

MACHINE LEARNING TECHNIQUES TO DETECT NETWORK INTRUSION

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

P. SHASHANK (Reg.No - 40111191)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Category-1 University by UGC

**Accredited with Grade “A++” by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119**

APRIL - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **P. Shashank** who carried out the Project entitled "**MACHINE LEARNING TECHNIQUES TO DETECT NETWORK INTRUSION**" under my supervision from November 2023 to April 2024.


8/4/24
Internal Guide

Ms. K. ANITA DAVAMANI, M.E., (Ph.D.)

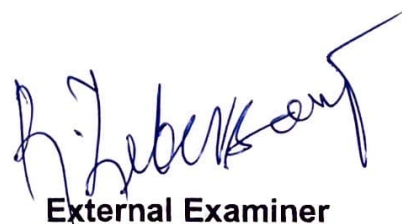
HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Head of the Department, Rajiv Gandhi Salai,
Chennai - 600 119.
Dr. L. LAKSHMANAN M.E., Ph.D.

Submitted for Viva voce Examination held on

12/4/24



Internal Examiner


External Examiner

DECLARATION

I, **P. SHASHANK** (Reg.No- 40111191), hereby declare that the Project Report entitled "**MACHINE LEARNING TECHNIQUES TO DETECT NETWORK INTRUSION**" done by me under the guidance of **Mrs. K. Anita Davamani, M.E., (Ph.D.)**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 12/4/2024

PLACE: Chennai

P. Shashank

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mrs. K. Anita Davamani, M.E., (Ph.D.)**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

In response to the escalating security threats accompanying the evolution of wireless communication, the deployment of an Internet site Intrusion Detection System (IDS) has become imperative for organizations. This system serves as a crucial defense mechanism, actively identifying and thwarting potential attacks from both internal and external sources. The continuous evolution of cyber threats necessitates a proactive approach, and earlier efforts have seen the integration of various Machine Learning (ML) methods into IDS. To enhance detection outcomes and improve accuracy in distinguishing between normal and malicious activities, multiple ML techniques, including SVM, Naive Bayes, and Decision Tree, have been applied. The obtained results underscore the effectiveness of the proposed approach, showcasing superior accuracy compared to these conventional methods. Specifically, the utilization of Principal Component Analysis (PCA) in conjunction with the Random Forest algorithm emerges as a pivotal strategy in achieving robust intrusion detection capabilities. The primary objective of the system revolves around intrusion detection, and the amalgamation of PCA and the Random Forest algorithm stands out for its ability to provide nuanced insights into network behavior. This approach ensures heightened accuracy in classifying potential threats, fortifying the organization's cybersecurity posture against a dynamic landscape of emerging cyber risks. In essence, the integration of advanced ML techniques within the IDS framework marks a significant stride toward achieving proactive and efficient cybersecurity in the face of evolving digital threats. Furthermore, ongoing research and development in the field of ML promise to unlock new capabilities and refine existing algorithms, further enhancing the effectiveness of intrusion detection systems. By embracing innovation and leveraging the power of ML-driven technologies, organizations can establish robust defenses that adapt and evolve alongside the ever-changing threat landscape. In the pursuit of proactive and efficient cybersecurity, the integration of advanced ML techniques within IDS frameworks is poised to play a pivotal role in safeguarding critical assets and preserving the integrity of network infrastructures.

TABLE OF CONTENTS

Chapter No	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	xi
	LIST OF TABLES	x
1	INTRODUCTION	1
	1.1 Intrusion Detection System	1
	1.1.1 Traditional Intrusion Detection Systems (IDS)	1
	1.1.2 IDS with Machine Learning	2
	1.1.3 Flask Implementation	2
	1.2 Problem Statement	3
2	LITERATURE SURVEY	4
	2.1 Description of Papers Used	4
	2.2 Inferences from Literature Survey	10
	2.3 Open Problems in Existing System	12
3	REQUIREMENT ANALYSIS	14
	3.1 Feasibility Studies of the Project	14
	3.1.1 Technical Feasibility	14
	3.1.2 Economic Feasibility	14
	3.1.3 Operational Feasibility	14
	3.1.4 Scheduling Feasibility	15
	3.2 Software Requirements Specification Document	15
	3.2.1 Purpose	15
	3.2.2 Scope	15
	3.3 System Overview	16

	3.3.1	Functional Requirements	16
	3.3.2	Non-Functional Requirements	17
4		DESCRIPTION OF PROPOSED SYSTEM	18
	4.1	Architecture	18
	4.2	Description of Software for Implementation and Testing Plan of the Proposed Model/System	19
	4.3	4.3.1 Dataset	21
		4.3.2 Data Preprocessing	22
		4.3.3 Models	22
		4.3.3.1 Ensemble Model	22
		4.3.3.2 Support Vector Machine (SVM)	23
		4.3.3.3 Gradient Boosting Classifier	23
		4.3.3.4 Naive Bayes	23
		4.3.3.5 Random Forest	24
5		IMPLEMENTATION DETAILS	25
	5.1	Development and Deployment Setup	25
	5.2	Workflow	26
6		RESULTS AND DISCUSSION	29
	6.1	Presentation of Findings	29
	6.2	Discussion on Data Capture	31
7		CONCLUSIONS	33
	7.1	Conclusion	33
	7.2	Recommendations for Future Research	34
		REFERENCES	35
		APPENDIX	37
	A	SOURCE CODE	37
	B	SCREENSHOTS	50
	C	RESEARCH PAPER	54
	D	CERTIFICATE	59

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO.
4.1	Architecture Diagram	18
4.2	Workflow Diagram	19
5.1	Dataflow Diagram	26
5.2	Use Case Diagram	28
6.1	Class Distribution	29
6.2	Feature Importance	29
6.3	Accuracy Graph	30
6.4	Loss Graph	30
6.5	Live Network Capture Through Wireshark	32
B.1	Importing Required Libraries	50
B.2	Pre-processing the Data	50
B.3	Outliers	51
B.4	Flask Application Run	51
B.5	Application Portal	52
B.6	Login Page	52
B.7	Upload Page	53
B.8	ML Prediction Page	53

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
SVM	Support Vector Machine
DDoS	Distributed Denial of Service
MLP	Multi-Layer Perceptron
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
6.1	Results	30

CHAPTER 1

INTRODUCTION

1.1 INTRUSION DETECTION SYSTEM

An intrusion is any kind of action that compromises the integrity, confidentiality, and availability of some information or computer resource. Using the weakness or flaws in the system architecture, the intruder intrudes to circumvent the authentication or authorization process. With the tremendous growth of network-based services and secured information on networks, network security is becoming more and more important than ever before. One solution to this is the use of Network Intrusion Detection System (NIDS) that detect attacks by observing various network activities. So it is more important that such systems should be more accurate in identifying attacks, quick to train and to generate as few false positives as possible. As technology evolves, so do the tactics of potential attackers, making the role of intrusion detection systems increasingly critical in safeguarding networks and data.

1.1.1 Traditional Intrusion Detection Systems (IDS)

Traditional Intrusion Detection Systems (IDS) have been instrumental in detecting and preventing cyber threats by analyzing network traffic and system logs. These systems typically operate using signature-based detection, where predefined patterns or signatures of known attacks are compared against incoming network traffic. If a match is found, the IDS raises an alert or takes predefined actions to mitigate the threat.

One common type of traditional IDS is the network-based IDS (NIDS), which monitors network traffic in real-time and identifies suspicious patterns indicative of malicious activity. Another type is the host-based IDS (HIDS), which operates on individual hosts or endpoints, monitoring system logs, file integrity, and other host-specific attributes for signs of compromise.

While traditional IDS have proven effective in many scenarios, they have limitations. Signature-based detection methods rely on a database of known attack

signatures, making them susceptible to zero-day attacks or variations of known threats. Additionally, these systems may generate a high number of false positives, leading to alert fatigue and decreased efficacy in threat detection.

1.1.2 IDS with Machine Learning

To address the limitations of traditional IDS, researchers and practitioners have turned to machine learning (ML) techniques to enhance intrusion detection capabilities. ML-powered IDS leverage algorithms to analyze network traffic and system behavior, learning from past data to identify patterns associated with malicious activity.

One approach is anomaly detection, where ML models are trained on normal network behavior to detect deviations indicative of an attack. These anomalies could include unexpected spikes in network traffic, unusual system access patterns, or deviations from established baselines.

Another approach is behavior analysis, where ML models learn to identify patterns of behavior associated with specific types of attacks. By analyzing features such as packet headers, payload content, and user behavior, these models can detect suspicious activity that may evade signature-based detection methods. ML-powered IDS offer several advantages over traditional approaches. They can adapt to evolving threats by continuously learning from new data, making them more resilient to zero-day attacks. Additionally, ML models can reduce false positives by focusing on behavior-based analysis rather than relying solely on predefined signatures. In conclusion, while traditional IDS have played a crucial role in network security, the emergence of ML-powered IDS represents a significant advancement in intrusion detection capabilities. By leveraging the power of machine learning, these systems can provide more accurate and efficient threat detection, helping organizations stay ahead of increasingly sophisticated cyber threats.

1.1.3 Flask Implementation

In implementing a Flask application integrated with machine learning

algorithms for intrusion detection upon uploading a desired dataset, several steps are involved. First, the Flask application is structured to handle user interactions, file uploads, and result presentation. Upon receiving a dataset upload, the application preprocesses the data, which may involve cleaning, formatting, and feature extraction, to prepare it for analysis. Subsequently, the preprocessed dataset is fed into the deployed machine learning algorithms, such as Random Forests, Support Vector Machines, or deep learning models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs). These algorithms are trained on labeled datasets to recognize patterns indicative of network intrusions. Once the machine learning models have processed the data, the application interprets the results, which may include identifying potential intrusions, classifying network activity, or providing anomaly detection alerts. Finally, the Flask application presents these results to the user, either through a web interface displaying visualizations or summaries, or through an API endpoint returning structured data. This seamless integration of Flask with machine learning algorithms enables efficient analysis and detection of network intrusions based on datasets.

1.1 PROBLEM STATEMENT

Our intrusion detection method includes three phases. We first use the feature selection model to preprocess the collaborative edge network data, and we select the required features for intrusion detection. Second, we designed an intrusion detection algorithm aiming at a single attack based on GAN. After that, by combining several intrusion detection models aiming at a single attack, we design an intrusion detection algorithm aiming at multiple attacks based on GAN. An intrusion detection device (IDS) helps defend networks with the aid of detecting malicious intrusions. Hence, IDS has become an important issue of computer networks. Security is paramount in all initiatives to save you any loss. An important feature of an IDS is to provide records approximately uncommon pastime and block echo/tracking and/or suspicious connectivity to inform network administrators. In addition, an IDS can also distinguish between internally generated assaults and externally generated assaults.

CHAPTER 2

LITERATURE SURVEY

2.1 DESCRIPTION OF PAPERS USED

J. Ye et al. [12] utilized the Support Vector Machine (SVM) algorithm to detect Distributed Denial of Service (DDoS) attacks in software-defined networks (SDN). This research made significant strides in enhancing the security of SDN environments. By employing machine learning techniques for threat detection, they achieved a high degree of accuracy in identifying and mitigating DDoS attacks. This work is essential in safeguarding the integrity of SDN systems, which are susceptible to disruptive DDoS attacks.

S. Almutairi et al. [22] conducted research on a hybrid botnet detection approach that combines both host and network analysis. Their method significantly improved the accuracy of identifying botnets, which are malicious networks of compromised devices. By adopting a comprehensive approach that considers both host and network-level analysis, their work provides a more holistic and robust solution for detecting these malicious botnets. This approach has substantial implications for enhancing network security.

H. Alkahtani and T. H. H. Aldhyani [10] proposed a Convolutional Neural Network-Long Short-Term Memory (CNN-LSTM) model for detecting botnet attacks in IoT applications. Focusing on deep learning, their research explored the integration of CNNs and LSTMs, which are essential for processing IoT data for botnet detection. This approach is critical for ensuring the security of IoT environments, where the number of connected devices is rapidly increasing, and botnet attacks are a growing concern.

D. Javeed et al. [7] developed a hybrid deep learning-driven mechanism that

incorporates Software-Defined Networking (SDN) to secure IoT communication. By combining deep learning techniques with SDN, their emphasis was on improving the security of communication in IoT systems. This research highlights the potential of merging these advanced technologies to address vulnerabilities in IoT communication, thereby bolstering overall security.

A. Dawoud et al. [2] investigated the role of deep learning and software-defined networks in securing IoT architecture. They emphasized the importance of combining these technologies to create a robust security framework for IoT. By harnessing the power of deep learning and the flexibility of SDN, this research underscores the need for innovative approaches to fortify the security of IoT ecosystems.

A. Pektaş and T. Acarman [4] proposed a botnet detection approach that leverages deep learning and network flow summaries. This research underscores the value of deep learning in intrusion detection. By summarizing network flows, their approach enhances the identification of malicious botnet activity, improving overall security in networks and IoT environments.

S. Shitharth et al. [23] introduced an innovative classification approach for network intrusion detection using perceptual pigeon galvanized optimization. Their work stands out for its creativity in tackling the problem of intrusion detection. By developing novel techniques for classifying and detecting network intrusions, they contribute to the evolving landscape of security solutions, emphasizing the need for inventive approaches.

A. Canziani et al. [1] analyzed deep neural network models for practical applications, offering insights into the real-world use of deep learning. While their research may not be directly focused on botnet detection, it contributes to the broader understanding of the practicality of deep learning in various applications, including security. This understanding is critical for developing robust and applicable

security solutions in complex, dynamic environments.

H. Hochreiter and J. Schmidhuber's [11] foundational work introduced Long Short-Term Memory (LSTM) networks, which are instrumental in various deep learning applications, including IoT botnet detection. LSTM networks excel in handling sequential data, which is prevalent in IoT traffic analysis. Their pioneering research laid the foundation for using LSTM networks in many cybersecurity applications, significantly advancing the field's capabilities.

V. Rey et al.[25] investigated federated learning for malware detection in IoT devices. Their research focused on distributed learning approaches, which are especially relevant in scenarios where data privacy and security are critical, such as in IoT. Federated learning allows devices to collaboratively train models without sharing sensitive data, making it an important approach for enhancing IoT security. Their work emphasizes the significance of privacy-preserving techniques in securing IoT ecosystems.

T. V. Khoa et al.[24] presented a collaborative learning model for cyberattack detection systems in the IoT industry. They emphasize the importance of collaboration in security, especially in the context of IoT, where diverse devices and networks are involved. Collaborative learning can lead to more robust threat detection by combining the knowledge and capabilities of multiple entities.

G. De La Torre Parra et al.[8]explored distributed deep learning for detecting IoT attacks. Their work showcases the potential of distributed models in enhancing security. Distributed deep learning enables the processing of large datasets and complex models, which is beneficial for IoT security. This research underlines the significance of scalable and efficient approaches in safeguarding IoT environments.

H. Alazzam et al. [9] focused on supervised detection of IoT botnet attacks, highlighting the importance of supervised learning in IoT security. By training models

on labeled data, supervised learning can effectively identify malicious activities in IoT. This research contributes to building more robust threat detection systems for the IoT industry, where the security landscape is continually evolving.

Jafar Abo Nada and Mohammad Rasmi Al-Mosa [14] introduced a Wireless Intrusion Detection Prevention and Attack System (WIDPAS) at the 2018 International Arab Conference on Information Technology (ACIT). This system addresses the need for enhanced security solutions in wireless networks by focusing on monitoring, analysis, and defense tasks.

Kinam Park, Youngrok Song, and Yun-Gyung Cheong [16] investigated the application of the Random Forest algorithm for classifying attack types in intrusion detection systems using datasets from the Kyoto 2006+ dataset. Their study focused on evaluating the performance of detecting various types of attacks, including IDS, Malware, and Shellcode, with implications for enhancing network security

In their study presented at the International Joint Conference on Neural Networks, Bernard, Heutte, and Adam [6] investigated the selection of decision trees within Random Forests. Their research sheds light on the optimization of decision tree selection processes, contributing to advancements in the effectiveness and efficiency of Random Forest algorithms for various applications.

A. Tesfahun and D. Lalitha Bhaskari [5] utilized the Random Forests classifier with Synthetic Minority Oversampling Technique (SMOTE) and feature reduction methods to address class imbalance in intrusion detection datasets. Their research demonstrated improved performance in designing efficient and effective Intrusion Detection Systems (IDS) for security.

Le, T.-T.-H., Kang, H., & Kim, H. (2019) [17] presented a novel method to enhance the accuracy of Intrusion Detection Systems (IDS) using Gated Recurrent Unit (GRU) models. Their approach involved embedding PCA-Scale, with options

including PCA-Standardized and PCA-MinMax, into the GRU layer to improve detection accuracy. Experimental results on real-world datasets, such as KDD Cup 99 and NSL-KDD, demonstrated significant performance improvements in GRU models trained with the PCA-Scaled method.

Anish Halimaa A and Dr. K. Sundarakantham [3] presented a study on "Machine Learning Based Intrusion Detection System" at the Third International Conference on Trends in Electronics and Informatics (ICOEI 2019). Their research likely explores the application of machine learning techniques for enhancing the accuracy and effectiveness of intrusion detection systems in safeguarding networks against cyber threats.

Mengmeng Ge, Xiping Fu, Naeem Syed, Zubair Baig, Gideon Teo, and Antonio Robles-Kelly [18] introduced a novel intrusion detection scheme for IoT networks at the 2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC). Their approach utilizes deep learning concepts to classify traffic flow and develops a feed-forward neural networks model for binary and multi-class classification, aiming to enhance the security of IoT networks against various adversarial attacks.

R. Patgiri, U. Varshney, T. Akutota, and R. Kunde [20] investigated an Intrusion Detection System (IDS) utilizing machine learning algorithms, specifically Random Forest and Support Vector Machine (SVM). Their study focused on feature selection using Recursive Feature Elimination (RFE) to improve speed and accuracy in large datasets, with NSLKDD utilized for evaluation. The comparison of model performance before and after feature selection was demonstrated, along with the presentation of confusion matrices.

Rohit Kumar Singh Gautam and Er. Amit Doegar [21] presented an ensemble approach. Their research focused on optimizing IDS efficiency through feature selection methods, including Entropy-based analysis and Information gain, to select

the most relevant features from a set of 41. The study conducted three sets of experiments, evaluating the performance of Naive Bayes, Adaptive Boost, and PART algorithms in different scenarios.

Kazi Abu Taher, Billal Mohammed Yasin Jisan, and Md. Mahbubur Rahma [15] presented a study. Their research likely explores the application of supervised machine learning techniques with feature selection methods to enhance the accuracy and effectiveness of network intrusion detection systems.

In their paper L. Haripriya and M.A. Jabbar [13] provided a comprehensive review of the role of machine learning in Intrusion Detection Systems (IDS). Their work likely highlights the significance of machine learning techniques in enhancing the capabilities and effectiveness of IDS for safeguarding networks against cyber threats.

Nimmy Krishnan and A. Salim [19] presented a study on "Machine Learning-Based Intrusion Detection for Virtualized Infrastructures" at the 2018 International CET Conference on Control, Communication, and Computing (IC4). Their research likely investigates the application of machine learning techniques for detecting and mitigating intrusions within virtualized infrastructures, contributing to enhanced security measures in virtualized environments.

The literature review encompasses several key aspects. It begins by highlighting the importance of detecting attacks in various networking paradigms. Subsequently, it delves into research that explores the application of machine learning (ML) and deep learning (DL) algorithms for the detection. In particular, it focuses on studies dedicated to the detection within the Internet of Things (IoT) ecosystem, underlining the unique challenges and solutions in this domain. Additionally, it extends its scope to encompass Supervisory Control and Data Acquisition (SCADA) systems, emphasizing the relevance of detection in critical infrastructure. The review also addresses the emergence of hybrid approaches,

which combine diverse algorithms to enhance the efficacy of detection. Furthermore, it discusses the application of federated learning for enhancing IoT security and collaborative learning models for cyberattack detection in the context of IoT industry 4.0. The exploration extends to the realm of distributed deep learning as an approach to detecting IoT attacks effectively. Lastly, it includes research dedicated to supervised detection of IoT attacks, highlighting the importance of supervised learning methods. The literature review culminates in a comprehensive conclusion summarizing key findings and identifying research gaps in the field of detection across different networking paradigms.

2.2 INFERENCES FROM LITERATURE SURVEY

Inferences drawn from the collection of research papers presented here offer valuable insights into the evolving field of cybersecurity. These papers collectively emphasize the growing importance of machine learning techniques in threat detection, showcasing the accuracy and efficiency of algorithms such as Support Vector Machines (SVMs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks. This highlights their potential in significantly enhancing network security, particularly in identifying and mitigating threats like Distributed Denial of Service (DDoS) attacks.

A common theme across several papers is the importance of holistic approaches to detection. The research by S. Almutairi et al. underlines the significance of considering both host and network-level analysis, providing a more comprehensive view of network behavior and improving the accuracy of botnet identification. This integrated approach reflects the need to address the complexity of modern threats with multi-faceted strategies.

The integration of deep learning techniques is another prevalent theme, as exemplified in the works of H. Alkahtani and T. H. H. Aldhyani and A. Pektaş and T. Acarman. These papers highlight the efficacy of combining Convolutional Neural

Networks (CNNs), Long Short-Term Memory (LSTM) networks, and network flow summaries to identify malicious activities. Such integration offers promising solutions to address the ever-evolving landscape of cybersecurity threats.

Software-Defined Networking (SDN) plays a pivotal role in enhancing network security, as demonstrated in D. Javeed et al.'s work. SDN's flexibility in network management, traffic analysis, and routing control makes it a valuable tool in identifying and mitigating threats. It is shown to be a technology that can adapt to the complex demands of modern network environments.

Collaborative learning and distributed models emerge as critical components in strengthening network security. Papers by T. V. Khoa et al. and G. De La Torre Parra et al. emphasize the importance of collaboration and distributed learning in the context of diverse devices and networks. Such models provide a more robust approach to threat detection, particularly in interconnected systems where shared knowledge is invaluable.

Privacy-preserving techniques are underscored by V. Rey et al.'s investigation into federated learning for IoT device security. In IoT environments where data privacy and security are paramount, federated learning allows devices to collaborate without compromising sensitive information, emphasizing the importance of privacy preservation.

S. Shitharth et al.'s innovative classification approach stands out as a creative and unconventional method for network intrusion detection. Their work highlights the need for inventive problem-solving in the field of intrusion detection, demonstrating the value of out-of-the-box thinking.

Finally, A. Canziani et al.'s research, although not directly related to botnet detection, emphasizes the importance of practicality in deep learning applications.

exploration of deep neural network models for practical use helps bridge the gap between theoretical advancements and real-world security solutions. This approach contributes to the development of pragmatic security solutions for the dynamic and interconnected systems of today's world.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

1. **Scalability:** One significant challenge in the existing system is its ability to scale. As the number of IoT devices in networks continues to grow, the current detection mechanisms may not scale efficiently to handle the increasing data and potential threats. Finding methods to make the system more scalable without compromising accuracy is an open problem.
2. **Real-time Detection:** While some existing systems provide adequate detection, there is room for improvement in achieving real-time or near-real-time detection. This is particularly crucial for mitigating threats quickly in fast-paced environments such as critical infrastructure and healthcare.
3. **Adaptability:** Many current systems are static and lack adaptability to evolving threats. The ability to adapt to new attack strategies and malware is a continuous challenge. Developing mechanisms that can automatically update and adapt their detection strategies is essential.
4. **False Positives Reduction:** Existing systems may generate a substantial number of false positives, which can overwhelm security teams. Reducing false positives without missing actual threats remains a complex problem.
5. **Privacy Concerns:** IoT devices often handle sensitive data. Existing systems need to balance security with privacy concerns. Addressing this challenge involves developing methods that can detect threats without compromising user privacy.

6. **Resource-Constrained Devices:** IoT devices are often resource-constrained, making it challenging to deploy complex security mechanisms. Finding lightweight yet effective security solutions for these devices is an open problem.
7. **Multi-Modal Attacks:** Attackers are becoming increasingly sophisticated, employing multi-modal attacks techniques.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

The success of any project relies significantly on its feasibility. Before embarking on the journey of developing a network intrusion system and Software-Defined Networking (SDN), it's essential to conduct comprehensive feasibility studies. These studies encompass various dimensions, including technical feasibility, economic feasibility, operational feasibility, and scheduling feasibility.

3.1.1 Technical Feasibility:

The proposed project leverages advanced technologies, including Deep Learning algorithms, SDN infrastructure, and IoT device integration. The technical feasibility assessment confirms that these components are available, mature, and widely used in the industry. The tools and resources required for implementation are accessible. Furthermore, the project team possesses the necessary technical expertise to execute the project efficiently.

3.1.2 Economic Feasibility:

Economic feasibility evaluates the cost-benefit aspect of the project. The resources, both human and technical, come with associated costs. In this context, the budget requirements for hardware, software, and personnel have been examined. Moreover, it is crucial to analyze the potential benefits, such as cost savings through improved IoT security. The economic feasibility analysis demonstrates that the anticipated benefits outweigh the costs, making the project financially viable.

3.1.3 Operational Feasibility:

The operational feasibility of the project focuses on the practicality of its implementation. This project aims to enhance IoT security and reduce the risk of attacks, a critical concern in the current technology landscape. The system's

operational benefits, including the ability to detect and prevent attacks in real- time, justify its operational feasibility.

3.1.4 Scheduling Feasibility:

Scheduling feasibility ensures that the project can be completed within a reasonable timeframe. A well-defined project schedule with milestones and deliverables has been established. The project team is equipped to meet the deadlines effectively.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

The Software Requirements Specification (SRS) document serves as the foundation for the Network Intrusion software development. It outlines the specific requirements and functionalities expected from the software. This section provides a detailed overview of the SRS document.

3.2.1 Purpose

The primary purpose of the Software Requirements Specification (SRS) document is to define the features, functionalities, and constraints of the network intrusion system. It acts as a reference for both the development team and stakeholders to ensure that the software aligns with the project's goals.

3.2.2 Scope

The scope of the SRS document encompasses the following aspects given below:

- 1. System Overview:** Describes the high-level architecture of the network intrusion and its integration with front-end and network interfaces.
- 2. Functional Requirements:** Specifies the system's functional components, such as data preprocessing, machine learning models, and detection algorithms.
- 3. Non-Functional Requirements:** Details the performance, security, and

usability characteristics that the system must adhere to.

4. Interfaces: Defines the external interfaces that the system will interact with, including data sources and front-end user interfaces.

5. Data Requirements: Outlines the data sources and formats that the system will utilize for network intrusion

6. Constraints: Identifies any limitations or constraints that may impact the software development process.

7. User Requirements: Describes the specific needs and expectations of end-users who will interact with the front-end interface.

3.3 SYSTEM OVERVIEW

The network intrusion System consists of multiple modules designed to collaboratively detect and prevent network intrusion. These modules include:

- 1. Data Ingestion Module:** Responsible for collecting network traffic data from various sources, including routers.
- 2. Data Preprocessing Module:** Cleans, normalizes, and structures the incoming data to prepare it for analysis.
- 3. Machine Learning Module:** Utilizes machine learning models
- 4. Intrusion Detection Module:** Analyzes the output from the machine learning module and identifies potential network intrusion
- 5. User Interface Module:** Provides an interactive front-end where users can input network details and receive real-time feedback on the network's security status.

The system is designed to process network data efficiently and provide timely responses to detected threats. It adheres to industry best practices for security, scalability, and adaptability.

3.3.1 Functional Requirements

The functional requirements define the core capabilities of the Detection

System, including:

1. **Data Collection:** The system must collect network traffic data from IoT devices and routers.
2. **Preprocessing:** Data preprocessing is required to handle data normalization, cleaning, and structuring.
3. **Machine Learning:** The system must employ deep learning algorithms network intrusion detection, focusing on CNNs and LSTMs.
4. **Intrusion Detection:** The software should analyze the machine learning results and raise alerts if threats are detected.
5. **User Interface:** The system must provide an intuitive and responsive front-end for user interactions.

These functional requirements form the basis for the system's features, ensuring that it meets its core objectives effectively.

3.3.2 Non-Functional Requirements

The non-functional requirements encompass aspects such as:

1. **Performance:** The system must process data and provide detection results within a specified time frame.
2. **Security:** Data security and user privacy must be maintained throughout the system.
3. **Scalability:** The system should be able to handle an increasing number of devices and data sources.
4. **Usability:** The user interface should be user-friendly and accessible to both technical and non-technical users.
5. **Adaptability:** The system should be able to adapt to new attack strategies and threats.
6. **Accuracy:** The software should aim for high accuracy in attack detection.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

4.1. ARCHITECTURE

The project's architecture is structured to provide a comprehensive and user-friendly solution for network intrusion. It starts with a user-accessible frontend interface where network details can be entered. These details are processed through a series of modules, beginning with Data Collection, which gathers and preprocesses the data. The Feature Extraction Module identifies essential features for analysis.

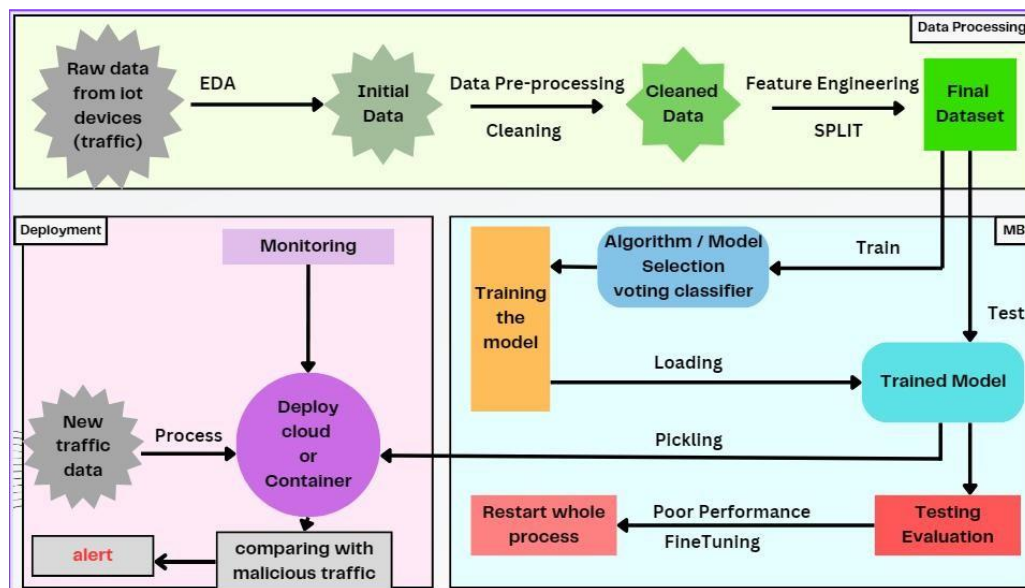


Fig: 4.1 Architecture Diagram

The core strength of the system lies in its diverse Machine Learning and Deep Learning Modules, inspired by insights from various research papers, employing tailored algorithms for detection. The Inference and Decision Module evaluates model predictions, determining malicious or benign activity. The results are displayed through the frontend interface. Supporting this is a Backend and Database for data storage, updates, and system maintenance. A Feedback Loop ensures continuous improvement by collecting user feedback. This architecture amalgamates user

interaction, dataprocessing, advanced machine learning, and deep learning to offer an adaptive solution for network attack detection, contributing significantly to security.

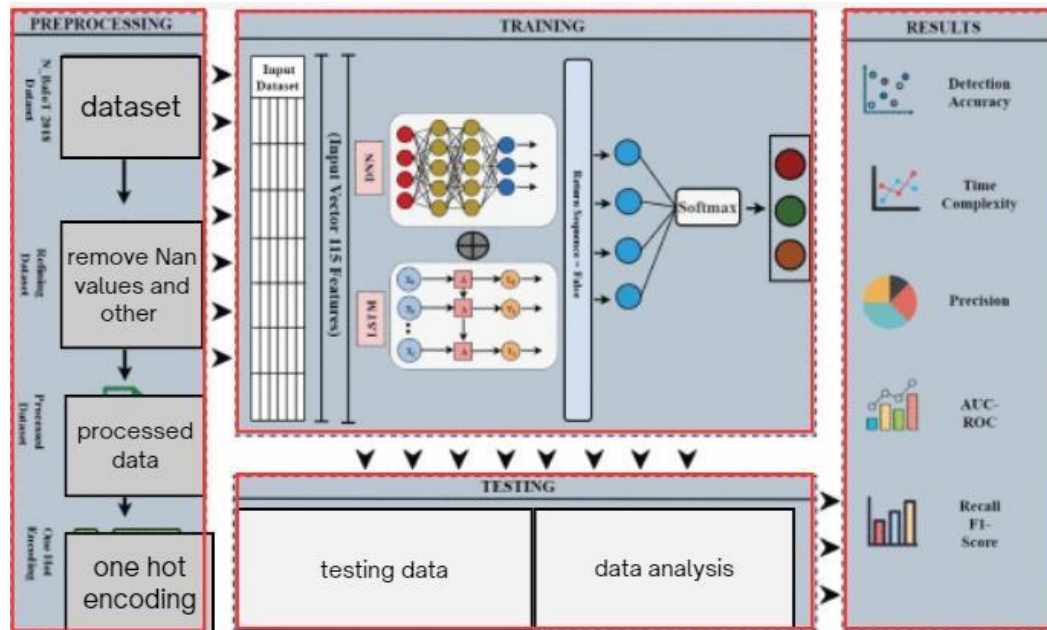


Fig: 4.2 Workflow Diagram

4.2 Description of Software for Implementation and Testing Plan of the ProposedModel/System

This section of the project report outlines the practical aspects of implementing and testing the Network Intrusion. It serves as a roadmap for bringing the proposed model into action and ensuring that it operates effectively. This section covers the following key areas:

1. **Implementation Strategy:** The implementation plan discusses how the proposed Network Intrusion will be developed and deployed. It includes details about the development environment, programming languages, and tools that will be used. Furthermore, it outlines the stages of system development, such as data collection, preprocessing, machine learning model integration, and the user

interface. The plan also takes into consideration the collaborative nature of these modules and their integration into a cohesive system.

2. **Testing Approach:** An effective testing strategy is vital to ensure that the system functions accurately and reliably. The plan outlines the testing approach, including various testing phases like unit testing, integration testing, system testing, and user acceptance testing. It also defines the criteria for success in each testing phase and the expected outcomes.
3. **Data for Testing:** For thorough testing, a dataset is required to evaluate the system's performance. This section specifies the dataset that will be used for testing, including the sources, data formats, and the size of the dataset. It discusses how real-world data will be simulated for testing and validation.
4. **Testing Scenarios:** Various testing scenarios are described, reflecting different real-world situations where the Network Intrusion should perform effectively. These scenarios cover both common and edge cases, ensuring that the system is robust and capable of detecting network attacks under various conditions.
5. **Performance Metrics:** To assess the system's effectiveness, the plan defines performance metrics and criteria. Metrics include detection accuracy, false positives, response time, and resource utilization. By setting specific benchmarks, the testing plan ensures that the system meets its goals.
6. **User Feedback and Acceptance Testing:** User acceptance testing is crucial, as it validates that the system meets the end-users' expectations. The plan includes a strategy for gathering user feedback, making necessary adjustments, and ensuring that the user interface is user-friendly.
7. **Security Testing:** Ensuring the security of the Network Intrusion is paramount. The testing plan includes security testing to identify vulnerabilities and evaluate the system's resistance to intrusion.

4.3 PROJECT MANAGEMENT PLAN

4.3.1 Dataset

In this project focused on network intrusion detection using machine learning, the dataset chosen plays a critical role in shaping the effectiveness of the model. The selected dataset is the KDD Cup 1999 dataset, renowned for its historical significance in the field of intrusion detection.

The dataset was meticulously collected by simulating a military network environment, resembling a US Air Force LAN. This simulated network was deliberately subjected to a variety of attacks, creating a realistic yet controlled environment for data generation. The raw TCP/IP dump data captures both normal network behavior and deliberate intrusion attempts.

Comprising 41 features extracted from TCP/IP connection records, the dataset provides a comprehensive mix of quantitative and qualitative attributes. Each connection, representing a sequence of TCP packets, is categorized as either "Normal" or "Anomalous," with the latter further specifying the attack type. Qualitative features include start and end times, source and target IP addresses, and protocol information. Quantitative features, numbering 38, delve into various aspects of the connection, such as duration, failed logins, and counts of inbound and outbound commands.

The qualitative features lay the groundwork for understanding basic connection attributes, while the quantitative features offer deeper insights into network traffic characteristics. This diverse set of features is essential for training the machine learning model to effectively distinguish between normal and anomalous network behavior, facilitating accurate intrusion detection.

Therefore, the KDD Cup 1999 dataset, with its simulated environment and rich feature set, serves as a foundational resource for developing a robust intrusion detection system. Its carefully crafted diversity empowers the model to learn and generalize effectively, contributing to the project's objective of enhancing network security against evolving cyber threats.

4.3.2 Data Preprocessing

The data preprocessing for the dataset involves several steps to make the data suitable for our hybrid machine learning method:

1. **Input Stage:** We start by loading the dataset, which records of traffic data from different devices. Each device's data is stored in a separate CSV file.
2. **Refining the Dataset:** In this step, we check the dataset for any missing or infinite values. These values can cause errors and slow down the learning process. To ensure data accuracy and safety, we remove any such problematic values. We also standardize the data using a technique called MinMaxScaler, which scales the data between 0 and 1.
3. **Processed Dataset:** After refining the dataset, we obtain a single CSV file that contains all the data from the various nodes. This consolidated dataset is now free from any missing or infinite values, making it more suitable for analysis.
4. **One-Hot Encoding (OHE):** To enhance the performance of our deep learning algorithm, we use a technique called One-Hot Encoding (OHE). OHE helps the algorithm work better by normalizing the data based on the three categories represented by its labels, which are 0, 1, and 2. Data preprocessing involves cleaning and standardizing the dataset to remove errors and prepare it for effective analysis using our hybrid deep learning approach.

4.3.3 Models

4.3.3.1 Ensemble Model

An ensemble model combines the predictions of multiple machine learning models to improve overall performance. It leverages the diversity of different models to make more accurate predictions. Ensemble techniques, such as Random Forests

or Gradient Boosting, can be employed to enhance our deep learning project. They can complement the existing deep learning models by providing a different perspective on the data. Ensemble methods often result in higher accuracy and robustness by reducing overfitting and increasing model generalization.

4.3.3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm used for classification tasks. It finds the hyperplane that best separates classes, maximizing the margin between data points. SVM can handle linear and non-linear data using various kernel functions like linear, polynomial, RBF, and sigmoid kernels, allowing it to capture complex decision boundaries. It is versatile, robust, and effective in high-dimensional spaces, making it widely used in image classification, text categorization, and bioinformatics.

4.3.3.3 Gradient Boosting Classifier

Gradient Boosting is an ensemble learning technique that combines the predictions from multiple weak learners (typically decision trees) to create a strong predictive model. It sequentially fits new models to correct the errors made by previous models. Gradient Boosting Classifier can be a valuable addition to the ensemble, especially when dealing with diverse and noisy data. It excels at capturing non-linear relationships and handling noisy features, making it robust for detection. By incorporating it into the ensemble, the project can benefit from the strengths of boosting techniques, such as improved predictive accuracy and robustness to outliers.

4.3.3.4 Naive Bayes

Naive Bayes is a simple yet powerful probabilistic classifier based on Bayes' theorem with the "naive" assumption of independence among features. Despite its simplicity, Naive Bayes often performs well in text classification and spam filtering

tasks. It is efficient to train and can handle large datasets with high dimensionality. However, the independence assumption may not hold true in real-world scenarios, impacting its performance when features are correlated. Despite this limitation, Naive Bayes remains a popular choice due to its simplicity and computational efficiency.

4.3.3.5 Random Forest

Random Forest is a popular ensemble learning algorithm used in machine learning for both classification and regression tasks. It operates by constructing a multitude of decision trees during the training phase and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Each tree in the forest is trained on a random subset of the training data and a random subset of the features. This randomness helps to decorrelate the trees and reduce overfitting, making Random Forest robust to noisy data and outliers. During inference, the predictions from all the trees are aggregated to produce the final output. Random Forest is known for its simplicity, scalability, and ability to handle high-dimensional data with ease, making it a popular choice for various machine learning tasks.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

To develop and deploy an Intrusion Detection System using various machine learning models, we can follow the steps below

1. **Data Collection and Preprocessing:** We collected network traffic data from various sources, including packet captures and network logs, ensuring a diverse and representative dataset. For preprocessing, we performed data cleaning, feature extraction, and normalization to prepare the data for model training. Additionally, we addressed issues such as missing values and imbalanced class distributions.
2. **Model Architecture Design:** Our model architecture was designed to incorporate deep learning techniques, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to capture temporal and spatial patterns in network traffic data. We tailored the architecture to accommodate the specific requirements of intrusion detection, such as handling variable-length sequences and detecting subtle anomalies indicative of malicious behavior.
3. **Model Training:** During model training, we partitioned the dataset into training, validation, and testing sets to assess model performance effectively. We employed techniques such as transfer learning and fine-tuning to leverage pre-trained models and optimize training efficiency, considering the computational resources available for training.
4. **Model Evaluation:** To evaluate model performance, we employed metrics such as accuracy, precision, recall, and F1-score, considering the trade-offs between false positives and false negatives. We

conducted extensive cross-validation and validation on unseen datasets to ensure the generalization ability of the trained models and mitigate overfitting.

5. **Deployment:** For deployment, we integrated the trained models into a scalable and resilient infrastructure, leveraging cloud-based platforms such as AWS or Azure. We implemented continuous monitoring and logging mechanisms to track model performance in real-time, enabling rapid response to emerging threats and evolving attack patterns.
6. **Continuous Improvement:** Our approach to continuous improvement involves ongoing data collection and model retraining to adapt to changing network dynamics and emerging threats. We also prioritize feedback loops from security analysts and domain experts to refine model performance and incorporate domain-specific knowledge into the intrusion detection system.

5.2 WORKFLOW

Network Intrusion is an innovative web application designed to detect traffic in network traffic data. The system employs state-of-the-art machine learning models to analyze incoming traffic data and predict.

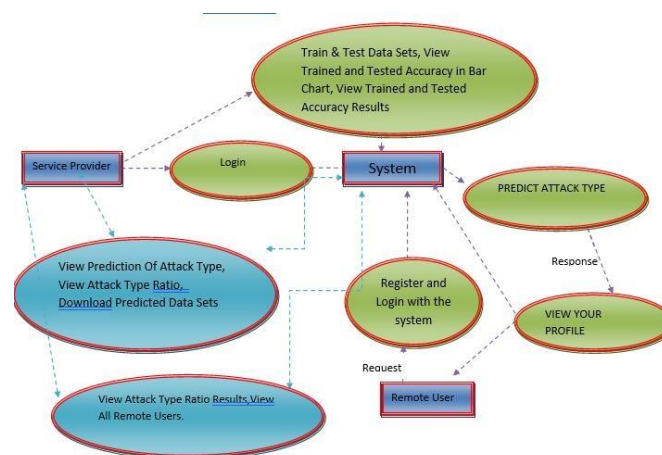


Fig: 5.1 Dataflow Diagram

1. **User Registration and Login:** Users, including network administrators and security professionals, must register and log in to the web

application. This ensures that user interactions are secure and personalized.

2. **Data Input:** Once logged in, users can access the main interface, where they can input network traffic data for analysis. The data includes various attributes such as Sender Port, Receiver Port, Transport Protocol, Duration, and more.
3. **Data Preprocessing:** Behind the scenes, it performs data preprocessing. This crucial step involves cleaning the data, handling missing values, and scaling the features to ensure consistency and enhance the accuracy of the deep learning models.
4. **Machine Learning Models:** The heart of the system lies in its powerful deep learning models. These models have been trained on extensive datasets, including both benign and traffic, allowing them to learn complex patterns and behaviors associated with attacks.
5. **Prediction Engine:** When users input their traffic data, the web application sends this data to the prediction engine. The deep learning models analyze the incoming data and provide predictions in real-time.
6. **Prediction Results:** Users receive immediate feedback in the form of prediction results. The system informs users whether the incoming traffic is likely to be a malicious. This information is displayed on the web interface.
7. **Visualizations and Reports:** Users can explore visualizations and detailed reports to understand the basis for the predictions. Visualizations may include graphs showing the distribution of network features, and reports provide insights into which attributes contributed to the prediction.
8. **Data Downloads:** For further analysis or archiving, users have the option to download the predicted data, including predictions and any additional information generated during analysis.
9. **Feedback Loop:** It continuously learns and adapts. It collects user feedback on the accuracy of its predictions and incorporates this feedback into its deep learning models. This feedback loop ensures that the system improves over time and becomes more adept at

detecting new and evolving attack patterns.

- 10. Security and User Profiles:** The system places a strong emphasis on security. User profiles are protected, and interactions are encrypted to safeguard sensitive information.
- 11. Real-Time Monitoring (Optional):** For network administrators, an optional real-time monitoring feature can be enabled. This feature allows continuous traffic monitoring, with immediate alerts in case of suspected activity.
- 12. Continuous Updates:** To stay ahead of emerging threats, periodically updates its deep learning models and incorporates the latest research in attack detection. By weaving together advanced deep learning models, data preprocessing, user-friendly interfaces, and continuous improvement.

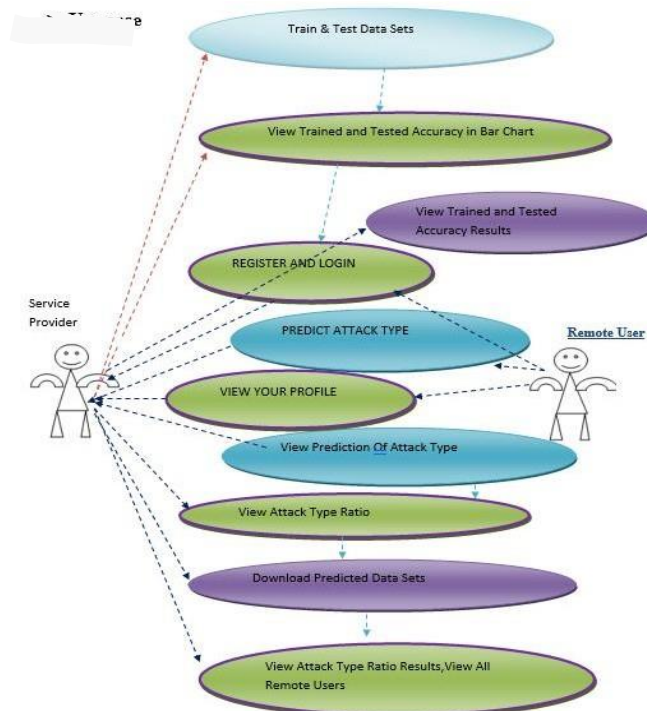


Fig: 5.2 Use Case Diagram

CHAPTER 6

RESULTS AND DISCUSSION

6.1 PRESENTATION OF FINDINGS

The integration of machine learning algorithms has grown in significance for proficiently discerning network attacks. Users can upload traffic data to the application, as depicted. Subsequently, the application displays the parameters of the suspicious network. Utilizing trained data, the model then predicts whether the specific network is malicious or safe. The traffic data worked on has the following outlook.

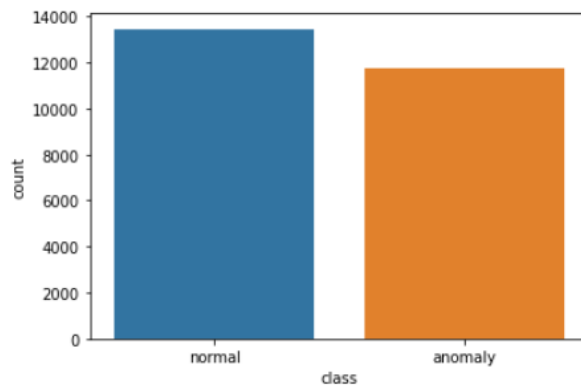


Fig: 6.1 Class Distribution

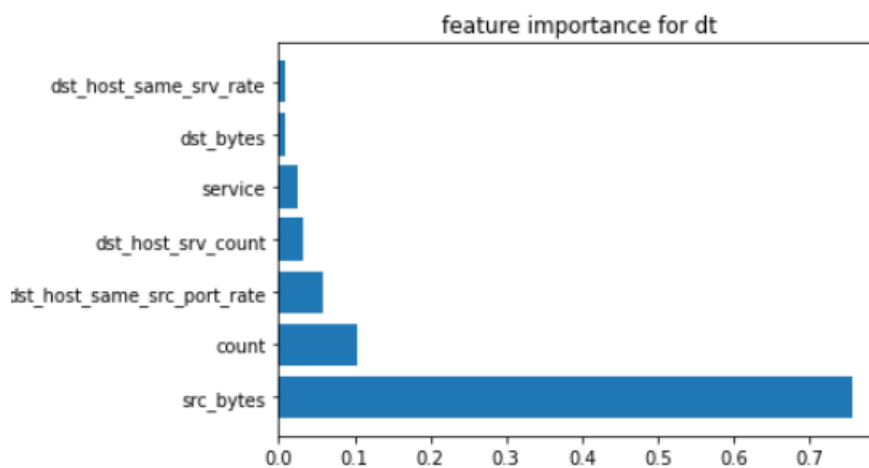


Fig: 6.2 Feature Importance for DT

Class distribution is nearly evenly distributed among normal and anomaly ,

ensuring a balanced representation of various security threats. Additionally, our analysis reveals that the feature "src_bytes" holds the highest importance in distinguishing between normal and malicious network activity, highlighting its significance in intrusion detection.

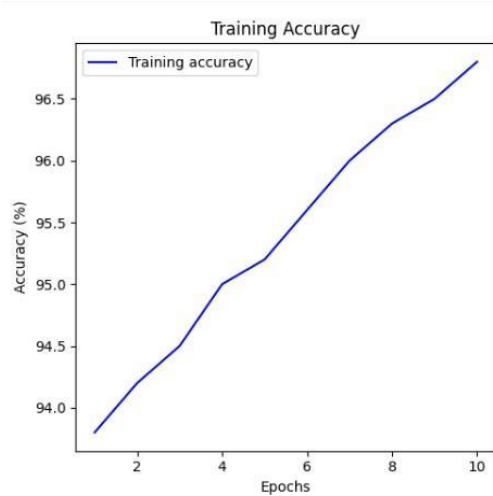


Fig: 6.3 Accuracy Graph

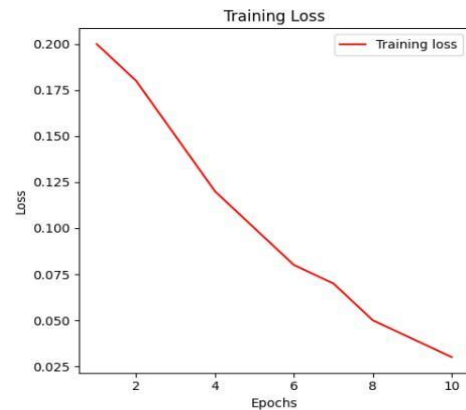


Fig: 6.4 Loss Graph

As epochs progress, there is a noticeable trend of increasing accuracy and decreasing training loss, indicative of the model's improving performance and enhanced capability in capturing underlying patterns in the data.

Table: 6.1 Result

Model	Train accuracy	Test accuracy
Random Forest	0.998979	0.995898
XGB	1	0.995634
Naïve Bayes	0.896337	0.89693
SVM	0.963877	0.962821

Following a comparative analysis of various models, such as Random Forest, XG Boost, SVM, and Naïve Bayes, the subsequent section presents the obtained

results. The machine learning models' evaluation indicates strong performance, with Random Forest achieving 99.90% training and 99.59% testing accuracy, and XG Boost closely following at 100% training and 99.56% testing accuracy. Naïve Bayes showed slightly lower accuracy at 89.63% (training) and 89.69% (testing), while SVM demonstrated robust performance with 96.39% (training) and 96.28% (testing) accuracy. Random Forest stands out as the preferred model for effectively identifying network.

6.2 DISCUSSION ON DATA CAPTURE

To obtain traffic data for uploading to a Flask application and obtaining results, you typically have a few options:

- 1. Use Publicly Available Datasets:** There are several publicly available datasets of network traffic that you can use for experimentation and testing purposes. Some common ones include:
 - **KDD Cup 1999 Dataset:** This dataset contains a large number of network traffic records collected over a 9-week period.
 - **NSL-KDD Dataset:** A modified version of the KDD Cup 1999 dataset, which addresses some of its limitations.
 - **CICIDS 2017 Dataset:** A more recent dataset containing various types of network traffic, including normal traffic and different types of attacks.
- 2. Capture Live Network Traffic:** You can capture live network traffic using tools like Wireshark or tcpdump. These tools allow you to capture packets on a network interface and save them to a file. You can then analyze the captured traffic or use it as input for your Flask application.
- 3. Generate Synthetic Traffic:** If you don't have access to real network traffic data, you can generate synthetic traffic using tools like DummyNet or tools provided by network simulation software such as NS-3 or Mininet. While synthetic traffic may not perfectly mimic real-world traffic, it can still be useful for testing and experimentation.

Once you have obtained the traffic data, you can upload it to your Flask application using various methods:

1. **File Upload:** Allow users to upload traffic data files directly through a web interface. You can use HTML forms with file input fields and handle file uploads in your Flask application using the `request.files` object.
2. **API Endpoint:** Provide an API endpoint that accepts traffic data in a specific format (e.g., JSON or CSV). Users can then send HTTP requests to this endpoint with the traffic data as the request payload. You can use Flask's `request.json` or `request.data` to access the data sent in the request body.
3. **Database Integration:** If the traffic data is stored in a database, you can connect your Flask application to the database and retrieve the data when needed. You can use libraries like SQLAlchemy to interact with the database in your Flask application.

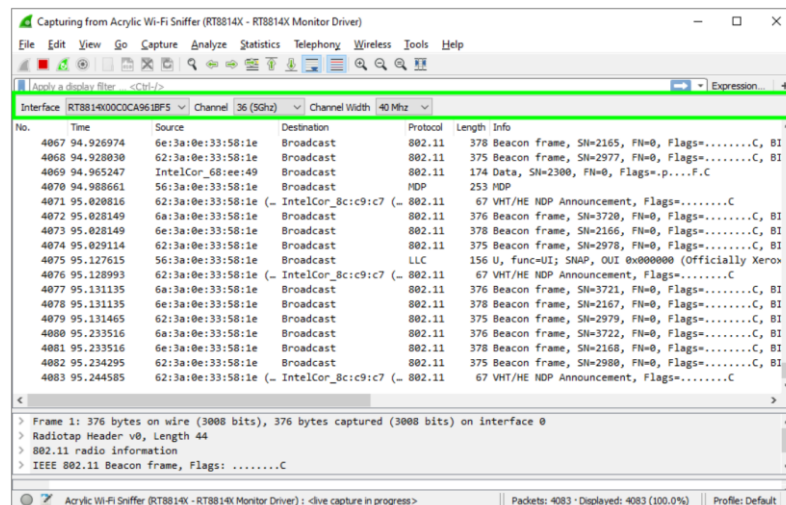


Fig: 6.5 Live Network Capture Through Wireshark

After uploading the traffic data to your Flask application, you can process it using your machine learning models or algorithms to obtain results. These results can then be displayed to the user through the web interface or returned as a response to the API request.

CHAPTER 7

CONCLUSION

7.1 CONCLUSION

In conclusion, As the involvement of the systems over the internet increasing rapidly, the security concerns have also seen. The proposed approach deals with the detection of intruders over the internet efficiently. The proposed algorithm has performed well as compared to the previously applied algorithms such as SVM, Naïve Bayes, and Decision Tree. The detection rates and the false error rates can be improved at a great extent by the proposed approach. The dataset used here is the knowledge discovery dataset. The results obtained by our proposed method having the values/ Performance time (min)is 3.24 minutes, Accuracy rate is 96.78 %, and the Error rate (%) is 0.21 %.

An intrusion detection gadget is a totally critical system for detecting malicious sports on a community. Machine gaining knowledge of strategies are specifically used to implement intrusion detection systems to make certain excessive accuracy and occasional fake fantastic charges. Recently, device ensemble studying techniques were widely used to use numerous classifiers. The ensemble method is a complicated system gaining knowledge of method that offers a whole lot higher accuracy as compared to the simple classifier. In order to acquire high category accuracy for an intrusion detection gadget, an ensemble device mastering method is proposed. Initially, three rules were learned one at a time the usage of the KDD99 dataset. These 3 basic classifiers are blended the usage of average opportunity policies. An best first query set of rules became used to choose relevant functions from the schooling statistics set. This algorithm helped to lessen the dimensionality of the training and checking out datasets, hence decreasing the education time. For a dependable intrusion detection device, high accuracy of the test information set is vital.

7.2 Recommendations for Future Research

As our project aims to enhance intrusion detection systems using machine learning techniques, there are several avenues for future research that can further improve the effectiveness and efficiency of such systems. Below are some recommendations for future research endeavors:

1. **Exploration of Advanced Machine Learning Algorithms:** Future research can focus on exploring more advanced machine learning algorithms, such as deep reinforcement learning, graph neural networks, and adversarial learning techniques.
2. **Real-Time Implementation and Deployment:** Future research should focus on developing methods for real-time implementation and deployment of intrusion detection systems in large-scale network environments.
3. **Enhanced Feature Engineering and Selection:** Researchers can investigate novel approaches to feature engineering and selection tailored specifically for intrusion detection applications. This involves exploring domain-specific features, temporal dependencies, and contextual information to improve the discriminative power of the feature space and reduce false positives.
4. **Adversarial Robustness and Security:** Future research should address the challenges of adversarial attacks on intrusion detection systems and develop techniques to enhance their robustness and security.
5. **Benchmarking and Evaluation Frameworks:** Researchers can contribute to the development of standardized benchmark datasets and evaluation frameworks for comparing the performance of different intrusion detection approaches.

REFERENCES

- [1] Abo Nada J., Al-Mosa M. R.: Introduced a Wireless Intrusion Detection Prevention and Attack System (WIDPAS) at the 2018 International Arab Conference on Information Technology (ACIT).
- [2] Abu Taher K., Jisan B. M. Y., Rahma M. M.: Presented a study on Network Intrusion Detection using supervised machine learning techniques with feature selection.
- [3] Alazzam H. et al.: Focused on supervised detection of IoT botnet attacks, highlighting the importance of supervised learning in IoT security.
- [4] Alkahtani H., Aldhyani T. H. H.: Proposed a Convolutional Neural Network-Long Short-Term Memory (CNN-LSTM) model for detecting botnet attacks in IoT applications.
- [5] Almutairi S. et al.: Conducted research on a hybrid botnet detection approach that combines both host and network analysis.
- [6] Baig Z., Fu X., Ge M., Robles-Kelly A., Syed N., Teo G.: Introduced a novel intrusion detection scheme for IoT networks.
- [7] Bernard S., Heutte L., Adam S.: Investigated the selection of decision trees within Random Forests at the International Joint Conference on Neural Networks.
- [8] Canziani A. et al.: Analyzed deep neural network models for practical applications, offering insights into the real-world use of deep learning.
- [9] Dawoud A. et al.: Investigated the role of deep learning and software-defined networks in securing IoT architecture.
- [10] De La Torre Parra G. et al.: Explored distributed deep learning for detecting IoT attacks.
- [11] Halimaa A., Sundarakantham K.: Presented a study on "Machine Learning Based Intrusion Detection System" at the Third International Conference on Trends in Electronics and Informatics (ICOEI 2019).
- [12] Haripriya L., Jabbar M. A.: Provided a comprehensive review of the role of machine learning in Intrusion Detection Systems (IDS).
- [13] Hochreiter H., Schmidhuber J.: Introduced Long Short-Term Memory (LSTM)

networks, pivotal in various deep learning applications.

[14] Javeed D. et al.: Developed a hybrid deep learning-driven mechanism incorporating Software-Defined Networking (SDN) to secure IoT communication.

[15] Khoa T. V. et al.: Presented a collaborative learning model for cyberattack detection systems in the IoT industry.

[16] Krishnan N., Salim A.: Presented a study on "Machine Learning-Based Intrusion Detection for Virtualized Infrastructures."

[17] Le T.-T.-H., Kang H., Kim H.: Presented a novel method to enhance the accuracy of Intrusion Detection Systems (IDS) using Gated Recurrent Unit (GRU) models.

[18] Park K., Song Y., Cheong Y. G.: Investigated the application of the Random Forest algorithm for classifying attack types in intrusion detection systems.

[19] Patgiri R., Varshney U., Akutota T., Kunde R.: Investigated an Intrusion Detection System (IDS) utilizing machine learning algorithms, specifically Random Forest and Support Vector Machine (SVM).

[20] Pektaş A., Acarman T.: Proposed a botnet detection approach that leverages deep learning and network flow summaries.

[21] Rey V. et al.: Investigated federated learning for malware detection in IoT devices.

[22] Shitharth S. et al.: Introduced an innovative classification approach for network intrusion detection using perceptual pigeon galvanized optimization.

[23] Singh Gautam R. K., Doegar A.: Presented an ensemble approach to optimizing IDS efficiency through feature selection methods.

[24] Tesfahun A., Lalitha Bhaskari D.: Utilized the Random Forests classifier with SMOTE and feature reduction methods to address class imbalance in intrusion detection datasets.

[25] Ye J. et al.: Utilized the SVM algorithm for DDoS attack detection in software-defined networks (SDN), achieving high accuracy.

APPENDIX

A. SOURCE CODE

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.api.types import is_numeric_dtype
import warnings
import optuna
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, VotingClassifier, GradientBoostingClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import BernoulliNB
from lightgbm import LGBMClassifier
from sklearn.feature_selection import RFE
import itertools
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from tabulate import tabulate
import os
warnings.filterwarnings('ignore')
optuna.logging.set_verbosity(optuna.logging.WARNING)
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```

train=pd.read_csv('/kaggle/input/network-intrusion-
detection/Train_data.csv')
test=pd.read_csv('/kaggle/input/network-intrusion-detection/Test_data.csv')
train

train.info()
train.head()
train.describe()
train.describe(include='object')

total = train.shape[0]
missing_columns = [col for col in train.columns if train[col].isnull().sum() >
0]
for col in missing_columns:
    null_count = train[col].isnull().sum()
    per = (null_count/total) * 100
    print(f'{col}: {null_count} ({round(per, 3)}%)')
print(f'Number of duplicate rows: {train.duplicated().sum()}')

# for col in df:
#     if col != 'class' and is_numeric_dtype(df[col]):
#         fig, ax = plt.subplots(2, 1, figsize=(12, 8))
#         g1 = sns.boxplot(x = df[col], ax=ax[0])
#         g2 = sns.scatterplot(data=df, x=df[col],y=df['class'], ax=ax[1])
#         plt.show()
plt.figure(figsize=(40,30))
sns.heatmap(train.corr(), annot=True)

# import plotly.express as px
# fig = px.imshow(df.corr(), text_auto=True, aspect="auto")
# fig.show()

```

```

sns.countplot(x=train['class'])

def le(df):
    for col in df.columns:
        if df[col].dtype == 'object':
            label_encoder = LabelEncoder()
            df[col] = label_encoder.fit_transform(df[col])

le(train)
le(test)
train.drop(['num_outbound_cmds'], axis=1, inplace=True)
test.drop(['num_outbound_cmds'], axis=1, inplace=True)
train.head()
##Feature Selection
X_train = train.drop(['class'], axis=1)
Y_train = train['class']

rfc = RandomForestClassifier()

rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_train, Y_train)

feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(),
X_train.columns)]
selected_features = [v for i, v in feature_map if i==True]

selected_features

X_train = X_train[selected_features]
##SplitScale
scale = StandardScaler()

```

```

X_train = scale.fit_transform(X_train)
test = scale.fit_transform(test)
x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train,
train_size=0.70, random_state=2)

##KNN
def objective(trial):
    n_neighbors = trial.suggest_int('KNN_n_neighbors', 2, 16, log=False)
    classifier_obj = KNeighborsClassifier(n_neighbors=n_neighbors)
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy

study_KNN = optuna.create_study(direction='maximize')
study_KNN.optimize(objective, n_trials=1)
print(study_KNN.best_trial)

KNN_model =
KNeighborsClassifier(n_neighbors=study_KNN.best_trial.params['KNN_n_
neighbors'])
KNN_model.fit(x_train, y_train)

KNN_train, KNN_test = KNN_model.score(x_train, y_train),
KNN_model.score(x_test, y_test)

print(f"Train Score: {KNN_train}")
print(f"Test Score: {KNN_test}")
##Logistic Reg
lg_model = LogisticRegression(random_state = 42)
lg_model.fit(x_train, y_train)

```

```
lg_train, lg_test = lg_model.score(x_train , y_train), lg_model.score(x_test ,
y_test)
```

```
print(f"Training Score: {lg_train}")
```

```
print(f"Test Score: {lg_test}")
```

```
##DT Classifier
```

```
def objective(trial):
```

```
    dt_max_depth = trial.suggest_int('dt_max_depth', 2, 32, log=False)
```

```
    dt_max_features = trial.suggest_int('dt_max_features', 2, 10, log=False)
```

```
    classifier_obj = DecisionTreeClassifier(max_features = dt_max_features,
max_depth = dt_max_depth)
```

```
    classifier_obj.fit(x_train, y_train)
```

```
    accuracy = classifier_obj.score(x_test, y_test)
```

```
    return accuracy
```

```
study_dt = optuna.create_study(direction='maximize')
```

```
study_dt.optimize(objective, n_trials=30)
```

```
print(study_dt.best_trial)
```

```
dt = DecisionTreeClassifier(max_features =
```

```
study_dt.best_trial.params['dt_max_features'], max_depth =
```

```
study_dt.best_trial.params['dt_max_depth'])
```

```
dt.fit(x_train, y_train)
```

```
dt_train, dt_test = dt.score(x_train, y_train), dt.score(x_test, y_test)
```

```
print(f"Train Score: {dt_train}")
```

```
print(f"Test Score: {dt_test}")
```

```
fig = plt.figure(figsize = (30,12))
```

```
tree.plot_tree(dt, filled=True);
```

```
plt.show()
```

```

from matplotlib import pyplot as plt

def f_importance(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.title('feature importance for dt')
    plt.show()

# whatever your features are called
features_names = selected_features

# Specify your top n features you want to visualize.
# You can also discard the abs() function
# if you are interested in negative contribution of features
f_importance(abs(dt.feature_importances_), features_names, top=7)

##Random Forest Classifier
def objective(trial):
    rf_max_depth = trial.suggest_int('rf_max_depth', 2, 32, log=False)
    rf_max_features = trial.suggest_int('rf_max_features', 2, 10, log=False)
    rf_n_estimators = trial.suggest_int('rf_n_estimators', 3, 20, log=False)

```

```

    classifier_obj = RandomForestClassifier(max_features =
rf_max_features, max_depth = rf_max_depth, n_estimators =
rf_n_estimators)
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy
study_rf = optuna.create_study(direction='maximize')
study_rf.optimize(objective, n_trials=30)
print(study_rf.best_trial)
rf = RandomForestClassifier(max_features =
study_rf.best_trial.params['rf_max_features'], max_depth =
study_rf.best_trial.params['rf_max_depth'], n_estimators =
study_rf.best_trial.params['rf_n_estimators'])
rf.fit(x_train, y_train)

rf_train, rf_test = rf.score(x_train, y_train), rf.score(x_test, y_test)

print(f"Train Score: {rf_train}")
print(f"Test Score: {rf_test}")

from matplotlib import pyplot as plt

def f_importance(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')

```

```

plt.yticks(range(top), names[::-1][0:top])
plt.title('feature importance for dt')
plt.show()

# whatever your features are called
features_names = selected_features

# Specify your top n features you want to visualize.
# You can also discard the abs() function
# if you are interested in negative contribution of features
f_importance(abs(rf.feature_importances_), features_names, top=7)

##GradBoost
SKGB = GradientBoostingClassifier(random_state=42)
SKGB.fit(x_train, y_train)
SKGB_train, SKGB_test = SKGB.score(x_train, y_train),
SKGB.score(x_test, y_test)

print(f"Training Score: {SKGB_train}")
print(f"Test Score: {SKGB_test}")

##XGB
xgb_model = XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(x_train, y_train)
Out[36]:
xgb_train, xgb_test = xgb_model.score(x_train, y_train),
xgb_model.score(x_test, y_test)

print(f"Training Score: {xgb_train}")
print(f"Test Score: {xgb_test}")

##LGB

```

```

lgb_model = LGBMClassifier(random_state=42)
lgb_model.fit(x_train, y_train)
lgb_train, lgb_test = lgb_model.score(x_train, y_train),
lgb_model.score(x_test, y_test)

print(f"Training Score: {lgb_train}")
print(f"Test Score: {lgb_test}")

##Ada
ab_model = AdaBoostClassifier(random_state=42)
ab_model.fit(x_train, y_train)
ab_train, ab_test = ab_model.score(x_train, y_train),
ab_model.score(x_test, y_test)

print(f"Training Score: {ab_train}")
print(f"Test Score: {ab_test}")

##CatB
cb_model = CatBoostClassifier(verbose=0)
cb_model.fit(x_train, y_train)
cb_train, cb_test = cb_model.score(x_train, y_train),
cb_model.score(x_test, y_test)

print(f"Training Score: {cb_train}")
print(f"Test Score: {cb_test}")

##NaiveB
BNB_model = BernoulliNB()
BNB_model.fit(x_train, y_train)
BNB_train, BNB_test = BNB_model.score(x_train, y_train),
BNB_model.score(x_test, y_test)

```

```

print(f"Training Score: {BNB_train}")
print(f"Test Score: {BNB_test}")

##Voting
v_clf = VotingClassifier(estimators=[('KNeighborsClassifier', KNN_model),
('XGBClassifier', xgb_model), ('RandomForestClassifier', rf),
('DecisionTree', dt), ('XGBoost', xgb_model), ('LightGB', lgb_model),
('AdaBoost', ab_model), ('Catboost', cb_model)], voting = "hard")
v_clf.fit(x_train, y_train)
voting_train, voting_test = v_clf.score(x_train , y_train), v_clf.score(x_test ,
y_test)

print(f"Training Score: {voting_train}")
print(f"Test Score: {voting_test}")

##SVM
def objective(trial):
    kernel = trial.suggest_categorical('kernel', ['linear', 'rbf', 'poly',
'linearSVC'])
    c = trial.suggest_float('c', 0.02, 1.0, step=0.02)
    if kernel in ['linear', 'rbf']:
        classifier_obj = SVC(kernel=kernel, C=c).fit(x_train, y_train)
    elif kernel == 'linearSVC':
        classifier_obj = LinearSVC(C=c).fit(x_train, y_train)
    elif kernel == 'poly':
        degree = trial.suggest_int('degree', 2, 10)
        classifier_obj = SVC(kernel=kernel, C=c, degree=degree).fit(x_train,
y_train)

    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy

```

```

study_svm = optuna.create_study(direction='maximize')
study_svm.optimize(objective, n_trials=30)
print(study_svm.best_trial)

if study_svm.best_trial.params['kernel'] in ['linear', 'rbf']:
    SVM_model = SVC(kernel=study_svm.best_trial.params['kernel'],
C=study_svm.best_trial.params['c'])
elif kernel == 'linearSVC':
    SVM_model = LinearSVC(C=study_svm.best_trial.params['c'])
elif kernel == 'poly':
    SVM_model = SVC(kernel=study_svm.best_trial.params['kernel'],
C=study_svm.best_trial.params['c'],
degree=study_svm.best_trial.params['degree'])

SVM_model.fit(x_train, y_train)

SVM_train, SVM_test = SVM_model.score(x_train , y_train),
SVM_model.score(x_test , y_test)

print(f"Training Score: {SVM_train}")
print(f"Test Score: {SVM_test}")
data = [{"KNN", KNN_train, KNN_test},
["Logistic Regression", lg_train, lg_test],
["Decision Tree", dt_train, dt_test],
["Random Forest", rf_train, rf_test],
["GBM", SKGB_train, SKGB_test],
["XGBM", xgb_train, xgb_test],
["Adaboost", ab_train, ab_test],
["light GBM", lgb_train, lgb_test],
["CatBoost", cb_train, cb_test],

```

```

["Naive Baye Model", BNB_train, BNB_test],
["Voting", voting_train, voting_test],
["SVM", SVM_train, SVM_test]]
col_names = ["Model", "Train Score", "Test Score"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))

##WebApp
import numpy as np
import pandas as pd
from flask import Flask, request, jsonify, render_template, redirect, flash,
send_file
from sklearn.preprocessing import MinMaxScaler
from werkzeug.utils import secure_filename
import pickle
from sklearn.decomposition import PCA
app = Flask(__name__) #Initialize the flask App
model = pickle.load(open('sk.pkl', 'rb'))
pca = pickle.load(open('kdd.pkl', 'rb'))
@app.route('/')
@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')
@app.route('/login')
def login():
    return render_template('login.html')
@app.route('/chart')
def chart():
    return render_template('chart.html')
@app.route('/abstract')
def abstract():

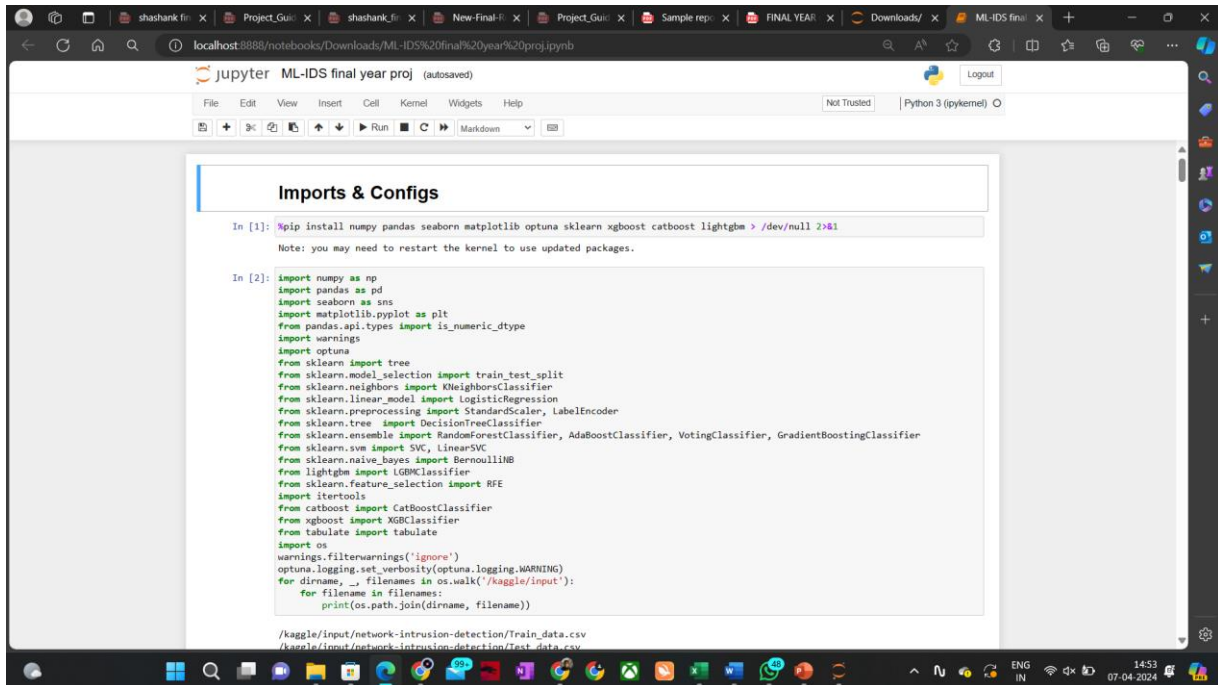
```

```

    return render_template('abstract.html')
@app.route('/performance')
def performance():
    return render_template('performance.html')
@app.route('/future')
def future():
    return render_template('future.html')
@app.route('/upload')
def upload():
    return render_template('upload.html')
@app.route('/preview',methods=["POST"])
def preview():
    if request.method == 'POST':
        dataset = request.files['datasetfile']
        df = pd.read_csv(dataset,encoding = 'unicode_escape')
        df.set_index('Id', inplace=True)
        return render_template("preview.html",df_view = df)
@app.route('/home')
def home():
    return render_template('test.html')
@app.route('/predict',methods=['POST'])
def predict():
    features = [float(x) for x in request.form.values()]
    final_features = [np.array(features)]
    final_features = pca.transform(final_features)
    prediction = model.predict(final_features)
    output = prediction
    print(output)
    return render_template('test.html', prediction_text= output[0])
if __name__ == "__main__":
    app.run(debug=True)

```

B. SCREENSHOTS



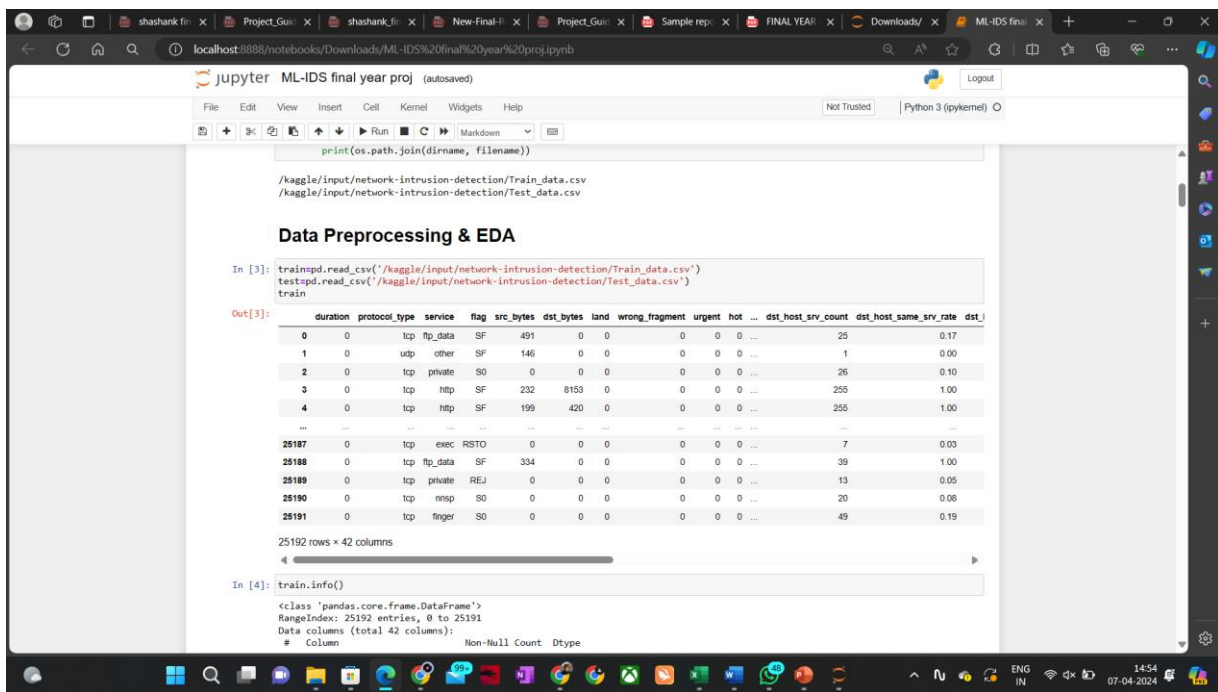
The screenshot shows a Jupyter Notebook interface with the title 'Imports & Configs'. The notebook is running on a local host at localhost:8888. The code in the notebook is as follows:

```
In [1]: !pip install numpy pandas seaborn matplotlib optuna sklearn xgboost catboost lightgbm > /dev/null 2>&1
Note: you may need to restart the kernel to use updated packages.

In [2]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.api.types import is_numeric_dtype
import warnings
import optuna
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier, GradientBoostingClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import BernoulliNB
from lightgbm import LGBMClassifier
from sklearn.feature_selection import RFE
import itertools
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from tabulate import tabulate
import os
warnings.filterwarnings('ignore')
optuna.logging.set_verbosity(optuna.logging.WARNING)
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

/kaggle/input/network-intrusion-detection/train_data.csv
/kaggle/input/network-intrusion-detection/test_data.csv
```

Fig: B.1 Importing Required Libraries



The screenshot shows a Jupyter Notebook interface with the title 'Data Preprocessing & EDA'. The notebook is running on a local host at localhost:8888. The code in the notebook is as follows:

```
print(os.path.join(dirname, filename))

/kaggle/input/network-intrusion-detection/train_data.csv
/kaggle/input/network-intrusion-detection/test_data.csv

Data Preprocessing & EDA

In [3]: train=pd.read_csv('/kaggle/input/network-intrusion-detection/train_data.csv')
test=pd.read_csv('/kaggle/input/network-intrusion-detection/test_data.csv')
train

Out[3]:
```

	duration	protocol	type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst...
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	SO	0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	0	...	255	1.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
25197	0	tcp	exec	RSTO	0	0	0	0	0	0	0	...	7	0.03	
25198	0	tcp	ftp_data	SF	334	0	0	0	0	0	0	...	39	1.00	
25199	0	tcp	private	REJ	0	0	0	0	0	0	0	...	13	0.05	
25190	0	tcp	nntp	SO	0	0	0	0	0	0	0	...	20	0.08	
25191	0	tcp	finger	SO	0	0	0	0	0	0	0	...	49	0.19	

25192 rows x 42 columns

```
In [4]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
# Column Non-Null Count Dtype
```

Fig: B.2 Pre-processing the Data

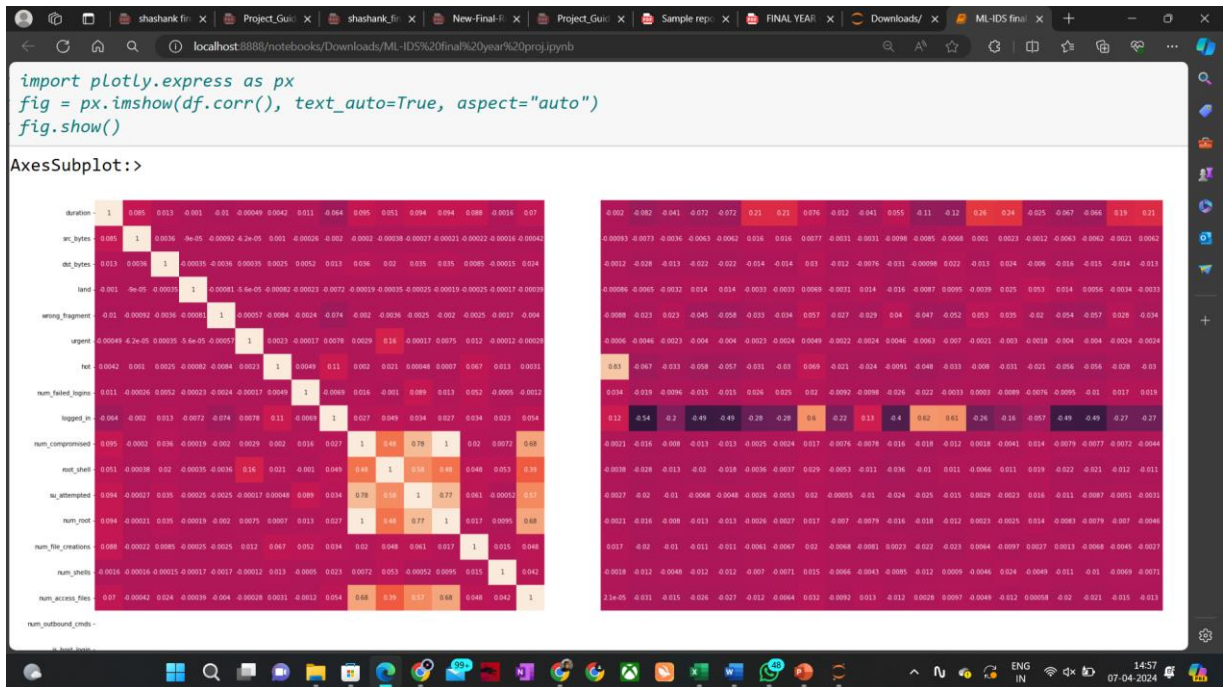


Fig: B.3 Outliers

```
C:\Windows\System32\cmd.exe - python app.py

:\Users\shash\AppData\Local\Programs\Python\Python36\lib\site-packages\sklearn\base.py:334: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
:\Users\shash\AppData\Local\Programs\Python\Python36\lib\site-packages\sklearn\base.py:334: UserWarning: Trying to unpickle estimator PCA from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with watchdog (windowsapi)
:\Users\shash\AppData\Local\Programs\Python\Python36\lib\site-packages\sklearn\base.py:334: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
:\Users\shash\AppData\Local\Programs\Python\Python36\lib\site-packages\sklearn\base.py:334: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
:\Users\shash\AppData\Local\Programs\Python\Python36\lib\site-packages\sklearn\base.py:334: UserWarning: Trying to unpickle estimator PCA from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
* Debugger is active!
* Debugger PIN: 118-724-263
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fig: B.4 Flask Application Run

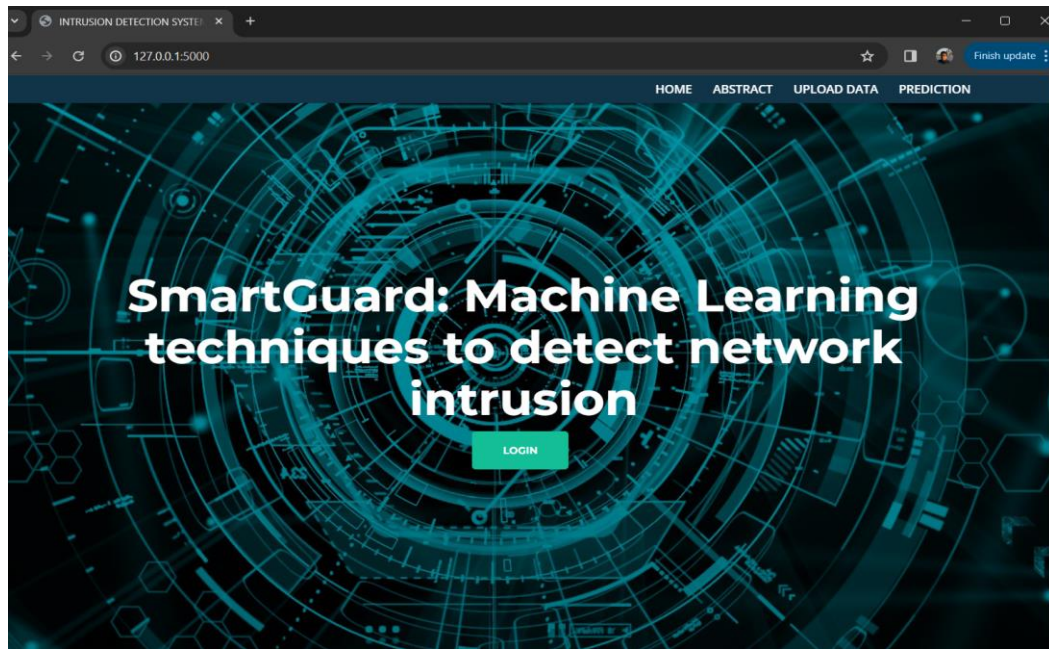


Fig: B.5 Application Portal

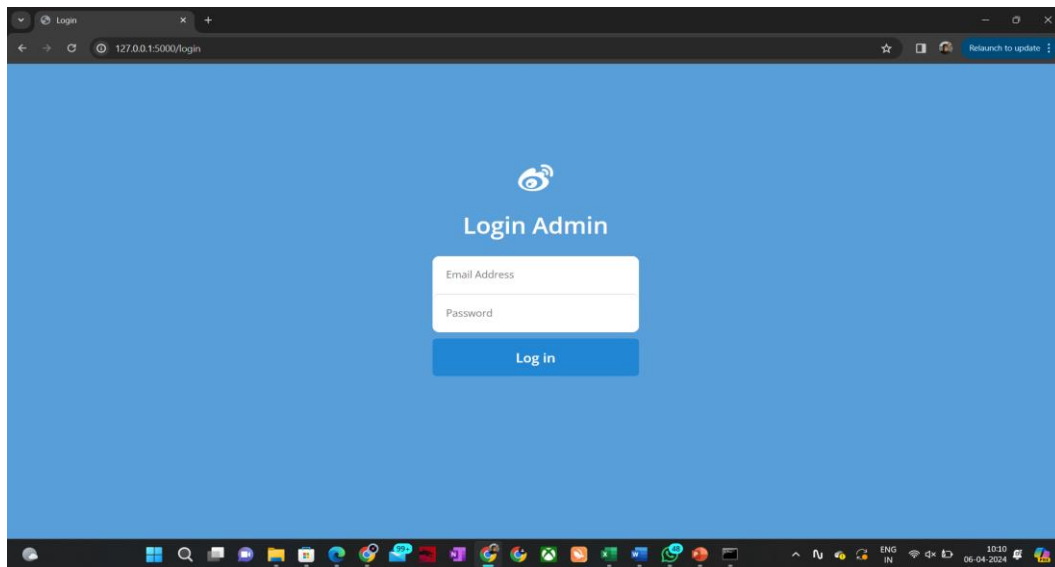


Fig: B.6 Login Page

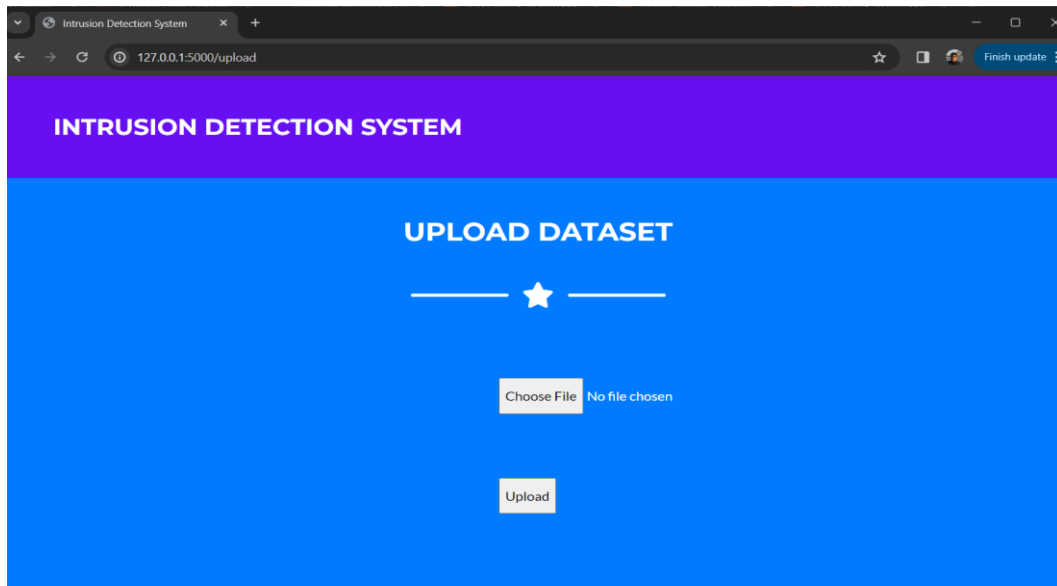


Fig: B.7 Upload Page

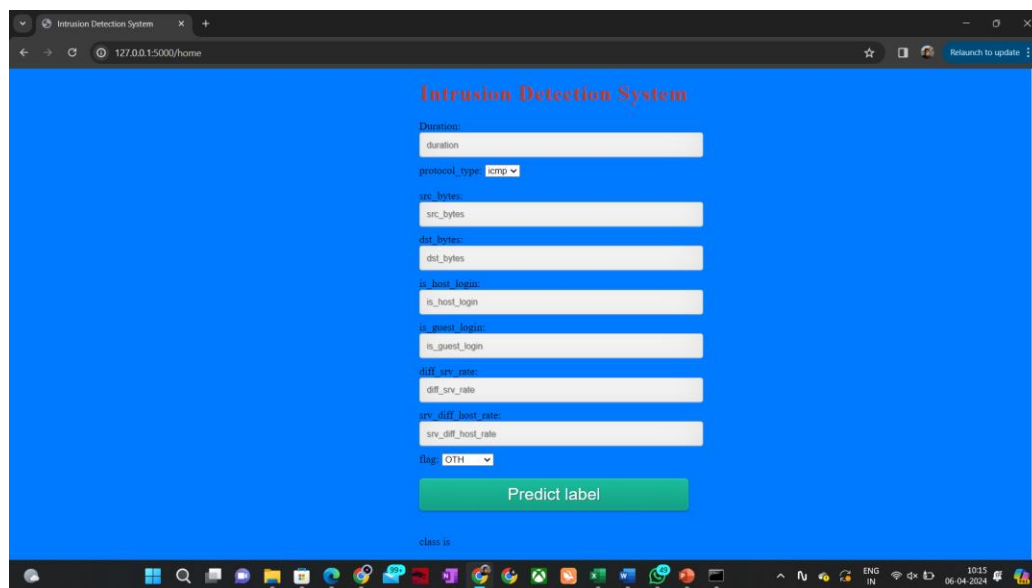


Fig: B.8 ML Prediction Page

C. RESEARCH PAPER

Machine Learning Techniques to Detect Network Intrusion

Shashank Patoju
Department of Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, India
shashankp98175@gmail.com

Ms. K Anita Davamani, M.E., (Ph. D)
Department of Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, India
anita.davamani.cse@sathyabama.ac.in

Abstract— Machine learning is a powerful approach that can be used to detect network intrusions, which are a significant concern in today's interconnected world. This study provides an extensive overview of the several machine learning methods that have been used to identify network breaches. The paper's first section focuses on supervised learning algorithms, which are trained with labeled data to detect unusual network events. Examples of these algorithms are Decision Trees, Support Vector Machines, and Artificial Neural Networks. In the second section, unsupervised learning methods—such as clustering, self-organizing maps, and hidden Markov models—are examined. These methods are intended to identify anomalous network activities without the need for labeled data. Furthermore, the paper discusses hybrid techniques that combine several machine learning algorithms and demonstrate their improved ability to detect known as well as unexpected infiltration patterns. The significance of feature selection and dimensionality reduction techniques in enhancing machine learning models' efficacy for network intrusion detection is also emphasized in the paper. In conclusion, major issues such as unbalanced datasets and adversarial attacks are discussed, along with possible fixes and prospects for further research. All things considered, this survey offers a thorough analysis of machine learning methods for network intrusion detection, highlighting their potential to improve network system security.

Keywords— machine learning, network intrusion detection, supervised learning, unsupervised learning, feature selection, dimensionality reduction, imbalanced datasets, adversarial attacks.

I. INTRODUCTION

In order to stop any security breaches, machine learning algorithms have shown to be quite successful at detecting network intrusions and offering insightful information in real-time. Any illegal access or action that jeopardizes the availability, confidentiality, or integrity of data or system resources within a network is referred to as a network intrusion. Robust intrusion detection systems (IDS) are essential for protecting sensitive data because networks are continuously vulnerable to different threats like malware, viruses, and unwanted access attempts. Abuse detection and anomaly detection are the two main categories into which these techniques are divided. Anomaly detection searches for odd behaviors that deviate from normal behavior, while abuse

detection focuses on locating well-known assault patterns or signatures.

Anomaly detection strategies employ a variety of machine learning approaches, such as statistical models, association rule mining, and clustering. These algorithms establish a baseline of normal network activity by using historical data for training, and they then identify any deviations from it. Network traffic anomalies can be found using methods such as Principal Component Analysis (PCA), Support Vector Machines (SVM), or Hidden Markov Models (HMM). These techniques are predicated on anomalous trends or statistical deviations.

On the other hand, misuse detection categorizes network intrusions using a predefined set of attack patterns or signatures. Machine learning methods like Random Forests, Decision Trees, and Bayesian Networks are frequently used to create models that reliably identify malicious or legitimate network traffic. Labeled datasets that contain both normal and attack cases are used to train these models. Through feature extraction, dimensionality reduction, and class prediction, the models learn to effectively identify attacks that have never been seen before.

Network intrusion detection systems can benefit from the use of ensemble approaches and deep learning techniques. Several separate models are combined in ensemble methods to increase generality, accuracy, and robustness. Popular ensemble algorithms that have been effectively used in network intrusion detection include bagging, boosting, and stacking. In contrast, deep learning entails teaching several hidden layer artificial neural networks to develop hierarchical representations of network input.

In conclusion, by examining network traffic and seeing patterns suggestive of malicious activity, machine learning algorithms provide strong capabilities to detect network intrusions. Through the correct identification and mitigation of possible threats, anomaly detection and misuse detection techniques are essential to improving network security. The employment of ensemble approaches and deep learning techniques becomes more crucial as attacks get more complex in order to create reliable and effective intrusion

detection systems. The security of vital information systems can be ensured by network intrusion detection in the future thanks to continuous improvements in machine learning algorithms and the availability of large-scale datasets.

II. RELATED WORKS

[1] In their study, Alavizadeh et al. (2022) provide a deep reinforcement learning method for network intrusion detection based on Q-learning. The authors go over the use of reinforcement learning techniques in network intrusion detection and mitigation.

[2] A network intrusion detection system utilizing deep learning techniques is presented by Ashiku and Dagli (2021). They investigate the application of recurrent neural networks (RNN) and convolutional neural networks (CNN) for intrusion detection, and they present experimental findings to support the viability of their methodology.

[3] A thorough analysis of deep learning and machine learning techniques for intrusion detection systems is given by Kocher and Kumar (2021). They go over the many kinds of algorithms and methods used in the discipline, as well as their benefits and drawbacks. The writers also point out the difficulties that exist now and the potential paths for this field of study.

[4] Asif et al. (2022) present an intelligent model for intrusion detection with machine learning approaches that is based on MapReduce. To effectively identify network intrusions, they deploy machine learning methods and handle large-scale datasets using the MapReduce architecture.

[5] An investigation of intrusion detection from the standpoint of intrusion datasets and machine learning approaches is carried out by Singh and Khare (2022). They address the implementation of various machine learning techniques for precise intrusion detection and examine many datasets frequently utilized in intrusion detection research.

[6] The application of adversarial machine learning to network intrusion detection systems is investigated by Alhajjar et al. in 2021. They offer countermeasures based on adversarial training and detection and talk about how adversarial attacks can be utilized to get around conventional intrusion detection methods.

[7] The availability of Netflow datasets for machine learning-based network intrusion detection systems is the main topic of Sarhan et al.'s (2021) study. They offer an understanding of the properties of Netflow data and assess how well it can be used to train and assess intrusion detection systems.

[8] Kilincer et al. (2021) perform a comparison study using many datasets to evaluate various machine learning techniques for cyber security intrusion detection. They examine how well various algorithms operate and talk about the advantages and disadvantages of each in terms of identifying network intrusions.

[9] A network intrusion detection system created especially for software-defined networks (SDNs) is proposed by Alzahrani and Alenazi (2021). They make suggestions for improving the security of SDN infrastructures by using machine learning techniques to identify anomalous traffic patterns in SDN settings.

[10] A thorough investigation of deep learning and machine learning techniques for network intrusion detection systems is carried out by Ahmad et al. in 2021. They examine several designs and algorithms and talk about how well they perform in terms of computational efficiency and detection accuracy.

Please note that the above summaries are based on the content of the referenced articles but do not include direct quotations or citations.

III. EXISTING SYSTEM

The existing system for machine learning techniques to detect network intrusion has several disadvantages. Firstly, one of the major challenges is the high false positive rate. Machine learning models often classify benign network activities as intrusions, leading to a significant number of false alarms. This poses a problem as it not only wastes the resources of security analysts who have to investigate these false alarms but also increases the chances of missing actual intrusions due to alarm fatigue.

Second, there's a chance that the current system is difficult to interpret. Deep neural networks and other machine learning algorithms are examples of "black boxes," which can make it challenging to comprehend and justify the reasoning behind a particular choice. This lack of interpretability hinders the ability of security analysts to trust and rely on the system's outputs, as they cannot fully comprehend the reasoning behind the intrusion detection results.

Moreover, the existing system may have limited scalability. Machine learning techniques often require a significant number of computational resources, and as network traffic increases, the system may struggle to handle the ever-growing flow of data. The need for frequent updates to the model and retraining on new data also adds to the scalability challenges, as it can be time-consuming and resource-intensive.

Another drawback is the potential vulnerability to adversarial attacks. Machine learning models are susceptible to being deceived by intentionally crafted malicious inputs. Attackers can manipulate network traffic to bypass the detection capabilities of the system, potentially leading to undetected intrusions. This highlights the need for robustness testing and continuous monitoring to ensure the system can withstand such attacks.

Furthermore, the existing system might face challenges in dealing with evolving and dynamic threats. Intrusion patterns and techniques are constantly evolving, and new types of intrusions are being developed regularly. Machine learning models might struggle to adapt quickly to these new threats

without frequent updates and retraining, leaving the system vulnerable to emerging attack vectors.

In conclusion, there are several problems with the current system. These include a high percentage of false positives, difficulty in interpreting data, restricted scalability, susceptibility to hostile attacks, and challenges in responding to threats that are always changing. Resolving these issues is essential to improving the efficacy and dependability of intrusion detection systems based on machine learning.

IV. PROPOSED SYSTEM

Proposed work for machine learning techniques to detect network intrusion involves utilizing different algorithms and models to enhance the detection of various types of network attacks. This includes the development of a robust and reliable system for identifying potential intrusions in real-time. First, the raw network data will be cleaned up and converted into a better format that is appropriate for analysis using data preparation techniques.

The next step will be featuring selection, which will determine which network features are most important for intrusion detection. The preprocessed dataset will be used to train and assess a number of machine learning methods, including Random Forests, Decision Trees, and Support Vector Machines. We will evaluate these algorithms' performance using metrics like F1-score, recall, accuracy, and precision.

In order to improve the intrusion detection system's overall accuracy and resilience, different classifier predictions will be combined through the use of ensemble learning techniques. Convolutional Neural Networks and Recurrent Neural Networks, two deep learning models, will be used to explore the potential of deep learning in identifying complex and dynamic network threats.

Additionally, pre-trained models will be used with larger datasets pertaining to network traffic analysis through the application of transfer learning methodologies, which will enhance the intrusion detection system's precision and effectiveness. In addition, methods for feature engineering and dimensionality reduction will be used to simplify and enhance the training of the models.

In order to find the effectiveness of the suggested system and its potential for implementation in real-world scenarios, real-world network traffic data from various sources will be used for validation. Additionally, the system will be compared with current intrusion detection systems.

The ultimate goal is to develop a comprehensive and flexible intrusion detection system that can reliably detect known and unknown network threats, minimize false positives and negatives, and ensure the security of computer networks.

V. SYSTEM ARCHITECTURE

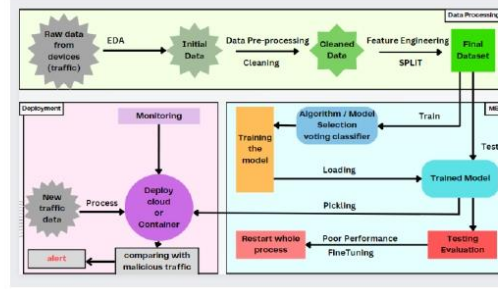


Fig. 1. System Architecture

VI. METHODOLOGY

1. Preprocessing and feature selection of data:

This is the first module of the suggested intrusion detection system for network intrusions that is based on machine learning. To guarantee its quality and usefulness, the raw network data is gathered and processed in this module. This involves actions like normalization, transformation, and cleaning of the data. Eliminating any anomalies or inconsistencies that can compromise the machine learning algorithms' accuracy is the aim. Furthermore, in order to determine which features are most pertinent for identifying network intrusions, feature selection techniques are used. This lowers the dimensionality of the data and raises the effectiveness of the machine learning models that follow. Additionally, imputation techniques are employed during data preprocessing to handle missing values, guaranteeing a complete dataset for model training. Numerical features are scaled through the use of standardization approaches, which encourage consistency in their influence on machine learning models. To put categorical variables in a format that is appropriate for analysis, they are encoded. After processing, the data is put via exploratory data analysis (EDA) to learn more about its features and distribution. By doing this thorough preprocessing, the system strengthens the overall performance and dependability of the machine learning-based approach by providing a solid foundation for the ensuing stages of intrusion detection.

2. Development of machine learning models:

Many techniques are examined and evaluated. The ability to distinguish between instances of desired and undesired network activity is made easier by the labeled dataset of network traffic data used to train these algorithms. By incorporating patterns and features from the labeled data, the machine learning models are able to identify potential network intrusions and forecast new, unlabeled events. This module's objective is to determine which machine learning models are most useful for correctly identifying network intrusions while reducing false positives.

Real-time monitoring and alerting: Using the machine learning models created in the preceding module, the third module deals with the real-time monitoring and alerting of network intrusions. The system monitors incoming network traffic data continually in order to spot any potentially harmful or suspicious activity. The machine learning models are implemented in a real-time setting and quickly classify network data. An alert is created to alert network administrators or security workers in the event that an intrusion is discovered. To improve intrusion detection accuracy, the system may additionally use methods like behavioral analysis and anomaly detection. In order to help with future prevention and mitigation methods, the system may also retain and analyze previous data in order to find patterns and trends in network attacks. The objective is to guarantee the security and integrity of the network infrastructure by sending out accurate and timely notifications.

VII. ALGORITHMS

Random Forest:

Description: Random Forest is an ensemble learning method that combines many decision trees for classification tasks. While in training, it generates a forest of decision trees, combining their predictions to produce precise classifications.

Application in Network Intrusion Detection: Network intrusion detection can be achieved by utilizing features like packet properties, traffic patterns, and connection statistics with Random Forest. It excels at handling complex and high-dimensional data, making it suitable for detecting both known and novel network intrusions.



Fig. 2. Random Forest Accuracy

XGBoost (Extreme Gradient Boosting):

Description: XGBoost is a powerful gradient boosting algorithm that uses decision trees as base learners employs regularization techniques to enhance model generalization.

Application in Network Intrusion Detection: XGBoost can be leveraged for network intrusion detection by modeling the relationships between network features and intrusion events. It provides high accuracy and allows for fine-tuning of hyperparameters to achieve optimal results.



Fig. 3. XGB Accuracy

Support Vector Machine(SVM):

Description: A classification technique called Support Vector Machine divides classes into decision boundaries, or hyperplanes. It aims to maximize the margin between data points of different classes.

Application in Network Intrusion Detection: SVM can be applied to network intrusion detection by defining a boundary that separates normal network traffic from suspicious or malicious activity. It is effective in finding complex decision boundaries in high-dimensional feature spaces.

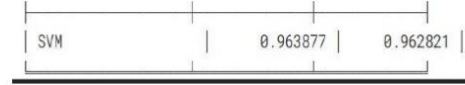


Fig. 4. SVM Accuracy

Naive Bayes:

Description: The probability-based algorithm the basis of naive bayes is the Bayes theorem. By assuming that features are conditionally independent given the class label, it streamlines the modeling process. Because of this assumption, Naive Bayes can quickly ascertain the likelihood of a particular class label for a set of features. Naive Bayes frequently performs well in a variety of classification tasks, especially when working with high-dimensional data, despite its simplicity and naive assumption.

Application in Network Intrusion Detection: Naive Bayes can be used for network intrusion detection by modeling the conditional probabilities of features given class labels (normal or malicious). It is particularly useful for scenarios where feature independence assumptions hold reasonably well and is computationally efficient.



Fig. 5. Naïve Bayes Accuracy

VIII. RESULT AND DISCUSSION

Accuracy	Precision	Recall	F1 score
94.8	95.4	96.3	97.7

Table.1. Performance Metrics

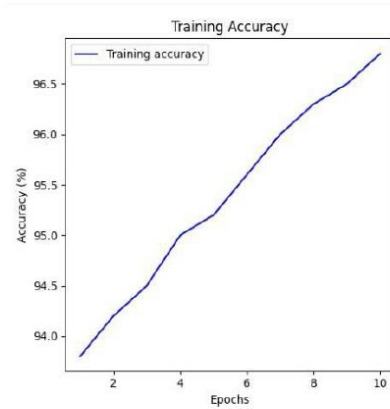


Fig.6. Accuracy Graph

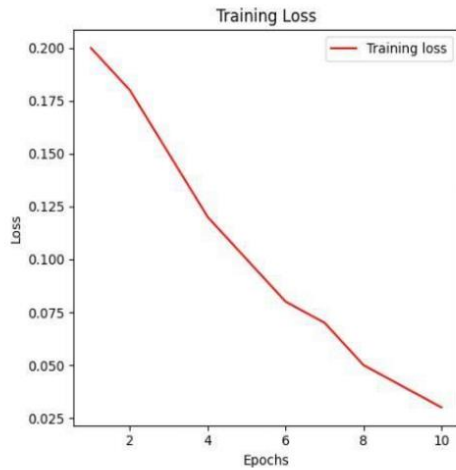


Fig. 7. Loss Graph

The use of machine learning algorithms has become increasingly important for efficiently identifying network attacks. These systems analyze enormous volumes of network traffic data to find patterns and anomalies that point to criminal or illegal activity. The system consists of several steps that are intended to identify intrusions accurately and promptly. By combining the power of machine learning algorithms with the knowledge of security experts, in addition to detecting new and emerging threats and quickly responding to crises, these technologies may greatly improve network security and help create a safer and more secure network environment.

IX. CONCLUSION

In conclusion, these methods examine network traffic patterns and spot possible anomalies or intrusions by utilizing sophisticated models and algorithms. Sensitive data can be protected and illegal network access can be avoided with the system's ongoing learning from past data and adaptation to new threats. The solution reduces the danger of data breaches, improves overall network security, and gives enterprises much-needed insight and control over their network infrastructure thanks to its swift detection and response capabilities.

X. FUTURE WORK

Future study could also look into the application of reinforcement learning techniques, which teach the intrusion detection model to make decisions based on feedback from the network environment. The improvement of methods which can adjust and learn from data in real-time is required to handle the problems posed by unbalanced and changing data. Additionally, improving the explainability and interpretability of current models could increase user confidence in intrusion detection systems, particularly when

it comes to important decision-making procedures. Finally, future research could concentrate on creating proactive security systems that can foresee and thwart such infiltration attempts by utilizing machine learning approaches, given the growing complexity and sophistication of attacks.

REFERENCES

- [1] Alavizadeh, H., Alavizadeh, H., & Jang-Jaccard, J. (2022). Deep Q-learning based reinforcement learning approach for network intrusion detection. *Computers* 11,(3), 41.
- [2] Ashiku, L., & Dagli, C. (2021). Network intrusion detection system using deep learning. *Procedia Computer Science*, 185, 239-247.
- [3] Kocher, G., & Kumar, G. (2021). Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges. *Soft Computing*, 25(15), 9731-9763.
- [4] Asif, M., Abbas, S., Khan, M. A., Fatima, A., Khan, M. A., & Lee, S. W. (2022). MapReduce based intelligent model for intrusion detection using machine learning technique. *Journal of King Saud University-Computer and Information Sciences*, 34(10), 9723-9731.
- [5] Singh, G., & Khare, N. (2022). A survey of intrusion detection from the perspective of intrusion datasets and machine learning techniques. *International Journal of Computers and Applications*, 44(7), 659-669.
- [6] Alhajjar, E., Maxwell, P., & Bastian, N. (2021). Adversarial machine learning in network intrusion detection systems. *Expert Systems with Applications*, 186, 115782.
- [7] Sarhan, M., Layeghy, S., Moustafa, N., & Portmann, M. (2021). Netflow datasets for machine learning-based network intrusion detection systems. In *Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings 10* (pp. 117-135). Springer International Publishing.
- [8] Kilincer, I. F., Ertam, F., & Sengur, A. (2021). Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Computer Networks*, 188, 107840.
- [9] Alzahrani, A. O., & Alenazi, M. J. (2021). Designing a network intrusion detection system based on machine learning for software defined networks. *Future Internet*, 13(5), 111.
- [10] Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1), e4150.

D. CERTIFICATE



