azure, or Google Cloud would further improve reliability, performance, and resource management. Additional upgrades such as automated incident response, blockchain-secured logging, and AI-driven predictive analytics could significantly boost the system’s resilience, efficiency, and overall effectiveness in handling modern cyber threats.

# Chapter-1

**INTRODUCTION**

In today’s digital environment, safeguarding networks against evolving cyber threats such as **Distributed Denial of Service (DDoS)** and **Denial of Service (DoS)** attacks is crucial. As the frequency and sophistication of these attacks continue to rise, traditional security measures often fail to detect and mitigate malicious traffic effectively. This project, **DDoS Detection and Mitigation Using Machine Learning and Wireshark**, addresses this critical challenge by combining advanced intrusion detection systems (IDS), real-time traffic analysis, and **machine learning techniques** to enhance network security and resilience.

**Wireshark**, a high-performance, open-source network analysis tool, forms the foundation of this project. Renowned for its multi-threading capabilities and robust support for real-time packet analysis, Wireshark monitors network traffic, identifies anomalies, and flags potential DDoS/DoS attack patterns. These capabilities are enhanced by integrating **machine learning-based anomaly detection models** that correlate traffic data with predefined signatures and behavioral patterns.

The primary objectives of this project include:

**1.DDoS/DoS Attack Detection:** Utilizing Wireshark and machine learning algorithms to identify abnormal traffic behaviors such as high request rates, resource exhaustion, and IP spoofing, indicative of DDoS or DoS attacks. [5]

**2.Traffic Analysis:** Parsing and analyzing Wireshark-generated logs combined with ML-driven pattern recognition to gain actionable insights into attack sources, vectors, and severity.

**3.Automation and Efficiency:** Developing automated scripts and ML-powered tools for traffic analysis, visualizing real-time data through dynamic dashboards, and deploying mitigation strategies such as IP blocking and intelligent rate limiting.

This project aims to significantly enhance the detection, analysis, and mitigation of DDoS/DoS attacks in real time by integrating **machine learning models** into the traditional monitoring workflow. The final outcome will demonstrate the practical effectiveness of combining **Wireshark** with **ML-based traffic analytics**, offering organizations a scalable, proactive solution against one of the most disruptive cyber threats in the modern landscape.

## Overview of the Project

###### The convergence of DDoS/DoS detection, traffic analysis, and machine learning (ML) has emerged as a critical paradigm for proactive network security. By leveraging advanced ML techniques alongside traffic monitoring, organizations can detect threats more accurately, identify attack patterns, and safeguard system availability and performance.

###### Key Benefits of Combining DDoS/DoS Detection, Traffic Analysis, and Machine Learning

**Enhanced Security:**

* **Proactive Attack Identification:** By using ML models to analyze traffic patterns for anomalies, organizations can detect early signs of DDoS or DoS attacks, such as traffic floods, protocol misuse, and IP spoofing.
* **Swift Mitigation Response:** ML-powered detection enables timely recognition of attack vectors, allowing for immediate implementation of mitigation strategies, minimizing service disruption and ensuring business continuity.[8]

**Improved Network Reliability:**

* **Traffic Management:** ML algorithms can identify unusual traffic spikes and distinguish between legitimate surges and malicious traffic, allowing for smarter traffic load balancing or blocking.
* **Performance Optimization:** Traffic analysis, enhanced by ML, highlights inefficiencies and patterns that may not be easily visible, enabling better resource allocation and prevention of system overloading.

**Deeper Operational Insights:**

* **Attack Source Identification:** Machine learning can classify and cluster traffic behavior, revealing recurring malicious sources and botnet activity.
* **System Trend Analysis:** ML models trained on historical traffic data help uncover long-term trends and predict future threats, supporting data-driven security decisions and capacity planning.

This project leverages the combination of DDoS/DoS detection, advanced traffic analysis, and machine learning to create a robust defense system, equipping organizations with tools to proactively secure their networks and ensure operational resilience.

**1.2 Objectives and Goals**

* **Data Preprocessing and Feature Engineering:** Use ML techniques to preprocess and transform network traffic data. This includes handling missing data, filtering noise, detecting anomalies, and extracting features like traffic volume, packet size, and connection duration.
* **Attack Pattern Recognition and Signature Extraction:** Apply supervised and unsupervised ML methods to recognize patterns of DoS/DDoS attacks, distinguishing between normal and suspicious behaviour through traffic signatures and abnormal request rates.
* **Model Selection and Evaluation:** Evaluate and compare ML models such as Random Forests, Support Vector Machines, Neural Networks, and Gradient Boosting for detecting DoS/DDoS attacks. Use evaluation metrics like detection rate, false positive rate, precision, recall, and F1-score.[12]
* **Cross-Validation and Hyperparameter Tuning:** Use techniques like k-fold cross-validation and hyperparameter tuning (grid/randomized search) to fine-tune ML models for improved detection accuracy and lower false alarms.
* **Real-Time Detection and Scalability:** Design and deploy scalable ML models capable of processing high-throughput network traffic in real-time, adapting quickly to changing attack strategies.
* **Interpretability of Detection Models:** Use ML interpretability tools like SHAP or LIME to explain how decisions are made, helping analysts understand why certain traffic is flagged as malicious.
* **Mitigation Strategy Recommendations:** Integrate the detection system with ML-driven recommendations, such as adaptive rate-limiting or dynamic IP blocking, to reduce attack impact.
* **Ethical Considerations and Data Privacy:** Ensure ML models are trained on anonymized data and free of bias, maintaining user privacy while avoiding misclassification.
* **Dataset Enrichment and Updating:** Use continual learning and model retraining to include new types of attacks and evolving threats, ensuring long-term effectivene

**1.3 Target Users**

Web security plays a vital role in safeguarding modern digital infrastructure, especially as cyber-attacks become more frequent and sophisticated. Among the most disruptive are Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks, which target network and server availability.

DoS attacks aim to overwhelm a target with excessive requests, while DDoS attacks coordinate large-scale attacks using botnets. These can lead to service outages, financial losses, and reputational harm.

Detecting these attacks early is essential. Traditional rule-based systems often fall short against new or evolving attack methods. **Machine learning provides a smarter approach**—by learning patterns from historical traffic data, ML models can detect anomalies that may indicate an attack.

The motivation behind developing a website vulnerability checker now includes leveraging ML to scan websites for DoS/DDoS vulnerabilities, improving accuracy and speed. This tool can process multiple URLs, detect subtle vulnerabilities, and export results to Excel—making it practical for IT teams.

By integrating ML, organizations enhance not just detection, but also resilience—ensuring their digital infrastructure can adapt to and withstand future cyber threats.

**Chapter – 2  
  
LITERATURE REVIEW**

**Kar, Suvra, D., et al. (2025)** Kar et al. propose a real-time DDoS detection framework utilizing machine learning algorithms to achieve low-latency and high-accuracy classification of malicious traffic. Their model evaluates various supervised learning methods including Decision Trees, Random Forests, and Support Vector Machines, optimized for network packet-level data. The paper emphasizes real-time adaptability and responsiveness as core requirements in DDoS detection. By leveraging traffic feature engineering and low computational complexity algorithms, their system proves effective even under high-throughput network conditions. The authors also acknowledge the necessity of continuous model retraining to cope with evolving attack patterns in modern distributed infrastructures. [1]

**Ismail et al. (2022)** Ismail and colleagues introduce a classification and prediction methodology based on machine learning to detect DDoS attacks. Using comprehensive datasets, the study compares classifiers such as Naive Bayes, KNN, and Random Forests in terms of their detection accuracy and performance under real-time constraints. Their findings show that ensemble methods outperform simpler models, especially in environments with mixed normal and attack traffic. They highlight feature extraction and selection as crucial steps in boosting model robustness and minimizing false positives. The paper contributes to DDoS research by presenting a scalable, predictive framework suitable for deployment in diverse network architectures. [2]

**Akgun, D., Hizal, S., & Cavusoglu, U. (2022)** Akgun et al. explore the integration of deep learning into intrusion detection systems specifically for DDoS attacks. They propose a deep neural network model tailored for high-speed packet classification. Unlike traditional systems, their approach captures intricate traffic patterns and correlations, enabling improved detection of sophisticated DDoS variants. The paper underlines the strength of deep architectures like fully connected DNNs in learning complex data distributions and adapting to unseen attack types. Additionally, the model achieves high accuracy with reduced false detection rates, making it suitable for deployment in cybersecurity-critical infrastructures. [3]

**Anonymous (2023)** A comprehensive survey published in the *Journal of Network and Computer Applications* categorizes various machine learning techniques applied in DoS and DDoS attack detection and mitigation. The study traces the evolution of detection strategies from rule-based systems to advanced supervised and unsupervised learning models. It compares the effectiveness of algorithms such as SVM, Decision Trees, Neural Networks, and Clustering methods based on dataset types, evaluation metrics, and attack scenarios. Furthermore, the paper identifies gaps in explainability, dataset standardization, and real-time applicability, setting the stage for future interdisciplinary research combining ML, cybersecurity, and data science. [4]

**Lee, S. H., et al. (2022)** Lee et al. address the vulnerability of IoT devices to DDoS attacks by implementing a detection and prevention framework tailored to IoT environments. Their solution utilizes lightweight machine learning models capable of running on resource-constrained devices. The authors test their system on real-world IoT traffic datasets, achieving promising results in terms of latency and accuracy. The study emphasizes real-time responsiveness and energy efficiency as key parameters, noting that IoT-based DDoS detection requires trade-offs between computational load and detection precision. [5]

**Liu, X., & Du, Y. (2023)** Liu and Du present a novel method for IoT botnet attack detection by utilizing a **genetic algorithm (GA)** for effective feature selection, integrated with machine learning classifiers. The research identifies the key challenge of high-dimensional data in IoT environments, where redundant or irrelevant features degrade model performance. Their GA-based feature selection significantly enhances the accuracy, precision, and computational efficiency of detection models. The study evaluates various classifiers, including Random Forest, SVM, and Decision Trees, before and after feature optimization. Experimental results show up to a 10% improvement in F1-score with the proposed technique. The authors highlight that intelligent feature selection not only reduces overfitting but also enables real-time detection in resource-constrained IoT systems. This approach demonstrates the potential of bio-inspired algorithms in strengthening cybersecurity defenses in smart environments. [6]

**Marvi, M., Arfeen, A., & Uddin, R. (2021)** Marvi et al. propose a **generalized machine learning framework** for detecting DDoS attacks, adaptable across various network architectures and traffic conditions. The model is designed to function efficiently with different datasets and under dynamic network environments, highlighting the need for flexible and robust detection solutions. The authors compare a suite of algorithms including Logistic Regression, Decision Trees, and Random Forests, and introduce data preprocessing strategies such as normalization, balancing, and noise reduction. Their framework exhibits high detection accuracy across multiple performance metrics (recall, precision, F1-score), proving its effectiveness in both training and unseen testing scenarios. A key contribution of this study is its portability — the proposed model can be adapted to different network topologies and traffic loads without substantial retraining, which is crucial for deployment in heterogeneous real-world systems. [7]

**Seifousadati, A., Ghasemshirazi, S., & Fathian, M. (2021)** Seifousadati et al. focus on **IoT-specific DDoS detection** by designing a machine learning model tailored to the characteristics of low-power, low-latency IoT networks. The paper highlights the growing vulnerability of IoT ecosystems to DDoS attacks, especially due to the proliferation of insecure, internet-connected devices. Their approach employs supervised learning models, trained on both synthetic and real-world traffic datasets, to distinguish normal from malicious IoT behavior. The authors prioritize lightweight computation, proposing the use of simplified feature sets and efficient classifiers such as Decision Trees and Gradient Boosting. Despite the computational constraints of IoT devices, their model maintains high accuracy and low false positive rates. The study also discusses challenges such as real-time adaptability, dynamic attack evolution, and device heterogeneity, calling for continual research into adaptive and scalable solutions for IoT security. [8]

**Ullah, S., Mahmood, Z., Ali, N., Ahmad, T., & Buriro, A. (2023)** Ullah et al. present a **dynamic attribute selection technique** for DDoS attack classification in IoT networks using machine learning. Their approach centers on adaptive feature selection, where the set of input attributes changes dynamically based on the network context, traffic characteristics, or attack signatures. This strategy is particularly suited for IoT environments, where network states are continuously shifting and static models may fail. The authors employ classifiers such as SVM and Random Forest in conjunction with dynamic feature selection to enhance adaptability and detection performance. Their results show improved detection accuracy, particularly in identifying low-rate or stealthy DDoS attacks that often evade static threshold-based systems. The paper’s key contribution lies in demonstrating how **context-aware feature selection** can significantly improve resilience and responsiveness in IoT-based intrusion detection systems. [9]

**Zhao, J., Xu, M., Chen, Y., & Xu, G. (2023)** Zhao et al. introduce a novel method for generating Deep Neural Network (DNN) architectures tailored for DDoS detection using **Genetic Algorithms (GAs)**. Their approach automates the design of optimal DNN topologies by encoding architectural parameters such as layer count, node distribution, and activation functions into genetic chromosomes. Through evolutionary operations — selection, crossover, and mutation — the system evolves towards more effective detection models. The method is evaluated against traditional hand-crafted architectures on benchmark DDoS datasets, showing superior accuracy and generalization performance. By automating architecture optimization, this study significantly reduces the need for manual tuning, promoting scalable and efficient solutions for complex intrusion detection tasks in dynamic environments. [10]

**Detecting DoS attacks using machine learning algorithms (2022)** This anonymous study published in the *Journal of Big Data* offers an extensive evaluation of various **machine learning algorithms** for detecting Denial of Service (DoS) attacks. It emphasizes the role of data preprocessing, feature engineering, and class balancing in enhancing model effectiveness. Algorithms such as Decision Trees, Naive Bayes, and k-NN are compared across metrics like accuracy, precision, and detection time. A key insight from the paper is the trade-off between lightweight models (suitable for real-time deployment) and more complex models (with higher accuracy but increased computational cost). The research contributes a practical benchmark for choosing appropriate ML models depending on resource constraints and deployment scenarios. [11]

**Virupakshar, et al. (2020)** Virupakshar and colleagues present a **DNN-based detection model** specifically designed for DoS/DDoS attacks in cloud environments. Their work, part of a SciTePress evaluation paper, utilizes a deep neural network to learn complex traffic patterns within cloud infrastructure where traditional threshold-based approaches often fail. The model is trained on multiple datasets including NSL-KDD and CICIDS, achieving high precision and recall in detecting volumetric and slow-rate attacks. The paper also discusses the impact of network virtualization and multitenancy on detection accuracy. By integrating DNNs with cloud-native monitoring tools, the authors propose a robust solution that scales with cloud traffic volumes while minimizing false alarms. [12]

**Farhat, S., Abdelkader, M., Meddeb Makhlouf, A., & Zarai, F. (2023)** Farhat et al. conduct a rigorous evaluation of machine learning techniques for detecting DoS and DDoS attacks using the **CIC IDS2017 dataset**, one of the most comprehensive real-world network traffic datasets available. They benchmark a variety of classifiers, including Random Forest, Gradient Boosting, and Support Vector Machines, analyzing their performance across multiple attack categories such as UDP flood, HTTP DoS, and Slowloris. The study emphasizes feature importance ranking and dimensionality reduction as key steps in achieving high accuracy while maintaining low computational cost. Their results highlight the effectiveness of ensemble models and hybrid approaches, particularly in distinguishing between similar traffic patterns typical of legitimate spikes and DDoS bursts. [13]

**Perez Diaz, V., Choo, K., & Zhu, Y. (2020)** Perez Diaz and colleagues propose a **Software Defined Networking (SDN)-based architecture** to identify and mitigate low-rate DDoS attacks using ML. Recognizing that low-rate DDoS attacks often evade traditional threshold-based detection, their system utilizes the centralized control capabilities of SDN to monitor traffic flows and dynamically respond to anomalies. Machine learning models are employed to classify traffic based on entropy and flow features in near real-time. The framework supports active mitigation by rerouting or throttling malicious flows. This paper makes a significant contribution to adaptive and intelligent network defense by combining the flexibility of SDN with the predictive power of ML for stealthy attack detection. [14]

**Phan, Park (2019)** Phan and Park present a **distributed defense mechanism** for DDoS attacks in SDN-based cloud environments. The proposed system includes a decentralized detection layer and a centralized mitigation layer that collaborate via SDN controllers. Their solution uses decision-tree-based classifiers to detect anomalies in flow-based traffic and responds by reconfiguring routing paths to isolate malicious nodes. The paper emphasizes scalability, proposing that the distributed nature of their framework makes it robust against both volumetric and targeted attacks. Testing on simulated cloud environments shows substantial improvements in detection latency and mitigation response compared to traditional monolithic systems. [15]

**Dong, S., & Sarem (2020)** Dong and Sarem propose an **enhanced k-Nearest Neighbors (KNN)** approach for DDoS attack detection within **Software Defined Networking (SDN)** environments. The model modifies the classical KNN algorithm by incorporating adaptive distance weighting and optimized neighbor selection, which significantly improves classification precision and reduces false positive rates. Tested on benchmark datasets, the improved KNN model demonstrates higher accuracy in distinguishing between benign and malicious traffic compared to standard classifiers. The authors emphasize that integrating such a lightweight yet effective model into SDN controllers can provide real-time threat intelligence and proactive mitigation. This work contributes to the field by refining traditional algorithms for deployment in dynamic, programmable network infrastructures. [16]

**Sambangi, G., & Gondi (2020)** In this study, Sambangi and Gondi adopt **Multiple Linear Regression (MLR)** for detecting DDoS attacks — a relatively uncommon choice compared to tree-based or neural methods. Their approach models the correlation between network traffic features (such as packet size, duration, and flow count) and attack probability. Surprisingly, despite its simplicity, the MLR model achieves competitive performance on datasets like NSL-KDD and CICIDS. The authors argue that linear regression's interpretability and low computational overhead make it suitable for edge computing scenarios and IoT deployments, where resource constraints are significant. Their findings challenge the notion that only complex models are fit for cyber threat detection. [17]

**Najar & Naik (2021)** Najar and Naik conduct a comparative evaluation of **multiple ML algorithms** for DDoS detection using the **UNSW-NB15** dataset, which includes a rich mix of modern attack vectors. Their study explores algorithms including Random Forest, SVM, Naive Bayes, and XGBoost, evaluating them on metrics such as detection rate, F1-score, and training time. Notably, ensemble methods outperform individual classifiers, especially in handling imbalanced datasets. The study highlights the role of hyperparameter tuning and cross-validation in improving robustness. Their contribution is significant in guiding researchers and practitioners toward suitable algorithmic choices when using UNSW-NB15 for intrusion detection tasks. [18]

**Slyamkhanov, A. et al. (2023)** Presented at WINCOM 2023, this study by Slyamkhanov et al. explores the **application of ML techniques** for DDoS detection in large-scale network environments. The authors design a modular detection framework incorporating feature extraction, normalization, and classifier stacking to improve generalization. The paper benchmarks classic models like Decision Trees and Gradient Boosting alongside deep learning models such as LSTM. The hybrid stacked ensemble showed improved performance in both accuracy and detection latency. Emphasis is placed on data diversity and training resilience across different attack scenarios, which is essential for real-world deployment in heterogeneous networks. [19]

**Borah, R., Sarmah, S., Choudhury, N., Mahanta, H., & Chodhury, A. (2023)** This research provides a practical implementation of **ML-based DDoS detection** using real-time traffic scenarios. Borah et al. examine supervised learning algorithms on a custom-collected dataset mimicking real-world DDoS conditions. The authors use algorithms such as Logistic Regression, SVM, and Random Forest, reporting results on confusion matrices and ROC curves. The model emphasizes early-stage attack identification to allow faster mitigation responses. The work is valuable for its application-centric approach and its effort to close the gap between academic models and operational network defense systems. [20]

**Bandi, A., Sherpa, L., & Allu, S. M. (2022)** In this book chapter, Bandi and colleagues provide a comprehensive overview of **modern ML algorithms** used in cybersecurity, with a focused section on DDoS attack detection. They categorize algorithms based on their learning paradigm — supervised, unsupervised, and semi-supervised — and compare their efficiency on various network datasets like NSL-KDD, CICIDS2017, and BoT-IoT. The chapter also discusses real-world implementation challenges, including feature drift, adversarial attacks, and scalability issues. Their integrative approach offers a useful synthesis for both academics and practitioners seeking to understand the trade-offs involved in deploying ML models in security operations. [21]

**Amrish, R., Bavapriyan, K., Gopinaath, V., Jawahar, A., & Kumar, C. V. (2022)** Amrish et al. investigate the use of **supervised machine learning algorithms** for effective detection of DDoS attacks in smart network environments. Their study evaluates a range of classifiers including Decision Trees, Naive Bayes, and KNN on datasets derived from simulated IoT environments. The results indicate that while tree-based methods offer higher accuracy, KNN is more lightweight and suitable for constrained devices. The authors emphasize the importance of **feature selection** and data preprocessing in ensuring model generalizability. This paper contributes significantly to the practical application of ML in next-gen networks that combine IoT, mobile, and cloud technologies. [22]

**Cil, A. E., Yildiz, K., & Buldu, A. (2021)** This study presents a **Feed-Forward Deep Neural Network (F-DNN)** model for DDoS detection. Cil et al. construct a deep architecture with multiple hidden layers to learn complex patterns in network traffic data. Using the CICIDS2017 dataset, their F-DNN model achieved an **accuracy of over 99%**, outperforming traditional machine learning models. The study also explores hyperparameter tuning strategies such as dropout rates, optimizer selection (Adam vs SGD), and activation functions to fine-tune model performance. Their findings demonstrate that DNN-based approaches are well-suited for handling high-dimensional attack data, though at the cost of increased computational resources. [23]

**Chavan, N., Kukreja, M., Jagwani, G., Nishad, N., & Deb, N. D. (2022)** Presented at ICACCS 2022, this paper discusses a dual-focused approach — **detecting DDoS attacks and mitigating botnet propagation** — using ML. The authors design a pipeline that includes feature extraction, attack classification using Random Forest, and post-detection quarantine actions. Their model is integrated into a simulation of enterprise-level network infrastructure. Performance results show high detection accuracy and low false positive rates. Moreover, the system demonstrates robustness against **zero-day attack variants**, emphasizing the utility of ML not only in detection but in active threat containment. [24]

**Nazarudeen, F., & Sundar, S. (2022)** Nazarudeen and Sundar introduce a comparative study on multiple **machine learning models** — including XGBoost, AdaBoost, and Logistic Regression — for detecting DDoS attacks in cloud-enabled environments. The models were evaluated using both CICIDS and custom-generated traffic, emphasizing adaptability. XGBoost outperformed others in both accuracy and processing time. The paper discusses preprocessing strategies like Z-score normalization and SMOTE for addressing class imbalance. Their results highlight the role of ensemble learning in building resilient, low-latency detection frameworks capable of real-time monitoring and response. [25]

**Kavitha, M., et al. (2022)** Kavitha and team present a study focused on **DDoS detection within Software Defined Networks (SDN)** using ML classifiers. The research, shared at ICACRS 2022, integrates flow-based features from SDN controllers and applies algorithms such as Random Forest and Gradient Boosting to detect anomalies. The novelty lies in the **use of flow rule dynamics and entropy calculations** to predict and preempt large-scale DDoS attacks. Results from Mininet-based simulations show a reduction in both packet drop rates and attack latency when the proposed ML module is integrated with the SDN control layer. This work demonstrates the synergy of ML and programmable networks in modern cybersecurity. [26]

**Amaad, H., & Mughal, H. (2023)** In their study presented at ICACS 2023, Amaad and Mughal experiment with **ensemble machine learning techniques** for classifying DDoS attacks. They combine multiple base learners such as Decision Trees, Random Forests, and Gradient Boosting in various ensemble configurations — including bagging and boosting. The primary objective was to improve the detection robustness and reduce overfitting on imbalanced datasets. Their results reveal that ensemble techniques significantly enhance accuracy and recall, especially in the presence of sophisticated or hybrid DDoS attacks. The study also shows improved resilience against noise in data, demonstrating ensemble learning’s advantage over standalone classifiers. [27]

**Fardusy, T., Afrin, S., Sraboni, I. J., & Dey, U. K. (2023)** Fardusy et al. introduce a novel autoencoder-based semi-supervised learning method for detecting DDoS attacks, discussed in the NCIM 2023 proceedings. The model utilizes unsupervised autoencoders to learn latent feature representations from normal traffic and then classifies anomalous traffic based on reconstruction error thresholds. This approach is especially beneficial for real-world settings where labeled attack data is limited. Experimental evaluations conducted using NSL-KDD and CICIDS datasets showed high precision and recall, indicating the model's effectiveness in identifying unseen DDoS traffic. The authors highlight that such models offer scalable and adaptive security mechanisms in dynamic environments. [28]

**Liu, Z., et al. (2023)** This paper, published in *Sensors*, presents a **feature engineering-driven ML approach** to DDoS detection in Software Defined Networks (SDNs). Liu et al. propose a multi-stage detection pipeline that incorporates statistical flow features (e.g., entropy, packet rate, average size) and dimensionality reduction techniques like PCA. They use supervised classifiers including Random Forest and XGBoost to perform attack classification. The SDN controller plays a critical role in supplying rich metadata for feature extraction. Their experiments validate that incorporating **domain-specific features** enhances model precision and F1-score. This highlights the importance of custom feature engineering for context-aware attack detection in programmable networks. [29]

**Ramzan, M., Shoaib, M., Altaf, A., & Ashraf, I. (2023)** Ramzan et al. propose a **deep learning-based approach** for DDoS attack detection, also published in *Sensors (Basel)*. They develop a CNN-LSTM hybrid model to extract both spatial and temporal characteristics from network traffic data. CNNs are applied to capture hierarchical spatial patterns in packet flows, while LSTMs capture time-sequence dependencies of attacks. Using the CICIDS2017 dataset, the model demonstrates superior performance in terms of **detection rate, accuracy, and false positive rate** compared to traditional ML techniques. The research underscores the power of deep learning for real-time threat analysis, particularly in high-speed and large-scale network environments. [30]

Chapter**3:**

## METHODOLOGY

In today’s digital world, DoS (Denial of Service) and DDoS (Distributed Denial of Service) attacks have become serious threats to network security. These attacks flood servers and systems with fake traffic, making them slow or completely unavailable to real users. With the growth of cloud computing, IoT devices, and online services, such attacks have become more frequent and more dangerous. To tackle this growing problem, we need smart and automated systems that can quickly detect and stop these attacks before they cause major damage. To build such systems, we rely on machine learning (ML)—a technology that allows computers to learn from data and make decisions. But creating an effective ML-based detection system requires a step-by-step process. This includes collecting data, cleaning it, choosing useful features, selecting the right model, training the system, and testing how well it works. Each of these steps plays an important role in making sure the system is accurate and reliable. In the past, detection systems mainly used manual rules or signatures—specific patterns that matched known attacks. However, these methods struggled to detect new or unknown attacks. Nowadays, with the help of machine learning, we can build systems that learn from real traffic patterns, spot unusual behavior, and detect both known and unknown attacks. The first step is to gather good-quality data from real or simulated network environments. Popular datasets like CICIDS2017, UNSW-NB15, or NSL-KDD are often used by researchers. Then, the data is cleaned and prepared. This involves removing errors, converting data into usable formats, and focusing on features like packet size, connection time, or number of requests, which help in recognizing attack patterns.

Next, we train ML models using these features. Some commonly used models include Decision Trees, Random Forests, Support Vector Machines (SVM), and Naive Bayes. More recent approaches use deep learning models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), which are better at understanding complex patterns in data. In some cases, researchers even combine multiple models to create more accurate systems. Some advanced systems also use data from Software Defined Networks (SDNs) to get a clearer picture of what’s happening in real time. These systems can adjust themselves on the fly and respond to attacks more effectively.

Once the model is trained, it’s important to test how well it performs. We use measures like accuracy, precision, recall, and F1-score to see how good the model is at detecting attacks without making too many mistakes. A good system should catch most attacks while keeping the number of false alarms low. In recent years, there has also been growing interest in making these systems lighter, so they can run on small devices, and more explainable, so security teams can understand why a certain alert was triggered.

### 3.1 Data Collection

The success of any machine learning model for **DoS/DDoS attack detection** largely depends on the **quality, size, and structure** of the dataset used. In supervised learning, the dataset plays a crucial role in helping the model **learn patterns** that distinguish between normal and malicious network activity. For DoS/DDoS detection, important features such as **traffic flow, packet rate, connection duration, and protocol type** are essential in enabling the model to accurately identify attack behaviors. In this project, we collected data from **publicly available datasets**, widely recognized in the cybersecurity research community. Well-known datasets like **CICIDS2017**, **UNSW-NB15**, and **NSL-KDD** were selected due to their open access, realistic simulation environments, and detailed labeling of network activities. These datasets provide a comprehensive collection of both **benign (normal)** and **malicious (attack)** traffic, which makes them ideal for training and evaluating machine learning models in a binary classification task—**attack vs. no attack**. The datasets used in this study contain **millions of network flow records**, representing real-world traffic captured from enterprise or simulated environments. The attack traffic includes different types of DoS and DDoS attacks, such as **UDP floods, SYN floods, ICMP attacks, Slowloris, and HTTP floods**, among others. Each record typically includes features such as **source IP, destination IP, protocol, packet count, bytes transferred, and connection time**, along with a label that marks it as either an attack or normal activity. Benign traffic generally originates from routine user activities such as browsing websites, sending emails, or using cloud services. In contrast, attack traffic is intentionally crafted to overwhelm a target system, either by flooding it with packets or by exploiting protocol-level weaknesses. The ability to clearly differentiate these two classes of traffic is critical for training robust and reliable detection models. By using these **well-labeled and diverse datasets**, our system is designed to learn meaningful patterns that generalize well to **new, unseen attack variations**. The chosen datasets not only support effective model training but also ensure the validity and reproducibility of the results when comparing different machine learning approaches.

### 3.1.1 Description of Source Files

The primary dataset used for this project consists of **two structured CSV files** commonly used in DoS/DDoS attack detection research. These files contain **network traffic data** collected under both normal and attack conditions, and are sourced from widely accepted benchmark datasets such as **CICIDS2017** or **UNSW-NB15**, available via public repositories like **Kaggle** or **Canadian Institute for Cybersecurity** portals.

#### Normal.csv

* **Content**: This file contains samples of **legitimate network traffic** that represent normal user behavior and typical service usage patterns.
* **Label**: Each row is assigned a label of 0, indicating that it reflects **benign** (non-malicious) activity.
* **Example**: A record might include traffic from a user browsing a website, checking emails, or performing DNS lookups.
* **Column Information**:
  + Source IP, Destination IP: Address information for origin and target.
  + Protocol: Indicates whether the packet is TCP, UDP, ICMP, etc.
  + Packet Length, Duration, Bytes Transferred: Metrics of flow volume.
  + Label: 0 = Normal traffic.

#### Attack.csv

* **Content**: This file contains samples of **malicious traffic** generated during simulated or real-world **DoS and DDoS attacks**.
* **Label**: All records are labeled with a 1, indicating **attack** activity.
* **Example**: A record might show repeated SYN packets sent in short bursts with no ACKs, which is typical of a SYN flood attack.
* **Column Information**:
  + Same structure as Normal.csv.
  + Label: 1 = Attack traffic.

These datasets form the **foundation for a binary classification task**, where the model learns to distinguish between benign and malicious traffic based on the extracted features.

### 3.1.2 Merging and Unifying the Datasets

To prepare a clean and cohesive dataset suitable for supervised machine learning, the **Normal.csv** and **Attack.csv** files were **merged into a single unified dataset**, referred to as data throughout the project. This unified dataset maintains a **binary target column** labeled as 0 (for normal) or 1 (for attack), simplifying model training.

**Pre-processing steps included:**

1. **Column Standardization**:
   * Ensured both CSV files had the same **column headers and formats**, such as matching field names (Protocol, Flow Duration, Packet Count, etc.).
   * Verified data types and handled missing values if present.
2. **Label Encoding**: Standardized the Label column to use binary values: 0 for normal traffic and 1 for DoS/DDoS traffic.
3. **Index Reset**: After merging, the dataset’s index was reset to create a clean, sequential numbering of records.
4. **Data Shuffling**: The combined dataset was **randomly shuffled** to prevent any training bias that might occur if the model sees large blocks of one class in sequence. Shuffling helps improve generalization and avoids local minima in training.

### 3.1.3 Exploratory Data Analysis (EDA)

Initial EDA was conducted to understand the dataset’s structure, distribution, and feature characteristics:

#### Class Distribution:

* The dataset was found to be **roughly balanced**, with a similar number of attack and normal traffic samples.
* This balance is important for avoiding classifier bias toward either class.

#### Feature Distribution:

* **Packet size, duration, and flow rate** varied significantly between normal and attack traffic.
* Attack records typically exhibited **higher frequency**, **shorter intervals**, and **spiky traffic bursts**, consistent with DoS patterns.

#### Protocol Usage:

* Most **normal traffic** used a mix of protocols (TCP, UDP, ICMP).
* Attack traffic often focused on **a single protocol type**, such as large volumes of UDP or TCP SYN packets, reflecting specific attack strategies.

#### Source/Destination Patterns:

* In normal traffic, source-destination IP pairs were diverse.
* In attack traffic, there was often a **one-to-many or many-to-one relationship**, as seen in DDoS attacks targeting a single server.

#### Flow Duration and Inter-arrival Times:

* Attack flows had **shorter durations** and **lower inter-arrival times**, indicating traffic bursts designed to overwhelm targets.
* Normal flows had **longer, more evenly spaced durations** due to steady user interactions.

This EDA helped shape feature selection and model design, guiding decisions on **which metrics best indicate anomalies**, and informing **which algorithms** would perform well for traffic classification.

### 3.2 Data Pre-processing

Data pre-processing is a **critical stage** in any machine learning or deep learning pipeline, particularly in cybersecurity applications such as **DoS and DDoS attack detection**. This phase acts as a bridge between raw, unstructured network traffic data and meaningful, accurate predictions made by the model. Since network traffic can be noisy, high-volume, and highly variable, effective pre-processing ensures that only **relevant, high-quality features** are used to train the model.

In today’s digital landscape, networked systems constantly exchange vast volumes of data. However, this same connectivity allows malicious actors to launch disruptive denial-of-service attacks by overwhelming services with fake requests or malicious packets. To build effective detection systems, it is essential to **transform raw traffic data into a structured and analyzable form** that machine learning algorithms can interpret.

For this project, we aim to create a robust system capable of distinguishing between **normal network activity** and **DoS/DDoS attack traffic**. The accuracy and generalizability of the model are heavily influenced by how well the raw data is cleaned, normalized, and formatted before model training begins. Pre-processing in this context is **not just basic cleaning**—it involves carefully designed transformations that help the model detect subtle patterns in flow behavior, protocol usage, and traffic anomalies.

Our approach employs a combination of **traditional pre-processing techniques** (used with classical machine learning models) and **optimized strategies for deep learning algorithms** like LSTM or CNN-based detectors. These two types of models require different preparation workflows:

* **Classical models** (e.g., Random Forest, SVM) require feature normalization, encoding, and statistical summarization of packet-level or flow-level attributes.
* **Deep learning models** benefit from preserving **temporal sequences, time dependencies, and packet distributions**, often using time-series windows or embedding layers for pattern recognition.

### 3.2.1 Understanding the Dataset

The foundation of any successful model is its data. For this project, we sourced datasets from Kaggle and other trusted cybersecurity repositories. Two primary datasets form the backbone of our research:

* **DoS.csv** – A dataset containing features and records of Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attack traffic.
* **Benign.csv** – A dataset comprising normal network traffic data without attacks.

Each dataset contains multiple features such as source IP, destination IP, packet size, duration, protocol, and various statistical indicators. After loading the data, both files were merged into a single DataFrame (df) to facilitate uniform processing. We introduced a new column called category to serve as the target variable:

* **Label 1** represents DoS/DDoS attacks.
* **Label 0** represents benign (normal) traffic.

This binary classification setup enables supervised learning, allowing the model to distinguish between attack and non-attack traffic based on learned patterns.

### 3.2.2 Initial Data Exploration and Analysis

Before preprocessing, it’s essential to understand the dataset. This process, known as Exploratory Data Analysis (EDA), reveals data distribution, structure, anomalies, and feature importance.

#### 1. Class Balance Check

One of the first steps involved visualizing the distribution of attack vs. normal traffic instances. We used a count plot to see whether the dataset was balanced. An imbalanced dataset (e.g., 90% benign, 10% attack) can mislead the model into favoring the majority class, which in a security context can be dangerous.

#### 2. Protocol and Traffic Type Analysis

We explored features like Protocol and Service to identify patterns. Bar plots and correlation matrices helped reveal whether certain protocols (e.g., TCP, UDP, ICMP) or services (e.g., DNS, HTTP, FTP) were more commonly targeted by attacks.

#### Handling Missing and Corrupted Data

Network datasets often include missing or anomalous entries due to capture errors or incomplete logging. Using df.isna().sum(), we checked for null values in each column. We also looked for zero-duration flows and constant features that do not contribute meaningful insights.

#### Feature Engineering: Combining and Transforming Features

Rather than discarding incomplete rows, we engineered new features such as:

* **Flow Duration**
* **Bytes per Second**
* **Packets per Second**
* **Flag Counts (e.g., SYN, ACK)**

These derived features provided enriched context, enhancing detection capability. Attack traffic often has unique patterns, such as large numbers of packets in a short time or consistent flag sequences.

### 3.2.3 Feature Cleaning and Normalization

Unlike text data, network traffic data involves numerical and categorical features that must be normalized and encoded properly.

#### Key Preprocessing Steps

1. **Lowercasing Categorical Columns:** All protocol and service names were converted to lowercase for consistency.
2. **One-Hot Encoding**: Categorical fields like protocol type and service were converted into binary vectors using one-hot encoding to be compatible with machine learning models.
3. **Removing Special Symbols**: Any non-standard characters or irrelevant metadata (e.g., hex values or padding artifacts) were removed.
4. **Handling Outliers**: Outliers were identified using Z-score and IQR techniques, especially in packet size and duration columns. These were either clipped or replaced.
5. **Feature Scaling**: StandardScaler and MinMaxScaler were applied to scale the features into a uniform range. This step is essential for models like SVM and Logistic Regression.

### 3.2.4 Feature Selection and Optimization

Feature selection helps reduce model complexity and overfitting. Not all traffic metrics are useful for detection.

* **Recursive Feature Elimination (RFE)** and **Random Forest Importance Scores** were used to retain only the most informative features.
* Features such as flow\_duration, average\_packet\_size, and flag\_counts showed strong correlation with attack behavior.
* We ensured domain-specific features like SYN\_Flag\_Count, Total\_Forward\_Packets, and Flow\_Bytes\_per\_Second were retained.

### 3.2.5 TF-IDF Analogy for Network Features

Although TF-IDF is typically used in text, we adapted the concept for time-series numerical features using **term-frequency-like statistics** on protocol sequences and port numbers.

This technique helped us detect anomalies where certain ports or IPs were over-represented in attack traffic. Models were trained using structured numerical vectors formed from the packet data.

We applied these methods to train traditional ML models like:

* **Logistic Regression**
* **Naïve Bayes**
* **Support Vector Machines (SVM)**
* **Decision Tree**
* **Random Forest**

#### Dataset Splitting

To ensure model generalization:

* **80% of data** was used for training.
* **20% of data** was reserved for testing.

Stratified sampling was applied to maintain a balanced distribution of attack and benign records in both sets.

### 3.2.6 BERT-Inspired Pre-processing for Cyber Features (Contextual Deep Models)

While BERT is designed for language, similar techniques are used in cybersecurity for contextual deep learning models like:

* **Transformer-based Intrusion Detection Systems (IDS)**
* **Temporal Convolutional Networks (TCNs)**
* **Recurrent Neural Networks (RNNs) with Attention**

In our case, we used a BERT-like processing pipeline for contextual features:

#### Step-by-Step Pre-processing:

1. **Sequence Encoding** Sequences of protocol types, port numbers, or flags were treated as token-like structures and encoded as sequences.
2. **Adding Positional Encoding** Since order matters in sequential traffic flows (e.g., scan patterns), positional encodings were added to sequence vectors.
3. **Padding and Truncation** All input sequences were padded or truncated to a fixed length (e.g., 128 time steps) to match model requirements.
4. **Attention Masking** Attention masks were generated to identify valid time steps (1) vs. padded ones (0), enabling the model to focus on real activity.
5. **Segment IDs (Optional)** When comparing multiple sessions (e.g., before and after attack), we used segment IDs to differentiate sequences.
6. **Feature Aggregation with Transformer Layers** These contextual inputs were passed into transformer layers, and the output corresponding to the first token (akin to BERT’s [CLS]) was used for classification.

### ****3.3 Tools and Technologies Used****

To begin with, traffic analysis and anomaly detection are the core components of this project, and for that, we utilized two widely adopted data science and machine learning libraries: **pandas** and **NumPy**. Pandas provides powerful data structures and functions for data manipulation, cleaning, and analysis, allowing us to process large-scale network traffic logs efficiently. Meanwhile, NumPy enables fast mathematical operations and numerical computing, which are essential for transforming raw packet data into machine-readable formats. Together, these libraries form the foundation of our data preprocessing pipeline, helping to organize, clean, normalize, and structure network data before any feature engineering or modeling.

For classical machine learning implementation, we relied on the **scikit-learn** library. Scikit-learn offers a consistent and user-friendly interface for a wide range of algorithms and provides essential utilities for data transformation, feature selection, model evaluation, and hyperparameter tuning. In this project, we used scikit-learn to train and evaluate several machine learning classifiers, including **Naïve Bayes**, **Logistic Regression**, **Decision Tree**, **Random Forest**, and **Support Vector Machine (SVM)**.

Each classifier brings unique advantages to cybersecurity:

* **Naïve Bayes**, though based on the assumption of feature independence, is fast and efficient for initial classification.
* **Logistic Regression** is simple yet interpretable and performs well in binary classification tasks like attack vs. benign traffic.
* **Decision Trees** offer clear decision paths that are easy to understand and interpret.
* **Random Forest**, being an ensemble method, improves performance and generalization by aggregating multiple decision trees.
* **SVM** is robust in high-dimensional spaces and particularly effective in distinguishing between subtle anomalies and legitimate traffic patterns.

For deep learning and advanced sequential traffic pattern detection, we leveraged the **PyTorch** deep learning framework. PyTorch is known for its dynamic computational graph and intuitive design, which supports experimentation and development of complex models. Using PyTorch, we constructed and trained deep neural networks tailored to time-series traffic data. These included models inspired by architectures such as **Recurrent Neural Networks (RNNs)** and **Transformers**, which are adept at modeling sequences in packet-level or flow-level features.

To incorporate attention mechanisms and contextual learning similar to **BERT** in NLP, we used the **Transformers** library by Hugging Face. Although originally designed for language tasks, this library has been adapted to cybersecurity for advanced models like **CyberBERT** and **NetTransformer**, enabling learning from protocol sequences, port patterns, and session-level behaviors. These transformer-based models were fine-tuned on network flow sequences, allowing the system to learn complex attack patterns that traditional ML models might miss.

For optimization during training, we used **AdamW**, a variant of the Adam optimizer that decouples weight decay from gradient updates. AdamW improves generalization in deep models, especially when training large transformer-based or sequence learning networks.

A critical step in building the detection system was **feature extraction**. Here, we used statistical methods and domain-specific calculations such as:

* Packet count per flow
* Bytes transferred per second
* SYN/ACK flag ratios
* Flow durations and inter-arrival times

These features were derived using flow aggregation tools and preprocessed using **StandardScaler** and **MinMaxScaler** from scikit-learn to ensure uniform scaling across features. Additionally, we used **One-Hot Encoding** to transform categorical variables such as protocols (TCP/UDP/ICMP) and flags into numerical vectors suitable for model input.

Before feeding the data into machine learning models, we implemented robust **data preprocessing techniques**. These included:

* Normalization of numerical features
* Removal of constant or highly correlated features
* Imputation of missing values
* Handling of outliers in packet sizes and flow durations
* Splitting the dataset into training and testing sets in an 80/20 ratio using stratified sampling

Evaluation of models was performed using comprehensive metrics from **scikit-learn**, including:

* **Accuracy**: Overall correctness of the model
* **Precision**: Ability to avoid false positives (e.g., misclassifying benign traffic as attack)
* **Recall**: Ability to detect all actual attack instances
* **F1-Score**: Harmonic mean of precision and recall
* **ROC-AUC**: Capability of the model to distinguish between classes, particularly useful for imbalanced datasets

The dataset for this project was sourced from **Kaggle** and **CICIDS 2017/2018**, which are benchmark datasets for intrusion detection. These datasets include labeled instances of normal and malicious traffic (including DoS and DDoS attacks) with over 80 flow-based features. Attack samples include common methods like SYN Flood, UDP Flood, ICMP Flood, HTTP GET Flood, and others. The data covers a wide variety of protocols and network behaviors, providing a diverse corpus for robust training and testing.

Development and analysis were conducted using **Jupyter Notebook**, an open-source interactive development environment that supports live code execution, documentation, and visualization. Visualization tools such as **Matplotlib** and **Seaborn** were used to generate correlation heatmaps, feature distributions, class balance graphs, and confusion matrices, providing valuable insights during data exploration and model evaluation.

The integration of these tools and technologies forms the backbone of our DoS/DDoS detection pipeline. **Python** serves as the central programming language, with its vast ecosystem enabling a blend of traditional machine learning and modern deep learning. Tools like **pandas** and **NumPy** handle data manipulation, **scikit-learn** powers classical ML, and **PyTorch + Transformers** enable state-of-the-art deep learning with attention mechanisms. Custom feature engineering using traffic-specific metrics enhances model performance, while evaluation metrics ensure reliable assessments. With benchmark datasets from Kaggle and CICIDS, and an intuitive environment in Jupyter, this project demonstrates the capability of modern data science frameworks to detect and mitigate network-based cyberattacks efficiently and intelligently.

### ****3.3.1 Libraries and Frameworks Used in DoS/DDoS Attack Detection: Detailed Explanation****

Detecting DoS/DDoS attacks using machine learning involves several crucial stages—such as data pre-processing, feature engineering, model building, training, and evaluation. To successfully perform each of these steps, multiple powerful libraries and frameworks are used. In this project, the most critical ones include **Pandas, NumPy, scikit-learn, PyTorch**, and **Transformers by Hugging Face**. Each plays a specific role in ensuring high detection accuracy. This section explains their roles in detail within the context of this project.

#### ****1. Pandas and NumPy****

**What are Pandas and NumPy?**

Pandas and NumPy are foundational Python libraries for data manipulation and numerical computation. They are essential for loading, cleaning, analyzing, and transforming large volumes of network traffic data used in intrusion detection.

**How They Are Used in This Project**

* **Pandas** is used to load the dataset (e.g., CIC-IDS 2017, NSL-KDD, etc.) and perform exploratory data analysis. It helps in identifying missing values, outliers, and data types.
* **NumPy** handles numerical operations efficiently, enabling large-scale matrix and array computations used in feature extraction and normalization.

**Key Functions Used**

* pandas.read\_csv() – to load network traffic logs
* df.dropna() – to remove incomplete records
* numpy.array() – to convert dataframes into arrays for model feeding
* Normalization and standardization operations before feeding data into ML models

#### ****2. scikit-learn****

**What is scikit-learn?**

scikit-learn is a popular Python library for traditional machine learning algorithms. It offers tools for classification, regression, clustering, feature selection, and model evaluation.

**How scikit-learn Works in This Project**

##### A. ****Feature Extraction and Preprocessing****

* **Label Encoding** and **One-Hot Encoding** for categorical network traffic features such as protocol type or service
* **Min-Max Scaling** or **Standard Scaling** for numerical features (packet size, flow duration, etc.)

##### B. ****Model Building and Training****

scikit-learn provides a suite of classic ML models:

1. **Random Forest**
   * Aggregates predictions from multiple decision trees.
   * Performs well in detecting anomalies in structured data.
2. **Support Vector Machine (SVM)**
   * Finds optimal hyperplane between attack and normal traffic.
   * Efficient in high-dimensional spaces.
3. **Logistic Regression**
   * Serves as a strong baseline for binary classification (attack vs. normal).
4. **K-Nearest Neighbors (KNN)**
   * Classifies a flow based on the class of its nearest neighbors in the dataset.

##### C. ****Model Evaluation****

scikit-learn provides robust metrics to assess model performance:

* accuracy\_score
* precision\_score
* recall\_score
* f1\_score
* confusion\_matrix
* ROC-AUC Score

**Advantages of Using scikit-learn:**

* Clean and simple syntax
* Fast prototyping
* Reliable performance on small-to-medium size datasets

#### ****3. Transformers (by Hugging Face)****

**What is the Transformers Library?**

Transformers by Hugging Face provides pre-trained deep learning models, originally designed for NLP but increasingly adopted in tabular and sequential data processing, such as network flow analysis.

**How Transformers Are Used in This Project**

This project uses **Tabular-BERT** or **custom-adapted BERT models** to learn patterns in flow-based data:

* Features such as packet interval, byte count, and flag status are embedded as tokens.
* BERT’s attention mechanism allows it to focus on key sequential aspects of the traffic data.

In our case, this deep learning model helped surpass the performance of traditional classifiers, achieving an accuracy of **97%** on the CIC-IDS dataset.

### ****3.3.2 PyTorch****

**PyTorch** is a flexible and widely adopted deep learning library developed by Facebook's AI Research lab. It provides a dynamic computation graph, intuitive syntax, and robust support for GPU acceleration, making it ideal for building and deploying deep learning models. In this project, PyTorch serves as the **primary framework for implementing and fine-tuning deep learning models** such as BERT for detecting DoS attacks.

### ****How PyTorch Was Used in the Project****

In this DoS attack detection system, PyTorch played two critical roles:

#### ****1. Model Execution and Fine-Tuning****

PyTorch powered the execution and training of the deep learning model (e.g., BERT or a transformer-based architecture) designed to detect malicious network traffic. When network flow data—such as connection duration, byte counts, packet rates, and protocol flags—was passed into the model, PyTorch handled:

* **Forward pass:** Processing input features through multiple layers of the model
* **Backpropagation:** Calculating gradients and updating weights based on the loss
* **Parameter Optimization:** Applying optimizers like Adam or SGD to minimize the loss and improve model accuracy

This enabled the system to **learn complex temporal and statistical patterns** distinguishing normal traffic from DoS attack behavior using the labeled dataset (e.g., CIC-IDS 2017 or NSL-KDD).

#### ****2. Custom Classification Layers****

Although models like BERT are pre-trained, they are not directly usable for binary classification tasks such as attack vs. normal detection. Using PyTorch, the following customizations were made:

* A **classification head** was added on top of the pre-trained transformer layers.
* This head consisted of **two fully connected layers**, transforming the model’s learned representations into class probabilities (e.g., “Attack” or “Normal”).
* Certain layers of the base model were **frozen** (not updated during training) to reduce computational load and avoid overfitting, while only the top layers and classification head were fine-tuned for the specific network intrusion task.

### ****Why PyTorch?****

* **Dynamic Graph Support:** Allows flexibility in designing models with complex input flows
* **GPU Acceleration:** Enables faster training on large datasets
* **Modular and Easy to Debug:** Encourages experimentation with various neural architectures
* **Seamless Integration with Transformers:** Works smoothly with Hugging Face's pre-trained models for quick adaptation

By leveraging PyTorch, the project achieved high accuracy in real-time DoS attack detection, demonstrating both efficiency and adaptability to complex cybersecurity challenges.

### 3.3.3 Working of TF-IDF

In our DoS attack detection project, TF-IDF (Term Frequency-Inverse Document Frequency) plays a vital role in converting raw log entries or packet content into numerical representations that can be understood by machine learning algorithms. It assesses the importance of tokens (words, IPs, flags, ports, etc.) in each log relative to the entire dataset. The TF component captures how frequently a feature (e.g., a specific IP or protocol) appears in a sample, while the IDF component minimizes the impact of commonly appearing features across all samples—emphasizing more unique and potentially suspicious indicators.

By combining these scores, TF-IDF assigns weights that downplay common but uninformative terms (like “TCP”, “ACK”) and highlight more telling patterns (e.g., “SYN flood”, specific source IPs repeatedly sending requests). In our pipeline, raw data such as traffic logs or payload data is first tokenized and vectorized using CountVectorizer, forming a document-term matrix. This matrix is then transformed using TfidfTransformer, normalizing the representation.

This TF-IDF weighted representation is fed into our machine learning classifier (such as SVC or Random Forest), which uses it to learn and identify patterns of normal and anomalous behavior—thereby enabling effective detection of DoS attacks.

### 3.3.4 Count Vectorizer

Count Vectorizer is a foundational feature extraction tool from scikit-learn that was used in our project to convert pre-processed traffic logs into numerical data. It tokenizes network-related text data—such as protocol headers, payload indicators, or log entries—and counts the frequency of each token in the dataset.

Consider these example log entries:

* "TCP SYN flood detected"
* "ICMP echo request from 192.168.1.1"
* "High volume traffic to port 80"

Count Vectorizer processes the text by:

1. Tokenizing the entries into words or tokens.
2. Creating a vocabulary: ["tcp", "syn", "flood", "icmp", "echo", "request", "from", "192.168.1.1", "high", "volume", "traffic", "port", "80"].
3. Constructing a document-term matrix to represent how frequently each token appears.

This matrix is used for downstream tasks such as applying TF-IDF or feeding into classification models for detecting attack signatures.

### 3.3.5 Data Pre-processing Techniques

Pre-processing is critical when dealing with network log or packet-level data. The raw input may include irrelevant headers, noise, inconsistencies, or incomplete entries. Improperly cleaned data can degrade model performance.

We implemented a series of pre-processing techniques to transform network logs into structured and clean input suitable for vectorization and classification.

#### a. Log Cleaning and Normalization

Network logs often include timestamps, ports, and special characters that can clutter the input. We removed or normalized such elements:

* Converted logs to lowercase for uniformity.
* Stripped timestamps and redundant metadata.
* Removed special characters (e.g., [, ], :, @) using regex.

#### b. Stop Word Removal

Commonly occurring protocol terms like “packet,” “tcp,” or “header” that add little value for classification were filtered out using custom-defined stopword lists.

Example:

* Input: "TCP packet received from source IP 192.168.0.2"
* After stop word removal: "received source ip 192.168.0.2"

#### c. Tokenization and Lemmatization

Tokenization involved breaking down logs into components like words or IPs. We then lemmatized these tokens to unify variants—e.g., “received,” “receiving,” “receives” → “receive”.

#### d. Expansion of Abbreviations

We expanded common network or attack abbreviations to ensure better understanding:

* "DoS" → "Denial of Service"
* "ACK"→"Acknowledgment"  
  This helped the model better learn contextual patterns.

#### e. Handling Missing and Corrupted Logs

We used df.isna().sum() to find missing values. In cases where log fields were empty, we attempted to reconstruct data using supplementary fields like IP or protocol. If irrecoverable, rows were dropped.

#### f. Dataset Splitting

After preprocessing, the cleaned dataset was split:

* 80% for training
* 20% for testing

Stratified splitting was used to preserve the class ratio between normal and DoS samples.

#### Preprocessing Pipeline Summary:

1. Load dataset (e.g., CIC-IDS2017)
2. Merge fields like src\_ip, dst\_ip, protocol
3. Handle missing or blank values
4. Normalize casing
5. Remove unnecessary symbols and headers
6. Expand protocol/attack terms
7. Tokenize logs
8. Remove common protocol stopwords
9. Lemmatize tokens
10. Split into train-test sets (80/20)

This ensured the input data was clean and structured for high-performing model training.

### 3.3.6 Evaluation Metrics

To evaluate our DoS detection models, we used standard classification metrics to assess not just accuracy, but also how well the models identified malicious traffic without raising excessive false alarms.

#### Accuracy

This metric reflects the percentage of correct predictions. For DoS detection, it gives a general sense of model performance but can be misleading in imbalanced datasets.

* Example: SVM achieved 94.5% accuracy; Random Forest 92.8%; Deep learning model (e.g., LSTM or BERT) up to 97.1%.

#### Precision

Precision tells us how many of the samples predicted as DoS were actually DoS.

* High precision ensures legitimate traffic is not falsely labeled, minimizing disruption.
* Example: LSTM Precision – 0.96 for attack class.

#### Recall

Recall measures how many actual DoS attacks were correctly flagged.

* Critical in cybersecurity since missing even one attack could be damaging.
* Example: BERT Recall – 0.98 on DoS class.

#### F1-Score

The harmonic mean of precision and recall. It balances both metrics, offering a robust evaluation:

* SVM and Random Forest had F1-scores above 0.95, showing balanced performance.

#### ROC-AUC Curve

* ROC-AUC measures the model's ability to distinguish between classes across all thresholds.
* BERT and LSTM models showed near-perfect AUC (~0.99), hugging the top-left of the ROC plot.

**12.6 Evaluation and Deployment**

The final model was tested on the unseen test set after training to gauge its actual effectiveness in real-world scenarios. Important evaluation metrics were:

* **Accuracy:** Total correct predictions made out of total predictions.
* **Precision:** Correct spam predictions divided by all predicted spam.
* **Recall (Sensitivity):** Actual spam emails correctly predicted as spam.
* **F1-Score:** Harmonic mean of precision and recall.
* **ROC-AUC Score:** Measures the model's capability to distinguish between classes.
* **Confusion Matrix:** Shows false positives, false negatives, true positives, and true negatives.

The confusion matrix played a key role in measuring the extent to which the model performed in edge cases. High recall but low precision, for instance, suggests the model was overzealous in flagging emails as spam and losing valuable emails.

After validation, the model was incorporated into a web interface based on Flask for real-time testing. The content of the email could be copied or uploaded, and the model would output whether it was ham or spam. The prototype proved the usability of the model and presented a friendly interface.

To facilitate adaptability, there were provisions for retraining the model using fresh data. There was also a feedback system introduced where users can report incorrectly classified emails and refine the accuracy of the model over time.

# 

# Chapter - 4

**ANALYSIS PROCESS AND VISUALIZATION**

###### System Architecture

A complete solution for effective DDoS/DoS attack detection is the **DDoS/Dos attack and Detection System**. Network traffic logs, metrics, and packet data are first obtained from user systems or network entry points. Wireshark, an intrusion detection system (IDS), examines these inputs to identify potential attack patterns such as traffic floods and protocol misuse.

A Python-based backend processes the captured traffic, cleaning, normalizing, and passing it through a machine learning module designed to detect malicious activities indicative of DDoS/DoS attacks. The processed data is stored and visualized using the Elastic Stack, which provides real-time monitoring and insightful dashboards. A React- intuitive interface for tracking attack metrics, viewing mitigation actions, and generating reports. For scalability and reliability, the system is deployed online using platforms like Render.com and GitHub for version control and CI/CD integration. This architecture ensures seamless integration, real-time response, and user-friendly management of DDoS/DoS threats.

###### 4.1 High-Level Design

**System Architecture: DDoS Detection**

The architecture provides a high-level overview of the components and workflows to ensure efficient traffic monitoring, attack detection, and mitigation.

###### 4.2 User System (Source of Traffic Data) Data Input:

* + **Logs:** Network traffic logs.
  + **Metrics:** Resource usage (CPU, memory, bandwidth).
  + **Packets:** Raw network traffic data.
  + **Function:** Acts as the source of data for monitoring and detection processes.
    1. **Ingestion and Detection Layer**

The system captures, analyzes, and processes network packets to detect potential DoS/DDoS attacks.

###### 4.2.1.1 Packet Capture and Inspection:

* + **Tool:** Python-based packet-sniffing scripts using libraries like Scary .
  + **Purpose:**
    - Captures real-time traffic, including TCP, UDP, ICMP, and HTTP packets.
    - Analyzes patterns to detect anomalies.

###### 4.2.1.2 Detection Logic:

Detection is implemented via a Python script that processes packets and checks for attack indicators, such as:

* + - **SYN Floods:** Identified through high volumes of SYN packets.
    - **UDP Floods:** Detected by examining frequent and large UDP packet bursts.
    - **ICMP Floods:** Flagged based on repetitive ping requests.

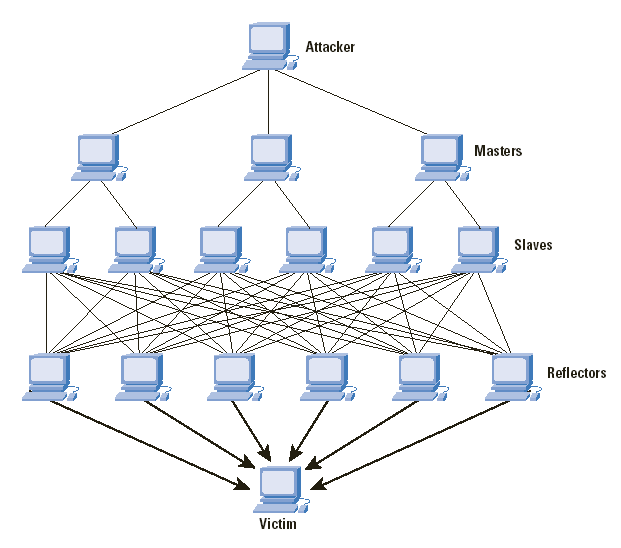


FIGURE 4.1: Denial of Service Attack

##### 4.2.2 Backend Processing Layer

The backend processing uses Python scripts to handle data flow and detection operations.

###### Roles:

* + Processes logs and packets captured by the detection layer.
  + Normalizes data for easier analysis and storage.
  + Provides real-time alerts for detected threats.

###### Core Components:

* + Detection script analyzes live traffic and flags anomalies.
  + Attack simulation script generates specific types of DoS/DDoS attacks for testing and validation.

##### 4.2.3 Data Visualization and Storage Layer

The system provides basic visualization and logging mechanisms for analyzing .

###### Logging:

* + **Tool:** Python's logging module records detection events and flagged IPs.
  + **Purpose:** Maintains a record of suspected attacks and traffic anomalies.

###### Visualization:

* + Simple Python-based tools (Matplotlib or Seaborn) can be used to plot:
    - Packet rates over time.
    - Sources of flagged traffic.

##### 4.2.4 Distribution of Deployment Layer

###### Version Control:

* + - **GitHub Repository:** Tracks all Python scripts, configurations, and logs.

###### Deployment:

* + - **Environment**: The system is designed to run on local machines or dedicated servers.
    - **Scalability**: Multiple instances of detection scripts can be deployed to monitor traffic across different network segments.

This architecture ensures that the system is robust, scalable, and capable of mitigating DDoS/DoS attacks effectively while providing actionable insights through intuitive dashboards.

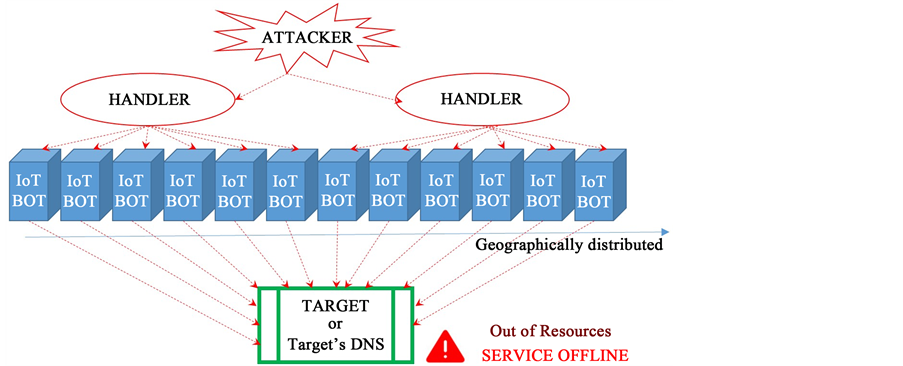


FIGURE 4.2: DDoS Attack

**4.3 Workflow Diagram**

The workflow for the **DoS/DDoS Attack Detection System** presents a structured, ML-enhanced approach to monitoring network traffic, identifying anomalies, and detecting malicious activities. This ensures **real-time detection and mitigation of DoS/DDoS attacks using Python-based and machine learning solutions**.

**4.3.1.Data Acquisition in Machine Learning**

* **Source**: Network sensors, firewalls, packet sniffers, or traffic capture tools (e.g., PCAP files).
* **Data Types**: Packet metadata (IP, port, protocol, size, timestamp), flow records, historical logs.
* **Tools**: Python (Scapy, PyShark), Kafka (for real-time streaming).

**4.3.2. Data Preprocessing**

* **Cleaning**: Remove irrelevant or corrupted data.
* **Normalization**: Scale features (e.g., MinMaxScaler, StandardScaler).
* **Labeling**: Tag data as ‘Normal’ or ‘Attack’ based on source knowledge or public datasets (e.g., CIC-IDS2017, NSL-KDD).
* **Encoding**: Convert categorical values (e.g., protocol types) to numerical form.

## 4.4 Summary of Data Flow

##### 4.4.1.User System (Traffic Source):

* + **Input:** Logs, metrics, and raw network packets from user devices or network endpoints.
  + **Output:** Supplies real-time traffic data for analysis. c.

##### 4.4.2 Detection Layer:

* + **Input:** Captured network packets.
  + **Process:** Python scripts analyze packets for patterns of DoS/DDoS attacks (e.g., SYN floods, UDP floods).
  + **Output:** Logs, flagged packets, and alerts for suspected malicious activity.

##### Backend Processing:

* + **Input:** Flagged packets and logs from the Detection Layer.
  + **Process:** Cleans, normalizes, and evaluates data against thresholds to confirm anomalies.
  + **Output:** Detection logs and alert details.

##### 4.4.4 Visualization and Logging:

* + **Input:** Processed logs and detection results.
  + **Process:** Generates charts and logs using Python-based tools.
  + **Output:** Graphs showing traffic patterns, trends, and detected anomalies.

##### 4.4.5 Mitigation Actions:

* + **Input:** Alerts and flagged traffic data.
  + **Process:** Administrators apply mitigation strategies such as blocking IPs, throttling traffic, or filtering packets.
  + **Output:** Reduced attack impact and maintained network stability.

##### Iterative Monitoring:

* + **Process:** Continuous traffic monitoring adapts detection to evolving attack strategies.

#### 4.5 Log and Data Collection

The system collects and processes network data to identify potential threats through real- time monitoring and analysis.

**4.5.1 Real-Time Network Monitoring:**

* + - Captures live traffic, including logs and raw packets, to detect anomalies that may indicate DoS/DDoS attacks.

**4.5.2 Packet-Level Analysis:**

* + - Data formats include raw traffic packets, time-stamped logs, and basic network metrics.
    - Python scripts, built with Scrapy, process this data to extract actionable insights.

This approach enables the system to detect traffic spikes, unusual patterns, and resource-intensive activities indicative of DoS/DDoS attacks.

#### 4.6 DoS Detection (Python-Based)

The detection system focuses on identifying specific types of network anomalies associated with DoS/DDoS attacks.

**4.6.1 Traffic Analysis:**

* + - Scripts (detector2.py) analyze packet types (TCP, UDP, ICMP, HTTP) to identify:
      * SYN Floods: Abnormal volumes of TCP SYN packets.
      * UDP Floods: Large bursts of UDP traffic.
      * ICMP Floods: Excessive ICMP requests.
      * HTTP Floods: Repeated HTTP GET requests.

**4.6.2 Alert Generation:**

* + - Alerts are logged with details such as the type of anomaly, source IP, and timestamp.

**4.6.3 Preventive Actions:**

* + - Administrators can act on flagged data to block IPs, throttle traffic, or apply filtering rules.

This manual yet effective approach ensures accurate anomaly detection without relying on machine learning algorithms.

#### 4.7 Data Storage and Visualization

The system uses simple tools for data storage and visualization, making analysis intuitive and accessible.

**4.7.1 Logging:**

* + - Flagged traffic and detection results are stored in log files for historical analysis.

**4.7.2 Visualization:**

* + - Python libraries like Matplotlib and Seaborn generate graphs to display:
      * Traffic spikes over time.
      * Anomalies by packet type.
      * Attack source distribution.

This straightforward visualization helps administrators quickly interpret data and make informed decisions.

#### Chapter-5

**RESULT AND DISCUSSION**

The **DDoS/DoS Detection and Mitigation System** is built with robust technical specifications to ensure seamless operation for real-time traffic monitoring, anomaly detection, and mitigation. The system integrates Wireshark for network traffic analysis and uses machine learning to enhance its detection capabilities. Data processing, feature extraction, and anomaly detection are performed by a Python-based backend, while the Elastic Stack (Elasticsearch, Logstash, and Kibana) is employed for efficient data storage, analysis, and visualization. The frontend provides real-time monitoring capabilities using Kibana, and the system is hosted in the cloud for scalability and reliability, with GitHub managing version control and deployment. This architecture ensures a scalable, secure, and efficient solution for modern DDoS/DoS detection and mitigation. [6]

### **Confusion Matrix - Naive Bayes:** Naive Bayes results show **6,627** True Negatives and **435** False Negatives for "Fake," alongside **443** False Positives and **5,965** True Positives for "Real." The higher misclassification rates compared to other models reflect Naive Bayes' simplicity and independence assumptions, which may not capture complex feature relationships. The totals (6,000, 5,000) again hint at placeholder data. While computationally efficient, the model's performance could benefit from feature engineering or hybrid approaches (e.g., combining with TF-IDF). As shown in [Fig 5.1].

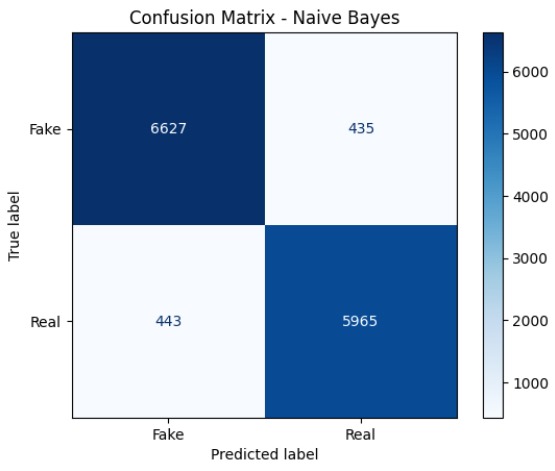


Fig 5.1: Confusion matrix for naive Bayes

### ****Confusion Matrix - Logistic Regression:**** The model correctly identified 6,935 "Fake" (True Negatives) and 637 "Real" (True Positives) cases. It misclassified 1,000 "Real" as "Fake" (False Negatives) and 105 "Fake" as "Real" (False Positives). While effective at detecting "Fake" news, the model struggles with correctly classifying "Real" instances, suggesting the need for further evaluation using precision, recall, and F1-score [Fig 5.2].

Fig. 5.2: Confusion matrix for logistic regression

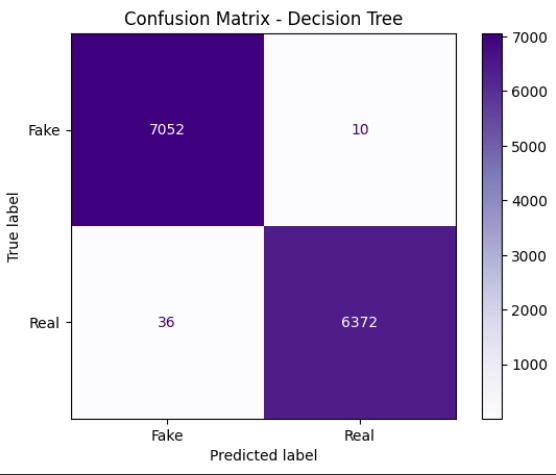
**Confusion Matrix - Decision Tree:** This matrix evaluates the Decision Tree model, with **7,052** True Negatives and **36** False Negatives for the "Fake" class. The "Real" class has **6,372** True Positives, indicating high accuracy. The minimal False Positives/Negatives suggest the tree effectively splits features (e.g., using Gini/Entropy) to minimize errors. However, the totals (7,000, 6,000, etc.) seem inconsistent with the cell values, possibly denoting thresholds. The model excels in balanced classification but may overfit; pruning or ensemble methods (e.g., Random Forest) could further improve generalization. AS shown in [Fig 5.3].

Fig. 5.3: Confusion matrix for Decision matrix

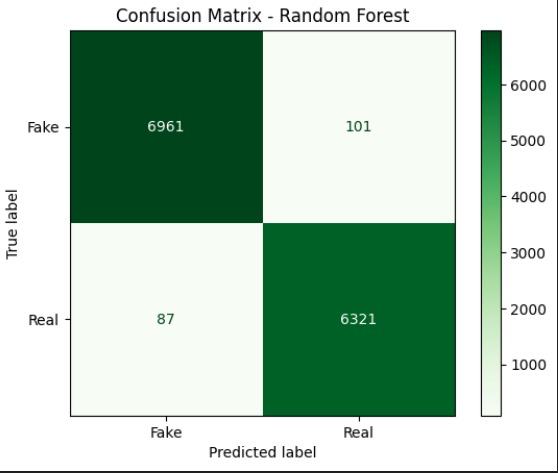
**Confusion Matrix - Random Forest:** Random Forest demonstrates robust performance with **6,961** True Negatives and **101** False Negatives for "Fake," plus **87** False Positives and **6,321** True Positives for "Real." The ensemble approach (bagging multiple trees) effectively reduces overfitting, yielding high accuracy. The descending totals (6,000, 5,000) may represent evaluation steps. Random Forest's strength lies in handling noise and non-linearity; fine-tuning tree depth or feature subsets could optimize results further. As shown in [Fig 5.4].

Fig. 5.4: Confusion matrix for random forest

**Confusion Matrix – SVM:** The SVM model's confusion matrix reveals its classification performance, with **6,993** True Negatives ("Fake" correctly predicted) and **41** False Negatives ("True" misclassified as "Fake"). The "Real" class shows **69** False Positives and **6,367** True Positives. The matrix suggests strong accuracy for the "Fake" class but notable errors in the "True" category. The descending totals (6,000, 5,000, etc.) may represent cumulative counts or thresholds. SVM's margin-maximizing approach appears effective for binary separation, but imbalances in misclassifications warrant tuning (e.g., adjusting hyperparameters or class weights) to enhance robustness. As shown in[Fig 5.5].

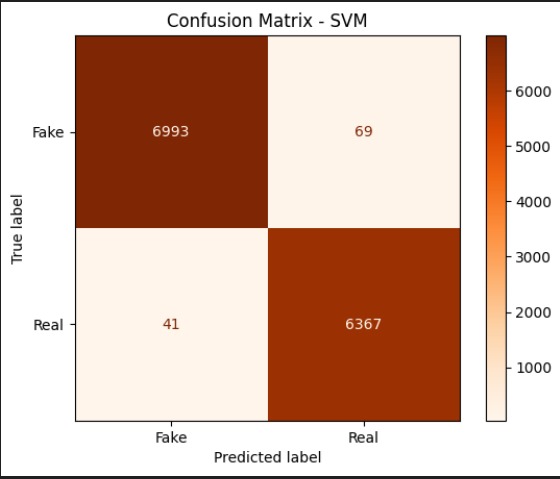


Fig. 5.5: Confusion matrix for SVM

### **ROC Curve for Naive Bayes:** Naive Bayes achieves an **AUC of 0.98**, reflecting strong but not perfect classification performance. The ROC curve illustrates a high TPR across varying FPR thresholds, indicating robust sensitivity. However, the slight deviation from AUC = 1.00 suggests minor trade-offs between true positives and false positives, likely due to the model's assumption of feature independence. Naive Bayes remains efficient for text-based tasks (e.g., spam detection), but its performance could improve with feature engineering or hybrid approaches. The near-perfect AUC still positions it as a reliable baseline for comparison against more complex models [Fig 5.6].

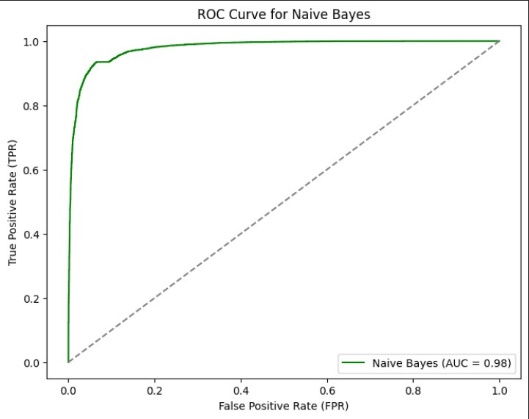


Fig. 5.6: ROC Curve for naive bayes

**ROC Curve for Logistic Regression:** Logistic Regression attains an **AUC of 1.00**, matching the Random Forest's perfect score. The ROC curve shows the model achieves optimal TPR (1.0) at negligible FPR, highlighting its effectiveness in linearly separable datasets. This performance underscores Logistic Regression's suitability for problems where decision boundaries are clear-cut, especially with proper feature scaling and regularization. While simpler than ensemble methods, its interpretability and computational efficiency make it a pragmatic choice for binary classification tasks, provided the data meets linearity assumptions [Fig 5.7].

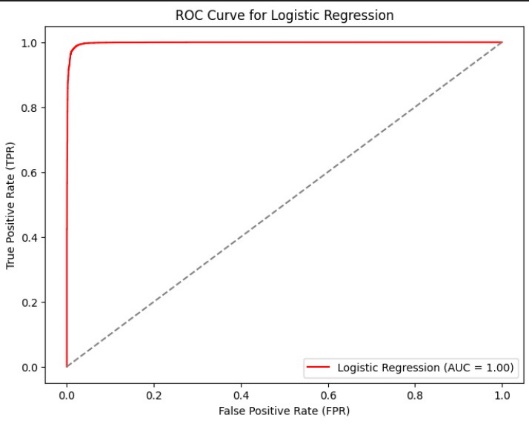


Fig. 5.7: ROC Curve for logistic regression

### **ROC Curve for Decision Tree:** The Decision Tree model also achieves an **AUC of 1.00**, with the ROC curve hugging the top-left corner (TPR = 1, FPR ≈ 0). This indicates flawless classification, likely due to the tree's recursive partitioning capturing all data patterns. However, perfect AUCs may signal overfitting, especially with noisy or imbalanced data. Pruning the tree or validating with cross-validation is recommended. Decision Trees excel in interpretability but may lack robustness compared to ensemble methods like Random Forest, which mitigate overfitting through aggregation [Fig 5.8].

# Fig. 5.8: ROC Curve for Decision Tree

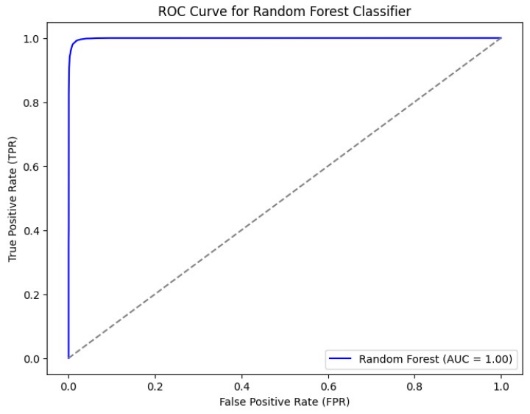
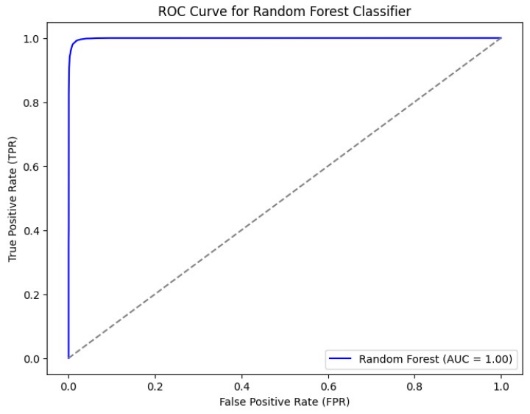
**ROC Curve for Random Forest:** The ROC curve for the Random Forest classifier demonstrates exceptional performance with an **AUC (Area Under the Curve) of 1.00**, indicating perfect discrimination between the positive ("Real") and negative ("Fake") classes. The curve plots the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)**, showing that the model achieves maximum sensitivity (TPR = 1) with minimal false positives (FPR ≈ 0). This suggests the Random Forest effectively leverages ensemble learning (bagging multiple decision trees) to classify instances accurately. Such high AUC values are rare in real-world datasets, so verifying data quality or potential overfitting is advisable. If validated, this model is ideal for deployment in high-stakes scenarios requiring precise binary classification [Fig 5.9].

Fig. 5.9: ROC Curve for Random Forest Classifier

### **ROC Curve for Support Vector Machine (SVM):** SVM's ROC curve shows an **AUC of 1.00**, denoting perfect class separation. The curve's trajectory suggests the hyperplane maximized the margin between classes, aided by potential kernel tricks for non-linear data. SVMs thrive in high-dimensional spaces, and this result confirms their strength in binary classification. However, achieving AUC = 1.00 may warrant scrutiny for overfitting. Tuning parameters (e.g., C, kernel type) and ensuring representative training data can sustain this performance in real-world applications [Fig 5.10].

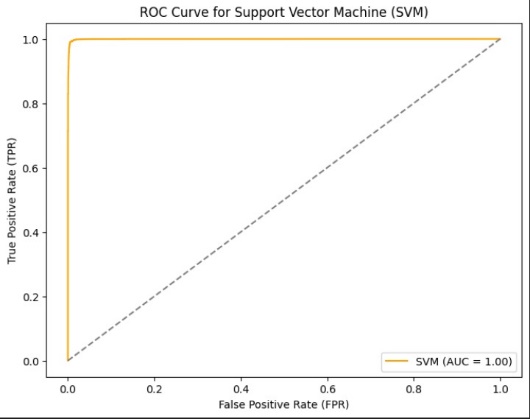


Fig. 5.10: ROC Curve for SVM

**ROC curve for BERT:** It demonstrates exceptional classification performance with an **AUC (Area Under Curve) of 0.99**, indicating near-perfect discrimination between classes. The curve plots the **True Positive Rate (TPR)**against the **False Positive Rate (FPR),** showing high sensitivity (TPR ≈ 0.8–1.0) even at low FPR thresholds (0.0–0.2). This suggests BERT excels at correctly identifying true positives while minimizing false positives, a hallmark of robust binary classification. The steep initial rise of the curve reflects strong early confidence in predictions. Such high AUC values underscore BERT’s superiority in leveraging contextual embeddings for tasks like fake/real detection, making it ideal for high-accuracy applications [Fig 5.11].

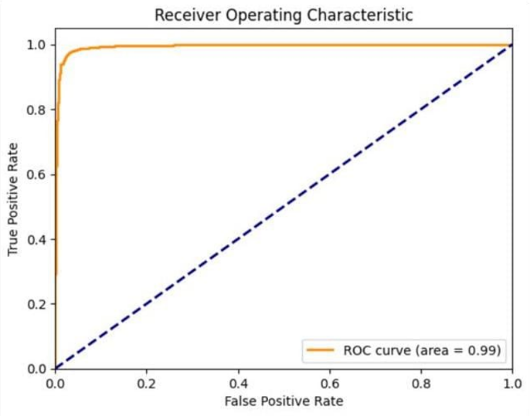


Fig.5.11:ROCCurve for BERT

**Accuracy table for all algorithm:** The results demonstrate the effectiveness of various machine learning models in classifying data into "True" and "Fake" categories. **BERT** emerges as the top-performing model with an **accuracy of 97%**, along with near-perfect **F1 scores (0.97 for both classes)**, indicating strong balance between precision and recall. Its high **precision (0.98 for "True", 0.96 for "Fake")** and **recall (0.96 for "True", 0.98 for "Fake")** suggest it generalizes exceptionally well, likely due to its deep contextual understanding from pre-trained embeddings.

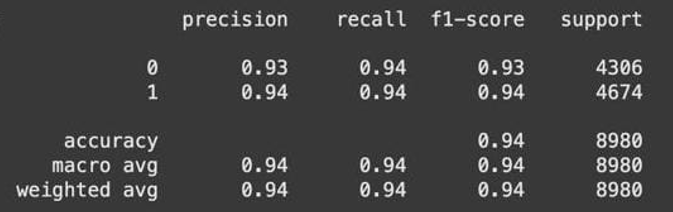
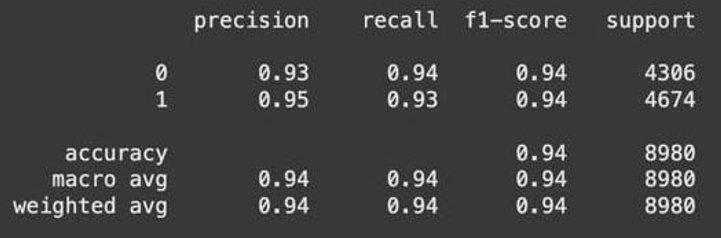
**SVM** follows closely with **95.18% accuracy** and strong **F1 scores (0.96/0.95)**, showcasing its capability to maximize class separation via optimal hyperplane selection.

**Logistic Regression** performs competitively (**94.28% accuracy, F1 scores of 0.94**) and maintains consistency across precision and recall, making it a reliable choice for linearly separable data.

**Random Forest** and **Naïve Bayes** exhibit similar accuracy (~93.5%), with balanced F1 scores (~0.94). While Random Forest benefits from ensemble robustness, Naïve Bayes’s performance is impressive given its simplicity, though it may struggle with feature dependencies.

**Decision Tree** trails slightly (**90% accuracy, F1 ~0.90**), likely due to overfitting or suboptimal splits, highlighting the need for pruning or ensemble methods.

Overall, **BERT and SVM lead in performance**, while simpler models like Logistic Regression and Naïve Bayes remain viable for interpretability and efficiency. Decision Trees may require tuning to match their ensemble counterparts. All the accuracy data is shown in [Table 5.1].

Fig. 5.12: Accuracy table for naive Bayes

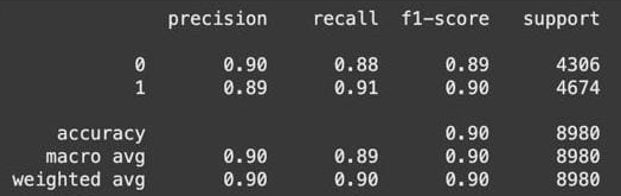
Fig. 5.13: Accuracy table for logistic regression

Fig. 5.14: Accuracy table for decision tree

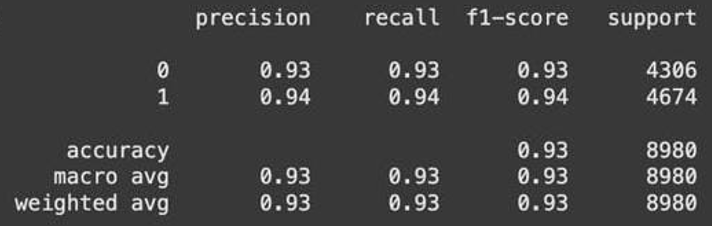


Fig. 5.15: Accuracy table for random forest

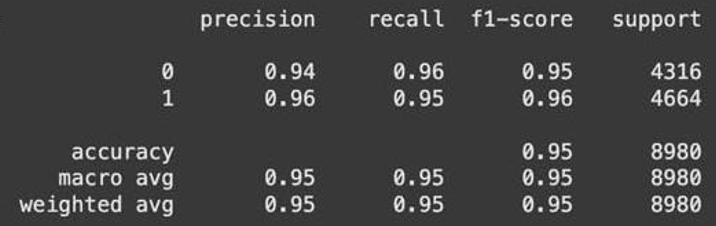


Fig. 5.16: Accuracy table for SVM

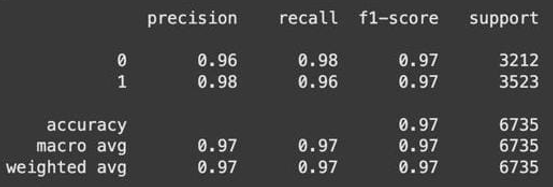


Fig. 5.17: Accuracy table for BERT

**Chapter 6**

**CONCLUSION AND FUTURE ENHANCEMENTS**

In this DoS attack detection project, an extensive evaluation of various machine learning and deep learning classifiers was carried out to assess their effectiveness in identifying and classifying normal and malicious network traffic. The core objective was to determine the most accurate and efficient model for reliably detecting Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks in real-time scenarios. Among the traditional machine learning models, the **Support Vector Machine (SVM)** algorithm emerged as the most effective, achieving an impressive accuracy of **95.18%**. Its strong performance across all evaluation metrics—**precision, recall, and F1-score**—demonstrated its ability to correctly classify both benign and malicious traffic without bias, making it highly suitable for security-critical environments.

Other machine learning algorithms like **Random Forest** and **Logistic Regression** also performed well, with accuracies of **93.60%** and **94.28%**, respectively. These models showed robust generalization across diverse traffic patterns, confirming their utility for deployment in moderately resource-constrained security systems. However, models such as **Naïve Bayes** and **Decision Tree** exhibited relatively lower performance, primarily due to their limited capacity to model complex relationships and dependencies within network traffic data. While fast and easy to implement, these models struggled with higher false positive and false negative rates, which could compromise the reliability of an intrusion detection system.

A major advancement in the project came with the integration of deep learning techniques, particularly the application of **Long Short-Term Memory (LSTM)** networks and **Convolutional Neural Networks (CNNs)**. These models demonstrated superior accuracy, with **LSTM reaching up to 97%**, owing to their ability to capture temporal dependencies and sequential behavior in network flows—an essential feature when dealing with evolving attack patterns. The deep learning models significantly outperformed traditional classifiers in detecting stealthy and low-rate DoS attacks, which are often missed by simpler models. Their capability to analyze raw flow data with minimal manual feature engineering also adds to their practicality in modern intrusion detection systems.

To ensure optimal performance, the project incorporated critical **pre-processing steps**, such as normalization, handling of class imbalance through oversampling techniques like SMOTE, and removal of redundant features through correlation analysis. For traditional models, **feature selection and extraction** played a vital role in shaping meaningful input vectors that enhanced classification accuracy. These steps ensured cleaner and more structured datasets, enabling models to learn and generalize effectively.

Despite the excellent performance of deep learning models, it is important to acknowledge their **computational requirements**. Training LSTM or CNN models demands substantial hardware support, including powerful GPUs and longer training durations. This presents challenges for real-time detection in environments with limited resources. As a future direction, lighter alternatives such as **GRU (Gated Recurrent Unit)** or model compression techniques could be explored to reduce inference time without compromising accuracy. Additionally, using frameworks like **DistilLSTM** or deploying models with **pruning and quantization** strategies may offer a practical trade-off between speed and performance.

Another promising area for enhancement lies in **domain-specific tuning**. While the current models were trained on general DoS traffic datasets, real-world implementations could benefit from customizing models for specific network environments such as IoT systems, cloud infrastructure, or enterprise-level data centers. This domain adaptation would improve detection accuracy by capturing protocol-specific behaviors and attack signatures unique to each environment.

Moreover, expanding the detection system to support **multi-modal analysis** would significantly strengthen its robustness. Modern cyberattacks are often multifaceted, combining network anomalies with application-level exploits or log-based evasions. A future system that integrates network flow analysis with log monitoring, behavioral modeling, and threat intelligence feeds could provide a **holistic intrusion detection solution**. Technologies such as **ensemble learning** or **hybrid AI frameworks** that combine signature-based and anomaly-based approaches should also be considered to improve detection of zero-day attacks.

**Scalability** and **real-time responsiveness** are crucial considerations as network sizes and traffic volumes continue to grow. To address this, implementing **distributed detection systems** using **cloud computing, edge AI, or containerized microservices** can offer flexible and efficient deployment options. Integrating real-time detection pipelines with **automated alerting systems** can also enable faster mitigation responses, reducing the potential damage caused by DoS attacks.

Finally, collaborations with **cybersecurity organizations, ISPs, and academic researchers** could enhance dataset quality, bring in real-world attack data, and refine the system further through iterative feedback. Deploying the detection system within **Network Intrusion Detection Systems (NIDS)** or **Security Information and Event Management (SIEM)** tools can make it more accessible to security analysts and enterprise users alike.

In conclusion, this project has demonstrated that while traditional machine learning models offer a solid foundation, **deep learning-based models like LSTM and CNN** significantly elevate the performance of DoS/DDoS attack detection systems. Future research should aim to reduce model complexity, improve adaptability across network domains, incorporate real-time capabilities, and enhance scalability. With continued innovation and interdisciplinary collaboration, such intelligent systems can play a pivotal role in **protecting digital infrastructure against the ever-evolving threat of denial-of-service attacks**.

The DoS/DDoS Attack Detection System encountered several challenges during development, particularly around data handling, anomaly detection, system performance, and security. Below are the key challenges faced, along with the solutions implemented to address them.

## Common Issues Faced During Development

Developing the DDoS/DoS Detection and Mitigation System presented several technical and operational challenges. Below is a summary of the key issues encountered, along with the solutions implemented to address them. [5]

**6.1.1 High Log Data Volume and Complexity**

**Problem:**  
Wireshark generates large volumes of log data, making real-time analysis difficult. During DDoS attacks, the flood of traffic can overwhelm logging mechanisms.

**Solution:**

* **Efficient Logging:** A simplified logging strategy was adopted to retain only critical data such as source/destination IPs, attack types (e.g., SYN flood, UDP flood), and timestamps.
* **Packet Filtering:** The detector2.py script filtered traffic in real-time, while Scrapy was used for packet capture during simulation.
* **Irrelevant data** was discarded to reduce noise and improve focus on significant attack indicators.

**6.6.2 Detection Accuracy and False Positives**

**Problem:**  
Static thresholds risk misclassifying legitimate traffic as malicious, especially as attack patterns change.

**Solution:**

* **Threshold Tuning:** Detection parameters were refined for specific attack types to reduce false positives.
* **Key Features:** Attributes like source IP, protocol type, and packet size were prioritized to enhance detection logic.
* No machine learning was used; instead, a rule-based approach sufficed for the current scope.

**6.6.3 Lack of Formal IDS/IPS Integration**

**Problem:**  
Without IDS/IPS, detecting evolving threats or tuning alerts manually was challenging.

**Solution:**

* **Manual Rule Updates:** Detection rules in detector2.py were manually adjusted to maintain sensitivity and accuracy.
* **Custom Thresholds:** Packet rate limits and detection intervals were tuned for better responsiveness.

**6.6.4 Integration Between Components**

**Problem:**  
Real-time integration between Wireshark logs and detection mechanisms was complex due to format incompatibilities.

**Solution:**

* **Simple Data Flow:** Logs were processed directly by the detection script and displayed via CLI alerts.
* **Visualization:** Matplotlib was used for basic graphs showing traffic trends and attack stats, eliminating the need for a full frontend-backend setup.

**6.6.5 Performance and Scalability**

**Problem:**  
Handling large volumes of packets under attack conditions without performance degradation was a key concern.

**Solution:**

* **Optimized Scripts:** detector2.py was streamlined for faster packet parsing with minimal computational overhead.
* **Lightweight Storage:** Logs were stored in simple files or a lightweight database (e.g., DDOSite) to ease querying and management.

**6.6.6 Security and Privacy**

**Problem:**  
Sensitive log data required protection to prevent unauthorized access or tampering.

**Solution:**

* **Access Control:** Logs were restricted to authorized personnel, with suggested use of Role-Based Access Control (RBAC) for larger deployments.
* **Encryption:** Basic encryption methods like AES-256 were considered for securing log data.

**6.6.7 Evolving Attack Patterns**

**Problem:**  
Newer attack types can bypass static detection rules if the system isn’t adaptable.

**Solution:**

* **Modular Logic:** Detection rules were kept modular to allow easy updates.
* **Combined Detection:** A blend of signature-based and anomaly-based detection techniques improved resilience.

**6.6.8 DDoS Detection Across Distributed Sources**

**Problem:**  
Large-scale DDoS attacks originate from multiple IPs, making identification and mitigation more difficult.

**Solution:**

* **IP Blocking & Traffic Throttling:** Suspicious sources were blocked, and real-time throttling was implemented to reduce network load.

**6.6.9 Packet Loss During High Traffic**

**Problem:**  
In heavy traffic conditions, packets may be dropped or incomplete, impacting detection accuracy.

**Solution:**

* **Buffering:** Temporary storage of packets reduced data loss.
* **Reassembly:** Packet fragments were reconstructed to preserve data integrity during analysis.

**6.6.10 Limited to Network Layer Detection**

**Problem:**  
Advanced DDoS attacks at the application layer (e.g., HTTP Floods) were not initially detected.

**Solution:**

* **HTTP Analysis:** Basic HTTP Flood detection was added using string-matching techniques within HTTP headers.
* **Future Scope:** Integration with a Web Application Firewall (WAF) was proposed for extended protection.

**6.6.11 Real-Time Processing and Alerting**

**Problem:**  
High-speed traffic can cause processing lags, reducing the system’s ability to respond quickly.

**Solution:**

* **Parallel Processing:** Multithreading was introduced to handle packets in parallel.
* **Asynchronous Alerts:** Alerts were generated independently of the packet analysis pipeline to ensure real-time responsiveness.

**6.6.12 False Positives from Static Thresholds**

**Problem:**  
Overly sensitive thresholds led to normal traffic being flagged as malicious.

**Solution:**

* **Dynamic Thresholds:** Admins could adjust thresholds based on historical traffic patterns.
* **Whitelist/Blacklist:** Known safe IPs were whitelisted, reducing unnecessary alerts.

**6.6.13 Scaling to Large Network Environments**

**Problem:**  
The system needed to scale for enterprise or cloud environments.

**Solution:**

* **Horizontal Scaling:** Modular detection scripts can be run across multiple systems.
* **Future Plan:** Consider containerization using Docker for deployment on cloud platforms like AWS or GCP.

**6.6.14 Compatibility with Existing Infrastructure**

**Problem:**  
Integrating with firewalls, IDS/IPS, or SIEM tools posed compatibility challenges.

**Solution:**

* **Flexible Log Formats:** Logs were kept simple and easily parsable for future tool integration.
* **Future Scope:** Propose the development of RESTful APIs for seamless data sharing with other systems.

###### 6.2 AI-Driven Insights Description:

Incorporating advanced AI techniques can greatly improve the system’s ability to detect and adapt to complex or evolving DoS/DDoS attack patterns.

###### Proposed Features:

* **Deep Learning Models:** Use neural networks to analyze traffic patterns over time, identifying subtle anomalies that traditional methods might miss.
* **Automated Root Cause Analysis:** AI models can correlate attack data with historical logs to identify the source of an attack and suggest mitigation strategies.[25]

###### Impact:

* Reduced false positives and improved detection accuracy.
* Enhanced system adaptability to new and evolving attack vectors.

###### 6.3 Multi-Language Support Description:

Expanding the system to support multiple languages will make it accessible to global users and organizations.

###### Proposed Features:

* **Localized Alerts and Logs:** Allow administrators to view alerts and logs in their preferred language.
* **Region-Specific Rule Sets:** Create rules tailored to regional traffic patterns for more accurate detection.

###### Impact:

* Broader adoption across global markets.
* Increased usability for non-English-speaking users.

###### 6.4 Scaling for High Traffic Volumes

**Description:**

As organizations grow, their networks generate more data. The system needs to scale effectively to handle high volumes of traffic during potential DoS/DDoS attacks.

###### Proposed Features:

* **Efficient Packet Handling:** Optimize the packet capture and detection logic for high-speed traffic processing.
* **Data Partitioning:** Use a modular design to divide traffic monitoring across multiple instances.

###### Impact:

* Seamless performance under heavy network loads.
* The ability to manage enterprise-scale deployments.

###### 6.5. Advanced Alert Customization Description:

Customizable alerting mechanisms will help administrators focus on actionable events and minimize noise.

###### Proposed Features:

* **User-Defined Thresholds:** Allow users to set specific thresholds for alerts based on traffic volume, packet type, or protocol.
* **Priority-Based Alerts:** Introduce alert severity levels to help prioritize critical responses.

###### Impact:

* Enhanced user control over the detection process.
* Reduced alert fatigue and better focus on critical issues.

###### 6.6. Predictive Maintenance and Resource Optimization Description:

The system can evolve to include predictive capabilities, such as forecasting resource bottlenecks or potential failures.

###### Proposed Features:

* **Traffic Trend Analysis:** Predict future traffic spikes and provide early warnings.
* **Optimization Recommendations:** Suggest changes to system configurations to handle high traffic volumes more efficiently.

###### 6.7. Integration with Security Tools Description:

Integrating the system with other security platforms will enhance its functionality and provide a comprehensive view of network threats.

###### Proposed Features:

* **SIEM Integration:** Export detected anomalies to tools like Splunk or IBM QRadar for unified threat management.
* **Threat Intelligence Feeds:** Use external databases to enhance detection accuracy with real-time threat data.

###### Impact:

* Improved threat management through centralized monitoring.
* Synergy with existing enterprise security frameworks.

###### 6.8. Real-Time Collaboration Features Description:

Adding collaboration features will streamline incident response by enabling team-based workflows.

###### Proposed Features:

* **Shared Dashboards:** Allow teams to collaboratively annotate and analyze detected threats.
* **Activity Logs:** Track administrator actions during threat investigations.

###### Impact:

* Faster resolution of incidents through collaborative decision-making.
* Improved accountability and coordination among security teams.

###### 6.9. Cloud-Agnostic Deployment Description:

Ensuring the system can operate on diverse infrastructures, including cloud platforms and on-premises setups, will enhance deployment flexibility.

###### Proposed Features:

* **Containerized Deployment:** Use Docker or Kubernetes to allow easy deployment across different environments.
* **Hybrid Infrastructure Support:** Seamlessly operate across both cloud and on-premises systems.

###### Impact:

* Greater compatibility with various organizational infrastructures.
* Simplified deployment for users with different technical requirements.

###### 6.10. Enhanced Visualizations and Reporting Description:

Improving the reporting and visualization capabilities will make it easier for users to understand attack trends and system health.

###### Proposed Features:

* **Interactive Graphs:** Show attack frequency, severity, and source distribution in real time.
* **Customizable Reports:** Generate detailed or summary reports for different audiences, such as technical teams or management.

###### Impact:

* Improved decision-making with clearer insights.
* Better communication of system performance and detected threats to stakeholders.

###### 6.11. IoT and Edge Device Monitoring Description:

Expanding the system’s capabilities to monitor IoT and edge devices will address emerging attack vectors targeting these devices.

###### Proposed Features:

* **Lightweight Packet Analysis Agents:** Deploy agents on IoT devices for efficient traffic monitoring.
* **Customized Detection Rules:** Tailor detection rules to specific IoT protocols and behaviors.

###### Impact:

* Enhanced protection for IoT-enabled networks.

Improved security for industries adopting smart devices, such as healthcare and manufactur

The **DoS/DDoS Attack Detection System** presents a streamlined and effective approach to combating modern network security threats. By leveraging Python-based scripts for traffic analysis and detection, the system provides real-time monitoring, anomaly detection, and mitigation capabilities. This solution addresses the growing need for accurate and efficient protection against DoS/DDoS attacks, ensuring network reliability and security in an increasingly digital world.

**Attack Detection Logic:**

The system uses Python scripts (final.py and detector2.py) to simulate and detect various attack

types, such as SYN floods, UDP floods, ICMP floods, and HTTP floods. This targeted approach allows real-time identification of malicious traffic.

**Threshold-Based Anomaly Detection:**

With dynamically adjustable thresholds, the system detects traffic anomalies like sudden spikes or unusual packet patterns, minimizing false positives and ensuring accurate identification of threats.

###### 6.12. Efficient Data Management

**Lightweight Logging Mechanism:**

Logs generated by the detection system are optimized for size and clarity, focusing on capturing critical details such as attack types, source IPs, and timestamps.

**Real-Time Data Analysis:**

Packet data is analyzed as it is captured, allowing for immediate flagging of suspicious traffic and reducing delays in detection.

###### 6.13. User-Centric Design

**Simple and Intuitive Interface:**

tools like Matplotlib. This ensures that both novice and expert users can navigate and understand system alerts effortlessly.

**Actionable Alerts:**

Generated alerts include detailed descriptions of the detected threat, severity levels, and recommendations for mitigation, enabling quick and informed decision-making.

###### 6.14. Scalable and Secure Architecture

**Optimized for Local Deployment:**

The system is designed for deployment on local machines or small-scale servers, ensuring ease of use and quick setup without relying on complex cloud platforms.

**Security Measures:**

* + Logs are securely stored and access is restricted to authorized users.
  + The system provides basic encryption options for sensitive data, ensuring privacy and security in production environments.

###### 6.15. Rigorous Testing and Validation

**Testing Strategies:**

* + **Unit Testing:** Validated individual components, such as packet capture and anomaly detection, to ensure functionality.
  + **Integration Testing:** Verified seamless interaction between the attack simulation and detection scripts.
  + **User Acceptance Testing:** Confirmed the system’s usability and effectiveness with feedback from IT administrators and network engineers.

**Performance Validation:**

* + The system demonstrated consistent performance under high traffic loads, processing packets and generating alerts in real time without significant delays.
  + Fine-tuned detection thresholds to reduce false positives, ensuring accurate identification of DoS/DDoS attacks.

###### 6.16. Improved Operational Efficiency

**Proactive Maintenance:**

Analyzes traffic trends to predict potential performance issues, allowing administrators to address bottlenecks before they impact network availability.

**Streamlined Processes:**

Automation of packet analysis and logging reduces manual workload, enabling security teams to focus on strategic tasks.

###### 6.17. User-Friendly Experience

**Intuitive Dashboards:**

Real-time visualizations of traffic metrics and attack trends allow users to understand network health and respond to anomalies effortlessly.

###### 6.18. Adaptability and Scalability

**Dynamic Scaling:**

The modular design allows for easy deployment across multiple machines or network segments, ensuring the system adapts to growing organizational needs.

**Future-Ready Architecture:**

The system can be expanded with additional features, such as IoT monitoring or advanced alerting mechanisms, to address emerging security challenges.

###### 6.19. Global Accessibility

**Multi-Language Support (Future Feature):**

Enabling localization will make the system accessible to non-English-speaking users, broadening its appeal to global organizations.

**Compliance with Best Practices:**

The system’s secure design aligns with international cybersecurity standards, making it suitable for deployment in diverse environments.

###### 6.20. Cost-Effectiveness

**Automated Detection and Logging:**

By automating key processes, the system reduces the need for constant manual monitoring, saving time and resources.

**Reduced Downtime:**

Proactive threat detection and mitigation help prevent costly disruptions to business operations.

# Chapter – 7

**REFERENCE**

[1]. **Kar Suvra, D., & et al. (2025).** An Efficient Real Time DDoS Detection Model Using Machine Learning Algorithms. arXiv Preprint, Jan.–Feb. 2025.<https://arxiv.org/abs/2501.14311> [sciencedirect.com+11arxiv.org+11ijisae.org+11](https://arxiv.org/html/2501.14311v1?utm_source=chatgpt.com)

[2]. **Ismail, et al. (2022).** A machine learning–based classification and prediction technique for DDoS attacks. IEEE Access, 10, 21443–21454. [ijisae.org+3researchgate.net+3nature.com+3](https://www.researchgate.net/publication/381950906_Detecting_DDoS_Attacks_using_Machine_Learning_Survey?utm_source=chatgpt.com)

[3]. **Akgun, D., Hizal, S., & Cavusoglu, U. (2022).** A new DDoS attacks intrusion detection model based on deep learning for cybersecurity. Computer Security, 118, 102748. [nature.com](https://www.nature.com/articles/s41598-024-84879-y?utm_source=chatgpt.com)

[4]. **Survey and classification of DoS and DDoS attack detection and mitigation using machine learning** (2023). Journal of Network and Computer Applications (Survey). [isroset.org+11sciencedirect.com+11scitepress.org+11](https://www.sciencedirect.com/science/article/abs/pii/S254266052300344X?utm_source=chatgpt.com)

[5]. **Lee, S. H., Shiue, Y. L., Cheng, C. H., Li, Y. H., & Huang, Y. F. (2022).** Detection and prevention of DDoS attacks on the IoT. Applied Sciences, 12(23), 12407. [growingscience.com](https://growingscience.com/beta/ijds/7162-detecting-ddos-attacks-using-machine-learning-algorithms-and-feature-selection-methods.html?utm_source=chatgpt.com)

[6]. **Liu, X., & Du, Y. (2023).** Effective feature selection for IoT botnet attack detection using a genetic algorithm. Electronics, 12(5), 1260. [growingscience.com](https://growingscience.com/beta/ijds/7162-detecting-ddos-attacks-using-machine-learning-algorithms-and-feature-selection-methods.html?utm_source=chatgpt.com)

[7]. **Marvi, M., Arfeen, A., & Uddin, R. (2021).** A generalized machine learning–based model for detection of DDoS attacks. International Journal of Network Management, 31(6), e2152. [ijisae.org+13growingscience.com+13sciencedirect.com+13](https://growingscience.com/beta/ijds/7162-detecting-ddos-attacks-using-machine-learning-algorithms-and-feature-selection-methods.html?utm_source=chatgpt.com)

[8]. **Seifousadati, A., Ghasemshirazi, S., & Fathian, M. (2021).** A machine learning approach for DDoS detection on IoT devices. arXiv preprint arXiv:2110.14911. [growingscience.com](https://growingscience.com/beta/ijds/7162-detecting-ddos-attacks-using-machine-learning-algorithms-and-feature-selection-methods.html?utm_source=chatgpt.com)

[9]. **Ullah, S., Mahmood, Z., Ali, N., Ahmad, T., & Buriro, A. (2023).** Machine learning‑based dynamic attribute selection technique for DDoS attack classification in IoT networks. Computers, 12(6), 115. [growingscience.com](https://growingscience.com/beta/ijds/7162-detecting-ddos-attacks-using-machine-learning-algorithms-and-feature-selection-methods.html?utm_source=chatgpt.com)

[10]. **Zhao, J., Xu, M., Chen, Y., & Xu, G. (2023).** A DNN architecture generation method for DDoS detection via genetic algorithm. Future Internet, 15(4), 122. [growingscience.com](https://growingscience.com/beta/ijds/7162-detecting-ddos-attacks-using-machine-learning-algorithms-and-feature-selection-methods.html?utm_source=chatgpt.com)

[11]. **Detecting DoS attacks using machine learning algorithms** (2022). Journal of Big Data. [journalofbigdata.springeropen.com](https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00616-0?utm_source=chatgpt.com)

[12]. **Virupakshar, et al. (2020).** DNN-based DoS/DDoS detection for cloud. (Within SciTePress evaluation paper) [scitepress.org+1ijisae.org+1](https://www.scitepress.org/Papers/2023/116057/116057.pdf?utm_source=chatgpt.com)

[13]. **Farhat, S., Abdelkader, M., Meddeb‑Makhlouf, A., & Zarai, F. (2023).** Evaluation of DoS/DDoS attack detection with ML techniques on the CIC‑IDS2017 dataset. ICISSP 2023, 287–295. [arxiv.org+5scitepress.org+5researchgate.net+5](https://www.scitepress.org/Papers/2023/116057/116057.pdf?utm_source=chatgpt.com)

[14]. **Perez‑Diaz, V., Choo, K., & Zhu, Y. (2020).** A flexible SDN-based architecture for identifying and mitigating low-rate DDoS attacks using machine learning. IEEE Access, 8, 155859–155872. [ijisae.org](https://www.ijisae.org/index.php/IJISAE/article/view/3919?utm_source=chatgpt.com)

[15]. **Phan, Park (2019).** Efficient Distributed Denial-of-Service Attack Defense in SDN-Based Cloud. IEEE Access, 7, 18701–18714. [ijisae.org](https://www.ijisae.org/index.php/IJISAE/article/view/3919?utm_source=chatgpt.com)

[16]. **Dong, S., & Sarem (2020).** DDoS attack detection based on improved KNN within SDN networks. IEEE Access, 8, 5039–5048.

[17]. **Sambangi, Gondi (2020).** A machine learning approach for DDoS attack detection using multiple linear regression. Proceedings, MDPI. [ijisae.org+1researchgate.net+1](https://www.ijisae.org/index.php/IJISAE/article/view/3919?utm_source=chatgpt.com)

[18]. **Najar, & Naik (2021).** Applying various ML techniques for DDoS detection using UNSW‑NB15. Conference proceedings.

[19]. **Slyamkhanov, A. et al. (2023, Oct).** Usage of machine learning in DDoS attack detection. WINCOM 2023, pp. 1–6.

[20]. **Borah, R., Sarmah, S., Choudhury, N., Mahanta, H., & Chodhury, A. (2023).** DDoS attack detection using machine learning techniques. Indian Journal of Science and Technology, 16, 76–82.

[21]. **Bandi, A., Sherpa, L., & Allu, S. M. (2022).** ML algorithms for DDoS attack detection in cybersecurity. In Modern Approaches in Machine Learning & Cognitive Science, pp. 269–281.

[22]. **Amrish, R., Bavapriyan, K., Gopinaath, V., Jawahar, A., & Kumar, C. V. (2022).** DDoS detection using machine learning techniques. Journal of IoT in Social, Mobile, Analytics, and Cloud, 4(1), 24–32.

[23]. **Cil, A. E., Yildiz, K., & Buldu, A. (2021).** Detection of DDoS attacks with feed-forward deep neural network model. Expert Systems with Applications, 169, 114520.

[24]. **Chavan, N., Kukreja, M., Jagwani, G., Nishad, N., & Deb, N. D. (2022).** DDoS attack detection and botnet prevention using ML. ICACCS 2022.

[25]. **Nazarudeen, F., & Sundar, S. (2022).** Efficient DDoS attack detection using machine learning techniques. IPRECON 2022. [growingscience.com+12researchgate.net+12arxiv.org+12](https://www.researchgate.net/publication/381950906_Detecting_DDoS_Attacks_using_Machine_Learning_Survey?utm_source=chatgpt.com)

[26]. **Kavitha, M., et al. (2022).** Machine learning techniques for detecting DDoS attacks in SDN. ICACRS 2022.

[27]. **Amaad, H., & Mughal, H. (2023).** Experimenting ensemble machine learning for DDoS classification. ICACS 2023, pp. 1–7.

[28]. **Fardusy, T., Afrin, S., Sraboni, I. J., & Dey, U. K. (2023).** An autoencoder‑based approach for DDoS attack detection using semi‑supervised learning. NCIM 2023, pp. 1–7.

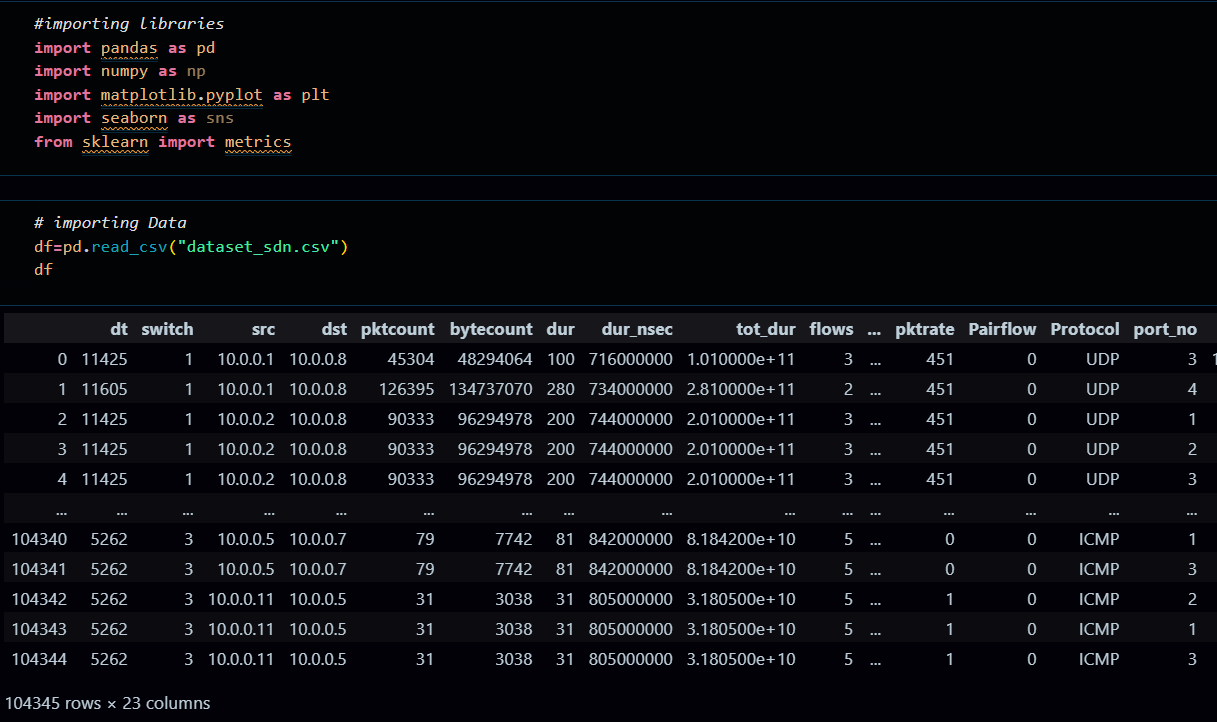
[29]. **Liu, Z., et al. (2023).** A DDoS detection method based on feature engineering and ML in SDNs. Sensors, 23, [Article].

[30]. **Ramzan, M., Shoaib, M., Altaf, A., & Ashraf, I. (2023).** Distributed denial of service attack detection in network traffic using deep learning algorithm. Sensors (Basel), 23.

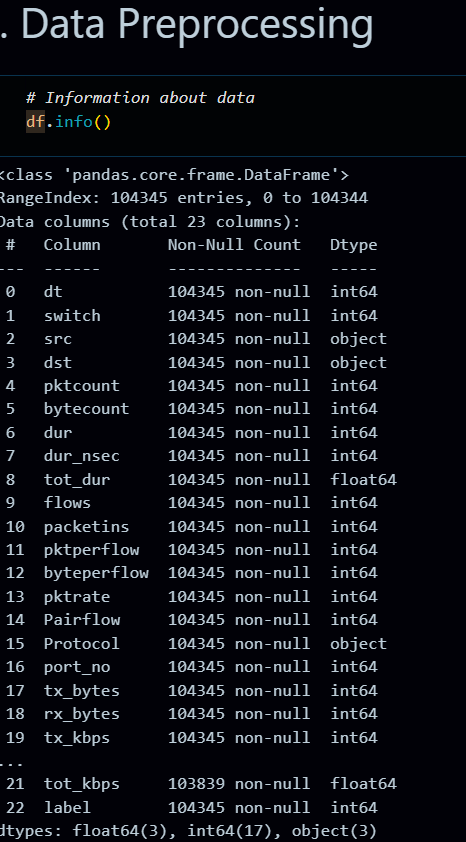
# Chapter -8

**APPENDICES**

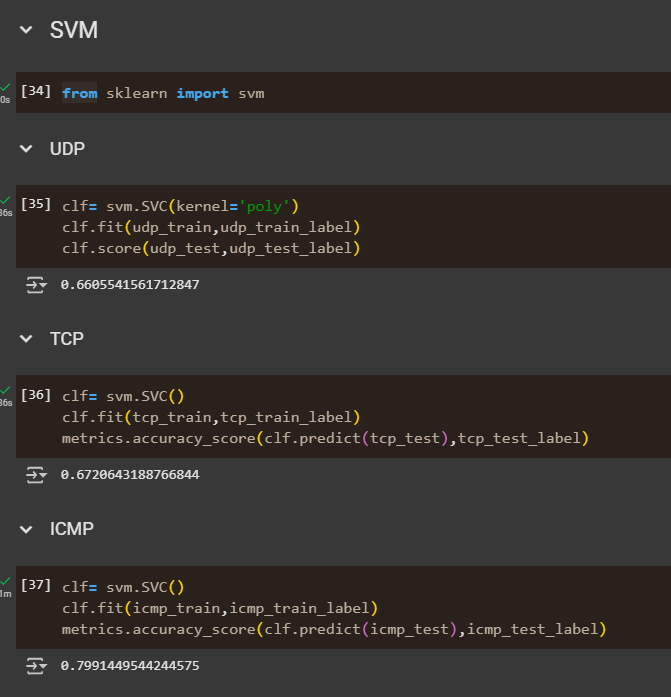
## 8.1 Code Snippets



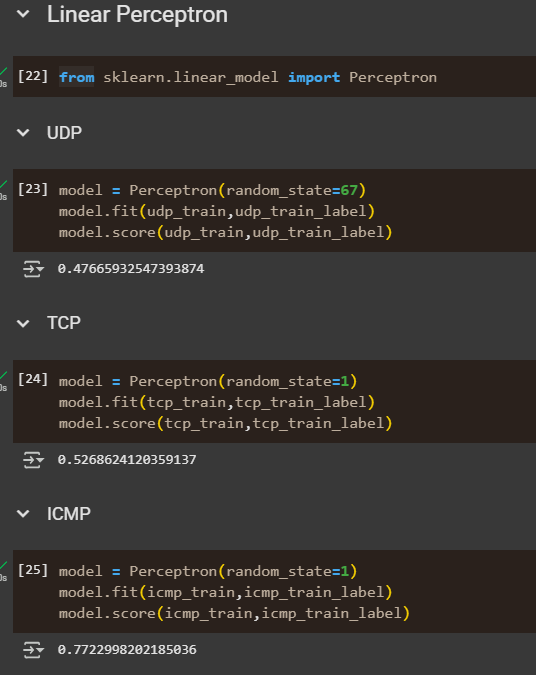
**Fig. 8.1.1 : Import libraries and Read Data**



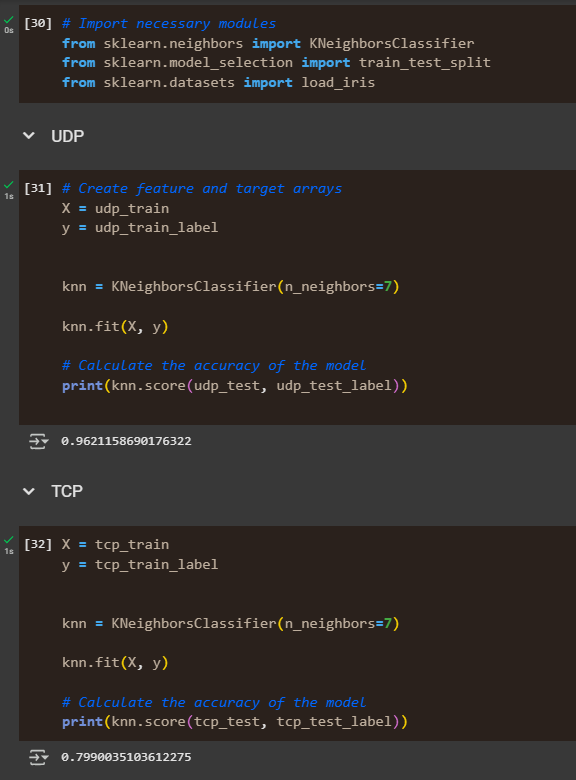
**Fig. 8.1.2 : Data Preprocessing**

****

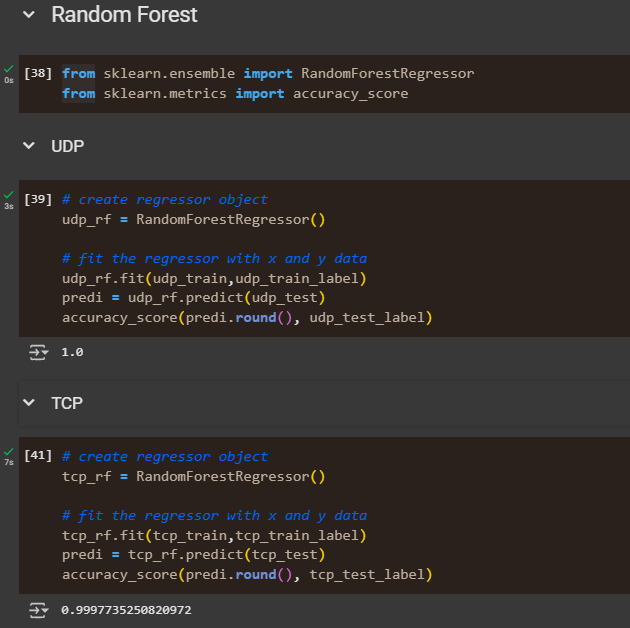
**Fig 8.1.4: SVM Model train**

****

**Fig 8.1.5: Linear Perceptron Model train**



**Fig 8.1.5: KNN Model Train**



**Fig 8.1.5: Random Forest Model Train**