Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches

A Project Report submitted in the partial fulfillment of the

Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

M. Bala Rangarao (21471A0505)
B.V. Siva Rama Krishna (21471A0507)
B.V. Srinivasulu (21471A0511)

Under the esteemed guidance of

Dr. S. V. N. Sreenivasu, M.Tech., Ph.D.,

Dean R&D, Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET
(AUTONOMOUS)
Accredited by NAAC with A+ Grade and NBA under Tyre -1
NIRF rank in the band of 201-300 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada
KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name **"**Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches**"** is a bonafide work done by the team M. Bala Rangarao (21471A0505), B.V. Siva Rama Krishna (21471A0507), B.V. Srinivasulu (21471A0511) in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2024-2025.

PROJECT GUIDE
Dr. S. V. N. Sreenivasu, M.Tech., Ph.D.,
Dean R&D, Professor

PROJECT CO-ORDINATOR
Dr. Sireesha Moturi, B.Tech., M.Tech., Ph.D.
Associate Professor

HEAD OF THE DEPARTMENT
Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,
Professor& HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled "BOOSTING NETWORK INTRUSION DETECTION WITH TWO-LEVEL ENSEMBLE LEARNING AND KNOWLEDGE DISTILLATION APPROACHES" is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has been submitted for any other degree or professional qualification except as specified.

M.Bala Rangarao (21471A0505)
B.V.Siva Rama Krishna(21471A0507)
B.V.Srinivasulu (21471A0511)

ACKNOWLEDGEMENT

We wish to express my thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman sri **M. V. Koteswara Rao**, **B.Sc.**, who took keen interest in us in every effort throughout thiscourse. We owe out sincere gratitude to our beloved principal **Dr. S. Venkateswarlu,** **Ph.D.**, for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr. S. N. Tirumala Rao,** **M.Tech.**, **Ph.D.**, HOD of CSE department and also to our guide **Dr. S.V.N. Sreenivasu,** **M.Tech.**, **Ph.D.**, Dean R&D, Professor of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Dr. Sireesha Moturi,** **B.Tech**, **M.Tech.**,**Ph.D.**, Associate professor & Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing our project.

By

M. Bala Rangarao **(21471A0505)**
B.V.Siva Rama Krishna **(21471A0507)**
B.V. Srinivasulu **(21471A0511)**

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA
NEC ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA
NEC ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering

solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Project Course Outcomes (CO'S):

CO421.1: Analyze the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO 2	PS O3
C421.1		✓											✓		
C421.2	✓		✓		✓								✓		
C421.3				✓		✓	✓	✓					✓		
C421.4			✓			✓	✓	✓					✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓		✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO 2	PS O3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level
2. Medium level
3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop a model for Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches	PO1, PO3
CC421.1, C2204.3, C22L3.2	Each and every requirement is critically analyzed, the process mode is identified	PO2, PO3
CC421.2, C2204.2, C22L3.3	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5
CC421.4, C2204.4, C22L3.2	Documentation is done by all our four members in the form of a group	PO10
CC421.5, C2204.2, C22L3.3	Each and every phase of the work in group is presented periodically	PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	The project, managed by a multi-disciplinary team, enhances network security using Two-Level Ensemble Learning and Knowledge Distillation to address challenges like real-time traffic analysis, emerging threats across IoT and cloud environments.	PO4, PO7
C32SC4.3	This design integrates diverse models, knowledge distillation, and load balancing for real-time network attack detection.	PO5, PO6

ABSTRACT

An advanced Intrusion Detection System IDS framework to complement the inadequacies of the traditional methods dealing with complex and diversified network flows is proposed in this paper. The framework comprehensively solved the two traditional problems of data imbalance and accuracy detection by adopting two level ensemble learning and knowledge distillation. The proposed system is tested with the NSL-KDD dataset, fusing several machine learning models to improve overall detection performance and leveraging knowledge distillation in transferring knowledge from an advanced complex model to an easy, simple, and computationally efficient one. These results prove significant improvement regarding the detection of both common and rare high-risk attacks; hence, the proposed IDS framework is truly robust and applicable for real-time applications in state of the art network security.

INDEX

LIST OF FIGURES & TABLES

1. INTRODUCTION

Introduction

With the increasing frequency and intensity of cyberattacks affecting business and personal data, network security has become much more important in the digital age. Considering these threats, which are becoming increasingly sophisticated and voluminous, traditional IDS solutions fail to provide insight into them.

This work presents a new IDS framework based on a two-level ensemble learning approach combined with knowledge distillation. This ensembles the concept of several machine learning models within two-level ensemble learning, in order to enhance the capabilities of detection. Knowledge distillation transfers knowledge from complex, yet heavy models to simpler, more efficient ones.

The proposed algorithm is tested on the NSL-KDD dataset and overcomes the challenges of imbalanced and irrelevant data. Thus, the detection of frequent and rare attacks would be more effective.

The proposed framework includes a wide range of base models, tending from support vector machines and decision trees to gradient boosting, whose outputs will be refined by the ensemble methods, including stacking and bagging.

This framework then applies knowledge distillation to build a lightweight student model from a much more complex teacher model, achieving a good tradeoff between accuracy and computational efficiency. This boosts not only the performance of detection but also in solving the practical challenges of deployment.

It presents a review of the contributions, which include a two-level ensemble learning framework, and an extensive evaluation by several models, and releases the source codes for further research and development in network intrusion detection[9][10].

With the development of fifth-generation (5G) mobile communication technology, diverse and heterogeneous data are transmitted through distributed

networks. This increases the attack surface, necessitating improved security measures. Artificial Intelligence (AI)-based Network Intrusion Detection Systems (NIDS) are gaining traction, but data imbalance remains a challenge, affecting their ability to detect threats accurately. This study proposes an AI-based NIDS leveraging Generative Adversarial Networks (GANs) to generate synthetic data, mitigating data imbalance issues and enhancing detection capabilities[1][3].

Software-Defined Networking (SDN) is designed to simplify network management, but it introduces security challenges, such as Distributed Denial-of-Service (DDoS) attacks and unauthorized access to SDN controllers. Intruders can compromise SDN controllers, leading to catastrophic consequences. While deep learning has improved intrusion detection, challenges such as data redundancy and imbalance persist, negatively affecting detection performance. This paper introduces a hybrid Convolutional Neural Network (CNN) and bidirectional Long Short-Term Memory (BiLSTM) model to enhance intrusion detection. The model is evaluated on widely used datasets (UNSW-NB15, NSL-KDD, and InSDN), demonstrating high accuracy and reduced training time[2].

The rapid expansion of technologies like the Internet of Things (IoT), big data, and cloud computing has made network security a crucial concern. Traditional security mechanisms such as firewalls struggle to counter evolving cyber threats. Intrusion Detection Systems (IDS) have been developed to provide data integrity and prevent unauthorized network access. While early IDS implementations relied on traditional machine learning, deep learning approaches have gained popularity due to their ability to automatically extract features. This study proposes a hybrid CNN-LSTM model that leverages CNN's ability to capture spatial features and LSTM's ability to model temporal dependencies. The model is trained and evaluated using CIC-IDS 2017, UNSW-NB15, and WSN-DS datasets, achieving high detection rates with low false alarms[3].

Traditional Network Intrusion Detection Systems (NIDS) rely on full-session traffic analysis, delaying intrusion detection until a session concludes. This study proposes a packet-based intrusion detection approach using a GAN-based one-class classifier combined with an LSTM-DNN model. The proposed method allows real-time detection without requiring full session termination,

reducing detection latency while maintaining high accuracy. Experimental evaluations confirm that the approach successfully detects intrusions at early stages while preserving performance comparable to traditional session-based methods[4].

With the rapid development of the Internet, the volume of multimedia data being transmitted has significantly increased, raising concerns about network security. Traditional network intrusion detection models struggle with efficiency and accuracy due to the evolving nature of cyber threats. To address these challenges, this study proposes a novel Deformable Vision Transformer (DE-VIT) model for network intrusion detection. DE-VIT integrates a deformable attention mechanism that dynamically selects key-value pairs, enabling better feature extraction while reducing computational overhead. Additionally, the model employs deformable convolutions and a sliding window mechanism to enhance edge information processing. Experimental results on public intrusion detection datasets demonstrate that DE-VIT achieves higher accuracy and efficiency than conventional methods[5].

The continuous expansion of Internet of Things (IoT) devices across various industries has heightened the need for robust network intrusion detection systems (NIDS). Traditional intrusion detection approaches often suffer from low accuracy and long detection times. This study introduces a hybrid classifier that combines improved residual network blocks and bidirectional gated recurrent units (BiGRUs) to enhance detection performance[8]. The model first reduces feature dimensionality using an improved autoencoder, followed by classification using the hybrid network. Experimental evaluations using NSL-KDD and UNSW-NB15 datasets demonstrate that the proposed approach outperforms traditional methods, achieving higher detection accuracy while maintaining computational efficiency[6].

Wireless Local Area Networks (WLAN) face increasing security threats due to the rise in cyberattacks, necessitating real-time intrusion detection mechanisms. Traditional intrusion detection systems struggle with high false positive rates and delayed responses, making them ineffective in handling modern threats. This study proposes a Conditional Deep Belief Network (CDBN)-based intrusion detection framework that efficiently recognizes attack patterns in real time. To address data imbalance and redundancy, the study

introduces a window-based instance selection algorithm ("SamSelect") for undersampling majority-class samples and employs a Stacked Contractive Auto-Encoder (SCAE) for dimensionality reduction. Experimental results confirm that the proposed model achieves high detection speed and accuracy, outperforming traditional wireless intrusion detection systems[7].

Intrusion Detection Systems (IDS) play a critical role in identifying network attacks as quickly as possible. While Artificial Intelligence (AI) and Machine Learning (ML) have shown significant promise in research, their adoption in real-world IDS solutions remains limited. This is due to several factors, including lack of explainability, usability concerns, and regulatory compliance issues[3][7]. This study takes a user-centric approach to understanding the challenges hindering the adoption of AI/ML-based intrusion detection. By applying the concept of personas, the study systematically identifies barriers and provides design guidelines to enhance the practical implementation of AI-based NIDS solutions. The findings emphasize that improving explainability, usability, and user trust is crucial for real-world deployment[8].

With the rapid evolution of cyberattack methodologies, network security remains a significant challenge. Traditional Network Intrusion Detection Systems (NIDS) often struggle to balance efficiency and effectiveness, especially in handling large-scale, complex network data. This study proposes a tier-based optimization framework that integrates feature selection, image transformation, and deep learning classification. Initially, redundant features are removed to improve overall efficiency, and then non-image network data is transformed into image representations for deep learning-based anomaly detection. The proposed method is tested on three benchmark intrusion detection datasets and demonstrates improved detection accuracy compared to existing image-processing-based NIDS approaches[9].

Wireless networks are increasingly targeted by sophisticated cyberattacks, and traditional intrusion detection systems suffer from high false positive rates and poor adaptability[5]. This study proposes a Wireless Network Intrusion Detection System (WNIDS) based on an Improved Convolutional Neural Network (ICNN). The ICNN model enhances intrusion detection by autonomously extracting advanced traffic features, optimizing network parameters through stochastic gradient descent, and improving classification

performance. Experimental results on the KDDTest+ dataset show that the proposed ICNN model achieves higher detection accuracy and a lower false positive rate compared to conventional intrusion detection models such as LeNet-5 and Deep Belief Networks (DBN)[10].

AI-based Network Intrusion Detection Systems (NIDS) have gained significant attention, but data imbalance issues limit their performance, making it difficult to detect rare but critical attacks. This study proposes an AI-based NIDS leveraging Generative Adversarial Networks (GANs) to generate synthetic data, addressing data imbalance and improving detection accuracy. Similarly, Software-Defined Networking (SDN) simplifies network management, but its centralized nature introduces new vulnerabilities, including Distributed Denial-of-Service (DDoS) and U2R (User to Root) attacks. Many deep learning-based NIDS models suffer from issues such as data redundancy and imbalance, which can impact performance[1][6]. This study proposes a hybrid CNN-BiLSTM model for binary and multiclass intrusion detection. The model is evaluated using UNSW-NB15, NSL-KDD, and InSDN datasets, achieving high accuracy and reduced training time.

With the rise of IoT, big data, and cloud computing, network security has become a major concern. Traditional security mechanisms, such as firewalls and signature-based intrusion detection, struggle against evolving cyber threats[6]. This paper introduces a CNN-LSTM hybrid deep learning model, combining CNN's spatial feature extraction capabilities with LSTM's temporal feature modeling to improve intrusion detection. The model is trained and tested on CIC-IDS 2017, UNSW-NB15, and WSN-DS datasets, demonstrating high accuracy and low false alarm rates. Most NIDS rely on session-based traffic analysis, delaying intrusion detection until after a session ends. This study proposes a packet-based detection system that enables real-time classification using a GAN-based one-class classifier combined with an LSTM-DNN model. By identifying patterns in individual packets rather than waiting for full-session data, this approach significantly reduces detection latency while maintaining performance comparable to conventional session-based models.

2. LITERATURE SURVEY

A study introduces a GAN-based one-class classifier for early network intrusion detection, addressing the limitations of session-based analysis[1]. Traditional IDS models often rely on full-session data for classification, causing delays in detecting intrusions. By leveraging a GAN trained on misclassified data, this approach effectively distinguishes between detectable and undetectable intrusion packets, enhancing real-time detection and reducing false positives. The method ensures that intrusions are identified before session termination, making it highly suitable for real-time security applications.

Another research explores the use of a Deformable Vision Transformer (DE-VIT) for intrusion detection, enhancing feature extraction through a deformable attention mechanism[2]. Unlike conventional deep learning models, DE-VIT dynamically selects key-value positions in the attention mechanism, allowing it to focus on relevant areas while reducing computational overhead. The use of deformable convolutions further expands the receptive field, improving detection accuracy. Experimental results demonstrate superior performance on benchmark intrusion detection datasets.

A hybrid deep learning approach combining CNN and Bidirectional Long Short-Term Memory (BiLSTM) is proposed for intrusion detection in software-defined networking (SDN) environments. The study employs hybrid feature selection to optimize data representation, allowing for more effective identification of malicious network activities. The CNN component extracts spatial features, while BiLSTM captures sequential patterns in the traffic flow. This integration enhances classification accuracy while minimizing false positives, demonstrating significant improvements over traditional machine learning techniques[3].

An AI-based network intrusion detection system (NIDS) utilizing generative adversarial networks (GANs) is designed to mitigate data imbalance issues. Since certain attack types are underrepresented in real-world datasets, GANs are used to generate synthetic attack data, enhancing model training. The system also incorporates an autoencoder-based feature extraction mechanism to further refine the input data. The experimental evaluation highlights its

effectiveness in detecting rare and evolving cyber threats[4].

A real-time intrusion detection system (IDS) for wireless networks is developed using a Conditional Deep Belief Network (CDBN). This approach addresses two major challenges: data imbalance and high-dimensional redundancy[2]. The "SamSelect" algorithm is used for under-sampling the majority class, while a Stacked Contractive Auto-Encoder (SCAE) is implemented for feature reduction. The resulting IDS demonstrates high detection accuracy and low latency, making it a viable solution for real-time security monitoring in wireless environments[5].

An analysis of AI/ML-based intrusion detection highlights key barriers to its practical adoption in enterprise settings. Despite extensive research on AI-driven IDS models, real-world deployment remains limited due to issues such as explainability, usability, and compliance with regulatory frameworks. The study proposes a set of guidelines to improve the integration of AI-based security solutions, emphasizing the need for transparent, user-friendly models that align with industry requirements[6].

A tier-based optimization framework is introduced for network intrusion detection by integrating image transformation techniques with deep learning classifiers. This method converts non-image network traffic data into images, enabling the application of convolutional neural networks (CNNs) for classification[3]. By leveraging optimized feature selection and image enhancement techniques, the model achieves high detection accuracy across multiple benchmark datasets, showcasing the potential of image-based approaches in intrusion detection[7].

An improved convolutional neural network (ICNN) is developed for wireless network intrusion detection, addressing limitations of traditional CNN architectures. The model enhances feature extraction and classification efficiency, improving detection accuracy and recall rates. By optimizing network parameters using stochastic gradient descent and incorporating advanced activation functions, ICNN reduces false positives while maintaining high detection performance. This makes it a promising approach for securing wireless communication networks[8].

A hybrid model combining residual network blocks with bidirectional

gated recurrent units (BiGRU) is proposed to enhance intrusion detection capabilities. The model begins by reducing the dimensionality of network traffic data through an improved autoencoder, which refines feature representation. The BiGRU classifier then captures both spatial and temporal patterns in the data, improving classification accuracy. The study demonstrates that this hybrid approach outperforms traditional deep learning models in detecting complex cyber threats[9].

Lastly, an intrusion detection system leveraging CNN-LSTM is introduced to enhance spatial and temporal feature learning. The CNN component extracts important network features, while the LSTM module captures long-term dependencies in the traffic flow. This hybrid approach improves classification performance, particularly in detecting sophisticated and previously unseen cyber threats. Experimental results indicate robust accuracy and generalization across multiple intrusion detection datasets, making it a reliable solution for modern cybersecurity challenges[10]. This study proposes a novel intrusion detection system utilizing Generative Adversarial Networks (GANs) to enhance the performance of network security models. Traditional intrusion detection systems often suffer from imbalanced datasets, leading to poor classification performance for minority attack classes. The paper addresses this issue by leveraging GANs to generate synthetic data that improves the model's ability to detect both known and unknown attacks. The proposed system integrates deep neural networks (DNN), convolutional neural networks (CNN), and long short-term memory (LSTM) networks, achieving significant improvements in detection accuracy and recall rates. The evaluation was conducted on the NSL-KDD and UNSW-NB15 datasets, demonstrating superior performance over conventional machine learning-based intrusion detection models[1].

This paper introduces a hybrid model combining Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (BiLSTM) networks to improve the detection accuracy of network intrusion detection systems (NIDS). Software-defined networking (SDN) environments present new security challenges, necessitating more advanced detection mechanisms. The proposed model utilizes CNN for spatial feature extraction and BiLSTM for capturing temporal dependencies in network traffic. The study employed datasets like UNSW-NB15 and NSL-KDD, applying feature selection techniques such as random forest and recursive feature elimination.

Experimental results demonstrated that CNN-BiLSTM outperforms traditional models like AlexNet, LeNet5, and CNN-LSTM in terms of precision, recall, and F1-score[2].

This research integrates CNN and LSTM architectures to develop a robust intrusion detection system. CNN is used to extract spatial patterns from network traffic, while LSTM captures sequential dependencies, making it effective for anomaly detection. The model was trained and evaluated using the CIC-IDS2017, UNSW-NB15, and WSN-DS datasets, demonstrating improved accuracy and reduced false alarm rates. The results indicate that the hybrid CNN-LSTM model outperforms standalone CNN and LSTM models, achieving high detection rates and robustness against evolving cyber threats. The paper also explores techniques such as batch normalization and dropout layers to enhance model generalization[3].

This study focuses on real-time intrusion detection using a Generative Adversarial Network (GAN)-based one-class classifier. Unlike conventional methods that rely on session-based features, this approach examines individual packets, enabling earlier detection of network threats[1][3].

The model uses a GAN to filter misclassified data, enhancing the ability of the LSTM-DNN classifier to distinguish between benign and malicious traffic. By leveraging GANs for anomaly detection, the system achieves early and accurate intrusion detection without requiring full session termination. The proposed approach was validated using datasets such as ISCX2012, CIC-IDS2017, and CSE2018, demonstrating superior performance in terms of detection speed and accuracy compared to traditional session-based methods [4].

Recent advancements in artificial intelligence (AI) and machine learning (ML) have significantly influenced network intrusion detection systems (NIDS), but their practical implementation still faces critical challenges. The study by Dietz et al. (2024) highlights the gap between AI/ML-driven intrusion detection research and real-world adoption, emphasizing the need for explainability, usability, and compliance with industry regulations. The paper underscores that while AI/ML models improve detection rates, they often lack transparency, making it difficult for security administrators to trust and implement them effectively. Meanwhile, Siddiqi and Pak (2022) introduce a tier-based optimization framework for NIDS that combines feature selection

with image transformation techniques, leveraging convolutional neural networks (CNNs) to enhance detection accuracy. Their method converts network traffic data into images for deep learning-based anomaly detection, achieving high detection rates on benchmark datasets like CSE-CIC-IDS2018 and CIC-IDS2017. Similarly, Yang and Wang (2019) propose an improved convolutional neural network (ICNN) for wireless network intrusion detection, addressing issues like high false positive rates and low generalization capabilities. Their model optimizes network parameters through stochastic gradient descent, demonstrating superior accuracy compared to traditional methods such as LeNet-5 and deep belief networks. Collectively, these studies highlight the need for AI-enhanced intrusion detection frameworks that balance accuracy, efficiency, and practical usability to ensure widespread adoption in real-world cybersecurity environments.

Siddiqi and Pak (2022) introduce a tier-based optimization framework that transforms non-image network traffic data into images for CNN-based anomaly detection. By integrating feature selection and image enhancement techniques, their model achieves high detection accuracy on benchmark datasets, demonstrating the potential of deep learning in intrusion detection. Yang and Wang (2019) propose an improved convolutional neural network (ICNN) model tailored for wireless network intrusion detection. Their approach optimizes network parameters using stochastic gradient descent, reducing false positives while enhancing classification accuracy over conventional methods like deep belief networks and LeNet-5. Meanwhile, Halbouni et al. (2022) explore a hybrid deep learning model combining CNN and LSTM to leverage both spatial and temporal features in network traffic. Their model, trained on multiple datasets, achieves high accuracy and low false alarm rates, underscoring the effectiveness of deep learning in IDS. Kim and Pak (2022) focus on early detection of network intrusions using a GAN-based one-class classifier, which identifies attack patterns in real-time without waiting for full session completion. Their approach significantly reduces detection latency while maintaining high precision.

3. SYSTEM ANALYSIS

3.1 Existing System

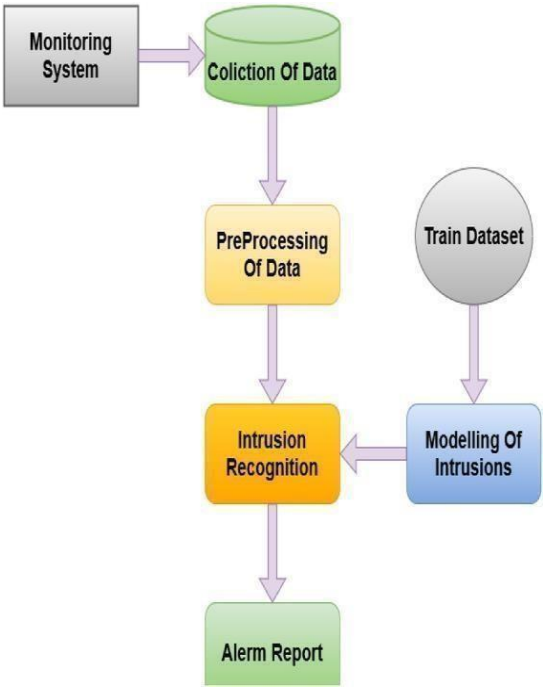

Fig 3.1.1 Flowchart of Existing System

Intrusion Detection Systems (IDS) employ various techniques to detect unauthorized access or suspicious activity. Signature-based detection compares network activity against known attack patterns, effectively identifying familiar threats but struggling with new ones. In contrast, anomaly-based detection establishes a baseline of normal behavior, flagging deviations as potential threats, which allows it to detect unknown attacks but can lead to more false positives. Heuristic and behavioral-based methods analyze user and network actions to identify unusual behavior patterns. Increasingly, machine learning (ML) techniques— including supervised and unsupervised models— are leveraged to dynamically classify and adapt to new attack patterns. Advanced IDS often use hybrid detection, combining multiple methods, such as ML with signature-based approaches, enhancing accuracy while reducing false positives. Additionally, reputation-based detection assesses traffic based on the history of IP sources, adding a layer of quick threat identification. Together, these techniques provide a robust, adaptive defense against evolving cyber threats.

3.2 Disadvantages Of Existing System

- **High Computational Cost** – The training process of GANs is computationally intensive, making real-time deployment challenging.
- **Mode Collapse** – GANs sometimes generate limited variations of attack patterns, reducing their ability to generalize to new threats.
- **Dependence on High-Quality Data** – The performance heavily relies on a well-prepared dataset, making it difficult to apply in dynamic network environments.
- **Difficulty in Training Stability** – GAN models are known for their unstable training process, which can lead to poor convergence and unreliable intrusion detection.
- **High Memory Consumption** – Transformers require significant memory for attention mechanisms, which can limit their application in resource-constrained environments.
- **Slow Processing Speed** – Despite improved accuracy, DE-VIT is slower than traditional CNN-based approaches, making it less suitable for real-time intrusion detection.
- **Overfitting to Specific Attack Patterns** – The model may perform well on known attacks but struggle with zero-day attacks due to its reliance on predefined patterns.
- **Complex Model Architecture** – The integration of CNN and LSTM increases the model's complexity, making it harder to deploy and maintain.
- **Higher Training Time** – Due to sequential processing in LSTM, the model requires longer training times compared to pure CNN-based solutions.
- **Limited Generalization to New Threats** – While effective on benchmark datasets, CNN-LSTM models may struggle with real-world data variations, requiring frequent retraining.

3.3 Proposed System

A two-level ensemble learning approach is combined with knowledge distillation techniques to enhance the efficiency and accuracy of network intrusion detection. At the first level, multiple base classifiers are deployed to capture diverse attack patterns, leveraging complementary strengths to maximize detection performance. These base classifiers' outputs are then aggregated through a secondary ensemble layer, which intelligently weighs the results to minimize false positives and detect sophisticated intrusion types. Knowledge distillation further refines this ensemble by transferring knowledge from a high-capacity teacher model to a smaller, resource-efficient student model. This enables high detection accuracy while reducing computational demands, making the system viable for real-time monitoring on large-scale networks. Through this combination, the proposed system achieves robust, adaptive intrusion detection, capable of identifying both known and emerging threats across complex network environments.

The system integrates deep learning and hybrid machine learning approaches to enhance network intrusion detection. It employs a CNN-BiLSTM model to extract both spatial and temporal features from network traffic data, improving detection accuracy. Additionally, a GAN-based one-class classifier is utilized to detect intrusions in real-time by generating synthetic samples to address data imbalance issues. The system also incorporates a tier-based optimization strategy that transforms non-image network data into images for CNN-based classification, enhancing anomaly detection. Evaluated on benchmark datasets like NSL-KDD and CIC-IDS2017, the system demonstrates high accuracy, low false positive rates, and improved detection of both common and rare attacks. These studies collectively highlight that an effective IDS should leverage deep learning, generative models, hybrid feature extraction, and real-time processing for robust cybersecurity defence.

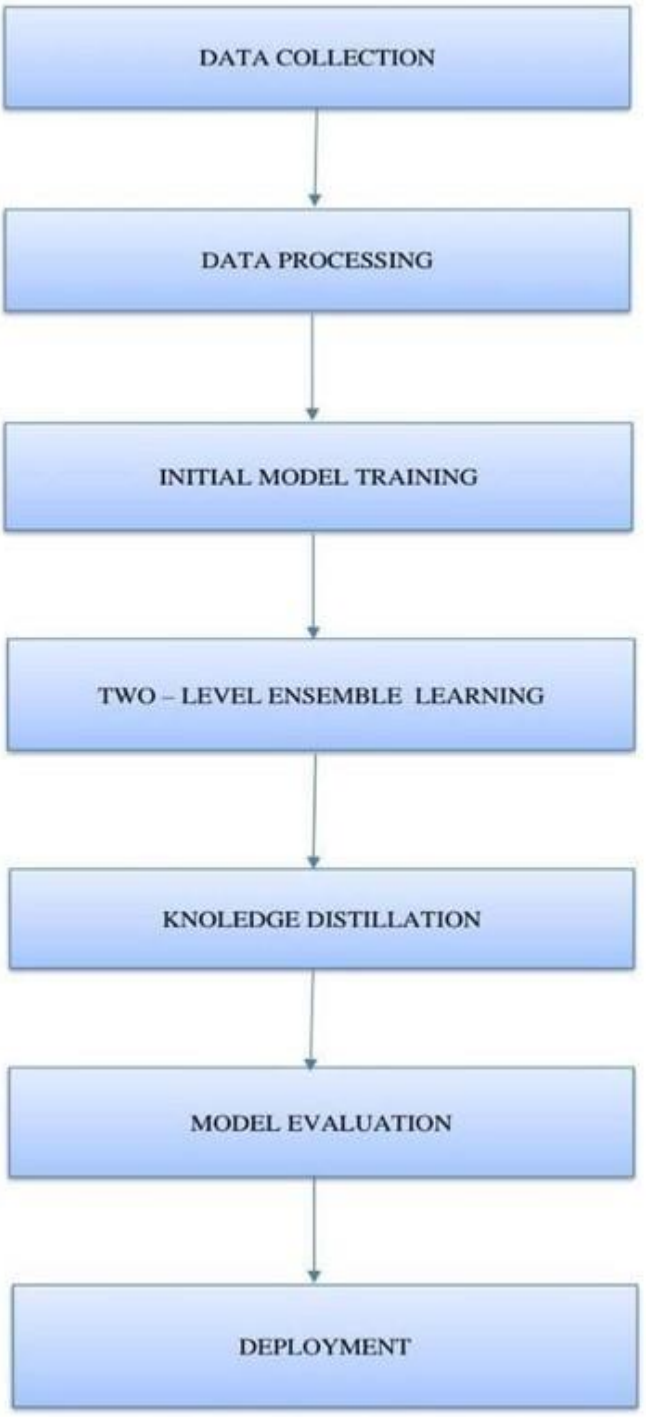

Fig 3.3.1 Flowchart of Proposed System

The flowchart in the image represents a systematic approach to developing and deploying an advanced intrusion detection system (IDS) using machine learning and deep learning techniques. The process begins with Data Collection, where network traffic data is gathered from sources such as NSL-KDD or CIC-IDS datasets. Next, Data Processing involves cleaning, normalization, and feature extraction to prepare the data for model training. In the Initial Model Training phase, baseline machine learning models are trained to establish a foundational understanding of network patterns. The Two-Level Ensemble Learning step enhances performance by combining multiple models using techniques like stacking and bagging, improving generalization and robustness. Knowledge Distillation follows, where a complex, high-performing model (teacher) transfers knowledge to a simpler, efficient model (student), optimizing computational efficiency without compromising accuracy. In the Model Evaluation stage, the system undergoes testing with real and synthetic attack scenarios to assess accuracy, false positive rates, and overall performance. Finally, in the Deployment phase, the optimized model is integrated into a real-world network environment to detect and mitigate cyber threats in real time. This approach ensures high detection accuracy, scalability, and practical usability for cybersecurity applications.

1. Data Collection:

The first step involves collecting data from various sources, such as student academic records, socio-demographic details, and performance in past courses. The dataset includes features like gender, admission scores, family background, and previous academic performance.

2. Data Preprocessing:

Once the data is collected, it undergoes a thorough preprocessing phase. This includes handling missing values, normalizing numerical data, and encoding categorical variables. Missing values are handled using imputation techniques, such as filling with mean, median, or mode, while categorical data is encoded using techniques like one-hot encoding.

3. Feature Extraction:

In this step, key features influencing student performance are extracted and selected based on their relevance. Feature engineering involves creating new

variables from existing data, such as combining scores from various subjects or considering socio-economic factors that could impact academic performance. Dimensionality reduction techniques like t-SNE (t-distributed Stochastic Neighbour Embedding) are used to reduce the complexity of high-dimensional data while preserving important information.

4. **ModelBuilding:**

   Various machine learning and deep learning models are employed to predict student performance:

- Machine Learning Models: Random Forest, XGBoost, and Support Vector Machine (SVM) are trained on the dataset. These models are effective in handling complex data and identifying key patterns associated with student success.
- Deep Learning Models: Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks are also used to model academic performance. CNNs are particularly useful for recognizing patterns in the data, while LSTMs excel at handling sequential data, such as semester-wise performance or trends over time.

5. **Model Training and Evaluation:**

The models are trained using a training dataset, and their performance is evaluated using various metrics, including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive view of how well the models are predicting student performance. A confusion matrix is used to visualize the performance of the classifiers and identify areas for improvement.

6. **Classification:**

The final step involves classifying students into different performance categories (e.g., high performers, low performers, and at-risk students) based on the predicted scores. This classification helps in early identification of students who may need academic support or intervention.[7]

7. **Results and Visualization:**

The results are analysed and presented using various visualization techniques such as bar charts, graphs, and heatmaps. The performance of different models is compared to identify the most accurate approach for predicting student success.

8. System Implementation and Integration:

The methodology is implemented in a user-friendly system that allows educators to input student data and receive performance predictions. The system is designed to be scalable and can be integrated with existing academic management systems to provide real-time performance insights.

By integrating machine learning and deep learning models, this proposed methodology aims to enhance the accuracy of academic performance predictions, leading to timely interventions and better resource allocation to improve student outcomes.

3.3.1 Data Collection and Pre-processing

Data preprocessing is a foundational step that transforms raw data into a format suitable for analysis, ensuring that the machine learning and deep learning models produce accurate and reliable results. The first step involves data cleaning, where missing values are handled through imputation or by removing rows with insufficient data, depending on the extent of the missing information. Normalization and scaling are then applied to numerical features to prevent any one feature from dominating due to differences in scale, ensuring that all features contribute equally to model performance.[2][3][4] Categorical data is converted into numerical form through techniques like one-hot encoding or label encoding, which helps the models interpret and process the data.

Additionally, outlier detection is performed to identify extreme values that could distort the results, with the outliers either being removed or adjusted to maintain data integrity. Feature selection is conducted to identify the most relevant attributes that contribute to predicting the target variable, while feature extraction techniques like principal component analysis (PCA) or t-SNE reduce dimensionality, making the dataset more manageable while retaining critical information. Data augmentation is sometimes used, especially in deep learning, to artificially expand the dataset and improve model robustness by generating new data points based on existing ones. Finally, data is split into training and testing sets to evaluate model performance accurately. These preprocessing steps ensure that the data is well-prepared, reducing noise and enhancing the models' ability to learn meaningful patterns, which ultimately improves the predictive accuracy of student performance models.

3.3.2 Feature Extraction:

Feature extraction is a critical step in the data preprocessing pipeline, particularly when working with machine learning and deep learning models. It involves transforming raw data into a set of meaningful variables (features) that can improve model performance and prediction accuracy. The objective of feature extraction is to identify the most relevant aspects of the data, reducing complexity while retaining valuable information. [2] In this study, feature extraction begins by selecting key variables that directly influence student performance, such as exam scores, attendance records, socio-demographic information (e.g., parental education level, gender), and other academic indicators. Techniques like Principal Component Analysis (PCA) or t-SNE (t-distributed Stochastic Neighbour Embedding) can be applied to reduce high-dimensional data into lower dimensions, making it easier for models to process without losing significant information.

Additionally, new features may be created through feature engineering, such as combining existing features (e.g., average scores from multiple subjects or a weighted score combining entrance exam and midterm results) to better capture the complexity of student performance. For example, socio economic status and prior academic achievements might be combined to form a new feature that better predicts future success.[7] The goal of feature extraction is to improve the accuracy and efficiency of the models by focusing on the most relevant data points and removing redundant or irrelevant features. This ensures that the models are better equipped to identify patterns and make accurate predictions. Feature extraction is particularly important when dealing with large and complex datasets, as it enables the models to work with the most informative variables while minimizing the risk of overfitting[15].

3.3.3 Machine Learning Algorithms:

In this project, several machine learning (ML) algorithms are employed to predict student performance and evaluate their effectiveness. These algorithms are chosen for their ability to handle large datasets, manage complex relationships, and provide accurate predictions. The key machine learning algorithms used are:

1. **Random Forest:**

Random Forest is an ensemble learning algorithm that builds multiple decision trees and combines their outputs to improve prediction accuracy.[7] It is particularly effective for handling high dimensional datasets with mixed data types (both numerical and categorical). Random Forest is known for its robustness, ability to avoid overfitting, and high accuracy in classification tasks, making it ideal for predicting student performance based on various features like academic scores, demographic data, and socio-economic factors.

2. **Logistic Regression**:

Logistic Regression is a statistical model used for binary classification problems, such as predicting whether a student will pass or fail. It works by estimating the probability of an event occurring, based on input features. In this project, logistic regression is used to model the likelihood of different student outcomes (e.g., success or failure) and can be an effective baseline model due to its simplicity and interpretability.[8]

3. **Support Vector Machine (SVM):**

Support Vector Machine is a powerful classification algorithm that works by finding the optimal hyperplane that best separates data points into different classes. SVM is effective in high-dimensional spaces and works well even when the data is not linearly separable. In the context of the project, SVM is used to classify students based on academic performance and predict future outcomes by creating decision boundaries between different categories of student performance.[8]

4. **K-Nearest Neighbors (KNN):**

KNN is a non-parametric, instance-based learning algorithm that makes predictions based on the similarity between data points. It classifies a student's performance by finding the 'k' nearest neighbors (students with similar features) and assigning the most common outcome among them. KNN is simple and intuitive but can be computationally expensive with large datasets. It is useful for predicting outcomes based on the closeness of a student's academic and socio-demographic features to others in the dataset.

5. **XGBoost (Extreme Gradient Boosting):**

XGBoost is a high-performance implementation of gradient boosting, an ensemble technique that combines the predictions of multiple weak models (typically decision trees) to create a stronger predictive model. XGBoost is known for its scalability, speed, and accuracy, particularly in handling large

datasets. It is effective in improving prediction accuracy by focusing on minimizing the residual errors from previous models, making it ideal for complex datasets like those used in student performance prediction.[10]

6. Gradient Boosting :

Gradient Boosting is employed in the project to enhance the accuracy of network intrusion detection by iteratively improving weak learners. It works by sequentially training decision trees, where each new tree corrects the errors made by the previous one. In the context of Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches, Gradient Boosting refines the detection of anomalies by reducing false positives and improving classification accuracy[2]. The model is particularly effective in handling imbalanced datasets, ensuring that rare but critical cyber threats are detected efficiently.

7. SKD (Knowledge Distillation) :

The Stacked Knowledge Distillation (SKD) approach is a key component of this project, designed to transfer knowledge from a complex ensemble model (teacher) to a simpler, lightweight model (student). This method allows the Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches system to maintain high detection accuracy while reducing computational overhead. The student model mimics the teacher model's behavior, making intrusion detection faster and more resource-efficient, which is crucial for real-time cybersecurity applications[7].

8. CatBoost :

CatBoost, a high-performance gradient boosting algorithm designed for categorical data, plays a vital role in improving network intrusion detection. In this project, Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches, CatBoost efficiently processes high-dimensional network traffic features and mitigates overfitting[4]. It enhances detection rates by handling categorical attributes in network logs, improving prediction stability and classification accuracy, particularly in recognizing sophisticated cyberattacks.

9. Voting :

The Voting classifier is utilized in this project to aggregate predictions from multiple base models, combining their strengths for improved intrusion detection. By integrating different machine learning classifiers, the Boosting

Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches system increases robustness and reduces model-specific biases[5]. The final prediction is determined through either hard voting (majority rule) or soft voting (weighted probability averaging), ensuring a more balanced and accurate identification of cyber threats.

10. Bag_DT :

Bagging with Decision Trees (Bag_DT) is applied in the project to create an ensemble of decision trees trained on different subsets of network traffic data. This technique enhances the reliability of Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches by reducing variance and preventing overfitting. By leveraging multiple decision trees, the model effectively identifies attack patterns while maintaining stability, leading to improved detection performance in complex network environments.

11. AdaBoost :

Adaptive Boosting (AdaBoost) is incorporated into this project to improve detection accuracy by sequentially correcting weak classifiers. In Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches, AdaBoost assigns higher weights to misclassified instances, ensuring that the model focuses on challenging attack patterns. This iterative refinement leads to a more precise identification of cyber threats, particularly in distinguishing between benign and malicious network activities.

12. Ridge Regression :

Ridge regression, a regularized linear model, is used in this project to handle multicollinearity in network traffic data. Within **Boosting** Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches, Ridge regression aids in feature selection by reducing the impact of less significant attributes while preserving important patterns in intrusion detection. This ensures that the model remains stable, particularly when processing high-dimensional network logs, leading to more reliable anomaly detection.

13. Stacking in Network Intrusion Detection

Stacking is a core component of the two-level ensemble learning framework in

this project. In Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches, multiple base models such as Gradient Boosting, CatBoost, and AdaBoost are trained, and their outputs are combined using a meta-learner. This hierarchical structure improves the system's predictive power by capturing diverse attack patterns and reducing false positives, ensuring a more comprehensive intrusion detection mechanism[9].

14. Bagging in Network Intrusion Detection

Bagging (Bootstrap Aggregating) is employed to enhance model stability and generalization in this project. In Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches, Bagging creates multiple instances of base models trained on different data subsets, reducing variance and preventing overfitting. This approach ensures that the IDS remains robust in detecting both known and emerging cyber threats, improving reliability in dynamic network environments[8].

3.3.4 Performance Metrics (Accuracy, precision, Recall, F1-Score)

Accuracy

Accuracy measures the proportion of correctly predicted instances (true positives and true negatives) out of the total predictions.

Accuracy=TP+TN/TP+FP+TN+FN

Precision

Precision evaluates the proportion of true positive predictions among all positive predictions made by the model. It highlights how precise or accurate the model is in predicting positive cases.

Precision=TP/TP+FP

Recall

Recall measures the proportion of actual positive cases that the model successfully identified. It is important when the focus is on minimizing false negatives.

Recall=TP/TP+FN F1

Score

The F1 score is the harmonic mean of precision and recall. It provides a balanced metric that accounts for both false positives and false negatives,

especially useful when dealing with imbalanced datasets

F1-Score=2·(Precision Recall/Precision Recall)

3.4 Feasibility Study

1. Economical Feasibility

The economical feasibility of the project determines whether the implementation is cost-effective. The project uses Google Colab Pro for model training and development, which is a cloud-based platform that eliminates the need for high-end local hardware.

Estimated Costs:

- Google Colab Pro Subscription: ₹1,093/month ($9.99/month)
- Flask Development (Frontend & Backend Integration): No additional cost as Flask is open-source.

Since Google Colab Pro provides GPU/TPU access, the costs remain lower compared to purchasing high-end GPUs for local training. Overall, the project is economically viable due to its reliance on free and affordable cloud-based tools.

2. Technical Feasibility

Technical feasibility evaluates the technologies and programming languages used to implement the project.

Technologies & Tools Used:

- Programming Languages: Python (Machine Learning & Deep Learning models)
- Machine Learning Models: Random Forest, XGBoost, , LSTM, SVM, Logistic Regression, Gradient Boosting, SKD (Stacked Knowledge Distillation), CatBoost, Voting, Bagging with Decision Trees (Bag_DT), AdaBoost, Ridge Regression, Stacking, Bagging.
- Development Framework: Flask (Used for the frontend and backend integration)
- Cloud Computing: Google Colab Pro (for model training and execution)

4. SYSTEM REQUIREMENTS

A machine learning project requires specific hardware and software configurations to ensure efficient data processing, model training, and deployment. Below is a detailed explanation of each requirement: Key Features of Google Colab

4.1 SOFTWARE REQUIREMENTS

1.Operating System: Windows 11, 64-bit

- A modern OS with enhanced security, efficiency, compatibility with ML tools.
- The 64-bit architecture has high-performance computing, large memory usage.

2.Coding Language: Python

- Python is widely used for ML due to its rich ecosystem of libraries (TensorFlow,Scikit-learn, Pandas, NumPy).
- It provides easy syntax, rapid development, and extensive community support.

3.Python Distribution: Google Colab Pro, Flask

- Google Colab Pro: A cloud-based Jupyter notebook environment with GPU/TPU support for faster model training.
- Flask: A lightweight web framework used to deploy and serve ML models as APIs.

4.Browser: Any Latest Browser (e.g., Chrome)

- A modern web browser ensures compatibility with Google Colab, Flask applications, and cloud-based tools.
- Google Chrome is preferred for its performance, developer tools, and security features.

4.2 Hardware Requirements

1.System Type: Intel® Core™ i5-7500U CPU @ 2.40GHz

- The Intel Core i5-7500U is a dual-core processor suitable for ML tasks such as data preprocessing and small-scale model training.

- It operates at 2.40 GHz, providing a balance of speed and power efficiency.

2.Cache Memory: 4MB (Megabyte)

- Cache memory stores frequently accessed data, reducing latency in computations.
- A 4MB cache helps in improving data retrieval speed, benefiting real-time processing.

3.RAM: Minimum 8GB (Gigabyte)

- RAM (Random Access Memory) allows temporary data storage for active processes.
- 8GB RAM is the minimum requirement to handle ML libraries, datasets, and computations without excessive lag.

4.Hard Disk: 229GB

- Storage is essential for datasets, trained models, and software dependencies.
- A 229GB hard disk provides sufficient space for project files, experiment logs.

5.Computer Engine: T4 GPU

- NVIDIA T4 GPU is designed for AI and deep learning applications.
- It accelerates training, inference tasks, reducing process time significantly.

4.3 Software Description

1.**Python (3.x):**

Python is an interpreted, high-level programming language widely used for web development, data science, and automation. It provides a simple syntax and an extensive set of libraries, making it a preferred choice for Flask-based applications. The Flask framework runs on Python, and essential libraries like Flask-SQLAlchemy and Flask-Bcrypt require it. To ensure compatibility, install Python 3.x . After installation, verify it using python --version. Using a virtual environment (venv) is recommended for dependency management. Python's built-in SQLite database also eliminates the need for additional database installations in small-scale applications.

2.pip (Python Package Manager):

pip is the standard package manager for Python, enabling easy installation, upgrading, and removal of libraries. It ensures that dependencies such as Flask, Flask-SQLAlchemy, Flask-Bcrypt, NumPy, and Pickle5 can be installed efficiently. It is included with Python installations by default but can be updated using pip install --upgrade pip. The requirements.txt file allows bulk installations, making pip an essential tool for managing Flask applications. Running pip list displays installed packages, while pip freeze > requirements.txt saves dependencies for future use. Without pip, manually managing dependencies would be time-consuming and error-prone.

3.Flask:

Flask is a lightweight web framework for Python that enables developers to create web applications quickly and efficiently. Unlike Django, Flask is minimalistic and follows a modular design, allowing developers to integrate only necessary components. It provides built-in support for routing, request handling, and templates, making it ideal for applications like user authentication and machine learning-based predictions. Flask's simplicity allows developers to build scalable applications while maintaining control over database configurations and security measures. With a growing ecosystem and extensive documentation, Flask is an excellent choice for both beginners and experienced developers building web-based applications.

4.Flask-SQLAlchemy:

Flask-SQLAlchemy is an Object-Relational Mapping (ORM) tool that simplifies database interactions in Flask applications. Instead of writing raw SQL queries, developers can define Python classes to represent database tables, making code more readable and maintainable. It supports multiple databases, including SQLite, MySQL, and PostgreSQL. Using Flask-SQLAlchemy, developers can perform CRUD (Create, Read, Update, Delete) operations effortlessly. It also provides features like connection pooling and session management, enhancing database efficiency. Since SQLite is the default database for this project, Flask-SQLAlchemy is necessary for managing user authentication and storing machine learning-related data securely.

5.Flask-Bcrypt:

Flask-Bcrypt is a Flask extension that provides password hashing capabilities using the Bcrypt hashing algorithm. Since storing plain-text passwords is a

security risk, Flask-Bcrypt ensures that passwords are securely hashed before storing them in the database This library enhances application security, ensuring that user credentials remain protected even in case of database leaks. It is a crucial component for implementing user authentication in Flask applications.

6. NumPy:

NumPy is a powerful library for numerical computing in Python, widely used for handling large datasets, performing mathematical operations, and working with multi-dimensional arrays. In this Flask project, NumPy is essential for pre-processing input data before making predictions with the machine learning model. It ensures that the user's input values are converted into NumPy arrays before being passed to the model.predict() function. This improves efficiency and ensures compatibility with machine learning models. NumPy's optimized array operations significantly boost performance, making it an indispensable tool for applications involving numerical data processing and machine learning predictions.

7. Pickle5:

Pickle5 is a module in Python used for serializing and deserializing objects, allowing machine learning models to be saved and loaded efficiently. In this project, the trained model is stored in model.pkl, which is loaded using pickle.load(open('model.pkl', 'rb')). This eliminates the need to retrain models every time the application starts, significantly reducing processing time. Pickle5 ensures seamless model integration into Flask applications, allowing real-time predictions from previously trained models. Since different Python versions affect pickled files, using Pickle5 ensures better compatibility, making it crucial for deploying machine learning models in web applications.

8. SQLite:

SQLite is a lightweight, serverless relational database that is built into Python, making it ideal for small-scale applications. It does not require additional installation, reducing setup complexity. SQLite stores data in a single .db file, allowing for quick and easy access. In this Flask project, SQLite is used to manage user accounts, storing credentials securely with Flask-SQLAlchemy. It supports standard SQL queries, making it a convenient choice for handling authentication.

5. SYSTEM DESIGN

A robust Network Intrusion Detection System (NIDS) leveraging Two-Level Ensemble Learning and Knowledge Distillation offers significant advantages. It begins by collecting and preprocessing network traffic data, extracting relevant features, and handling class imbalances. Multiple diverse machine learning models, such as Random Forest, XGBoost, LightGBM, and CatBoost, are trained on this data. The predictions from these models are then combined using techniques like averaging or voting to improve accuracy and robustness. To further enhance performance, a knowledge distillation technique is employed, where a powerful teacher model transfers its knowledge to a smaller, more efficient student model. The trained models are deployed in a production environment, often using a load balancer to distribute incoming network traffic across multiple prediction servers. This ensures efficient resource utilization and real-time analysis of network traffic. By combining these techniques, the NIDS can effectively detect a wide range of network attacks, including zero-day attacks, and provide timely alerts to security administrators.

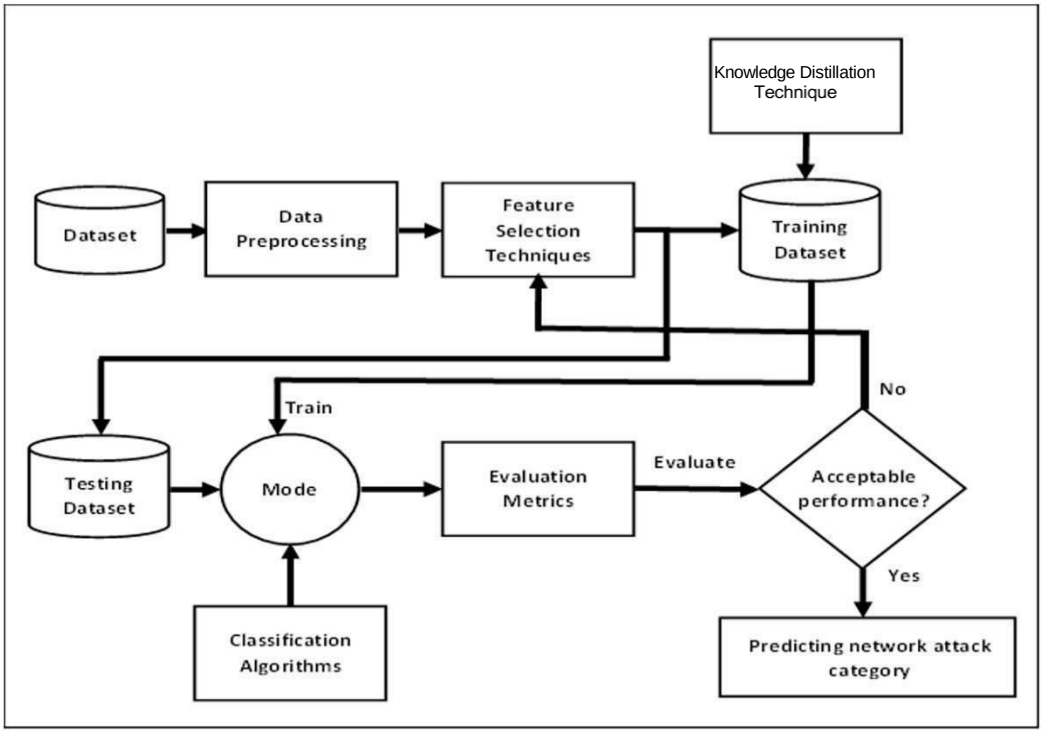

Fig 5.1 Design Overview

5.1 System Architecture

Dataset

The dataset presented in the image is NSL-KDD, a widely used benchmark dataset for network intrusion detection. The dataset contains 5 labels (attack categories), 41 features, and a total of 148,517 samples.

In terms of attack distribution, the dataset includes 48% normal traffic, meaning nearly half of the samples represent benign network activity. Among the various attack types, 10.05% belong to the "Probe" category, which consists of scanning and reconnaissance attacks. U2R (User-to-Root) attacks make up 0.47%, indicating rare but severe privilege escalation attempts. Additionally, R2L (Remote-to-Local) attacks constitute 6.48%, representing unauthorized access attempts from remote machines. Other attack categories like DoS (Denial of Service), PortScan, Brute Force, Web, and Bot attacks are not explicitly mentioned in this subset of the dataset, and the Infiltration category is listed but without specific details on its proportion.

Overall, this dataset is structured to facilitate machine learning-based intrusion detection, with a diverse distribution of attack types, ensuring a balanced evaluation of detection models.

(a)Basic statistics of dataset

Dataset	No.of Labels	No.of Features	No.of Samples
NSL-KDD	5	41	148,517

(b) Distribution of samples among differen attach types

Dataset	Normal	DoS	PortScan	Brute Force	Web	Bot	Infilt.	Probe	U2R	R2L
NSL-KDD	48%	-	-	-	-	-	-	10.05%	0.47%	6.48%

Table 5.1.1 Dataset Description

Pre-processing

The preprocessing of the NSL-KDD dataset involved several crucial steps to ensure the data was clean, well-structured, and ready for model training. Initially, the training and testing datasets were loaded and the columns were appropriately renamed to reflect the features within the dataset, such as duration, protocol-type, and service. Following this, a comprehensive check was conducted to ensure that no missing values were present in the datasets. Any duplicate entries were removed to avoid redundancy and potential bias during the model training process.

To handle categorical features like protocol-type, service, and flag, one-hot encoding was applied, converting these categories into binary vectors suitable for machine learning

algorithms. The target variable attack was also encoded into numerical values using LabelEncoder, which facilitated the classification process. The datasets were then split into features and labels, with further division into training and testing subsets to ensure robust validation of the model's performance.

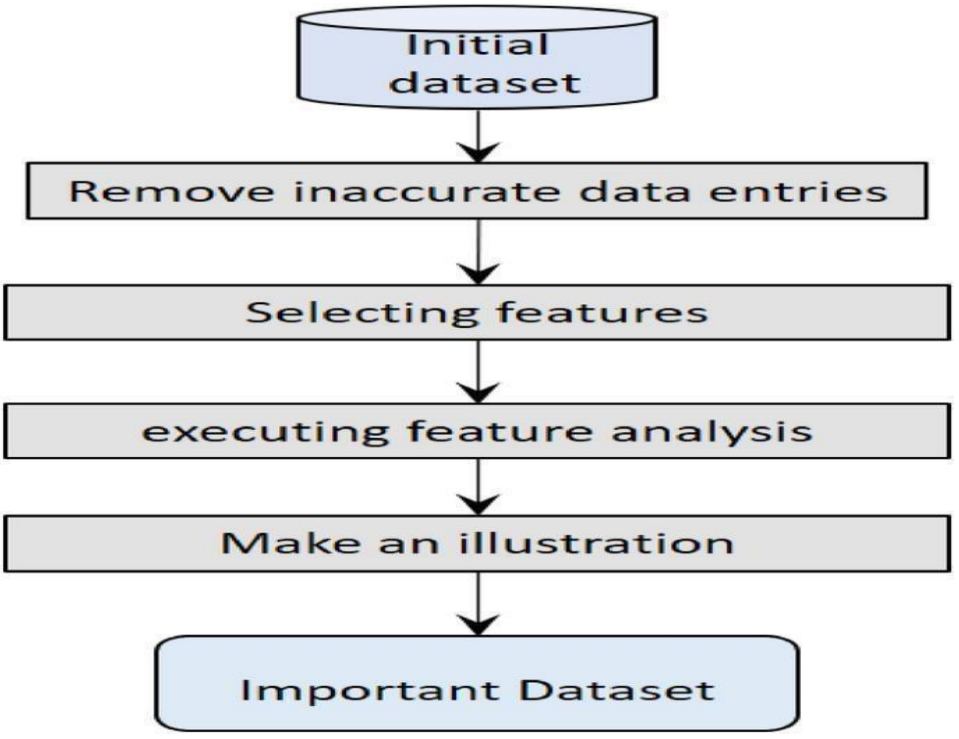

Fig 5.1.1 Data Pre-processing

Preprocessing is a crucial step in preparing the NSL-KDD dataset for machine learning-based intrusion detection. It involves multiple stages, including data cleaning, feature encoding, normalization, and splitting the dataset for training and testing.

Data Cleaning

- Duplicate and irrelevant records are removed to ensure data consistency.
- Missing or corrupted values, if present, are handled by either filling them with appropriate values or removing affected rows.

Feature Encoding

- The dataset contains categorical features (e.g., protocol type, service, and flag). These categorical attributes are converted into numerical values using techniques like One-Hot Encoding or Label Encoding to make them compatible with machine learning models.

Feature Scaling & Normalization

- Numerical features are normalized using Min-Max Scaling or Standardization to bring all features within a uniform range, ensuring that models do not assign undue importance to larger numerical values.

Handling Imbalanced Data

- Since attack categories may be imbalanced, techniques like oversampling (SMOTE - Synthetic Minority Over-sampling Technique) or under sampling can be applied to balance the dataset and prevent bias toward majority classes.

Splitting the Dataset

- The dataset is typically divided into training (70-80%) and testing (20-30%) subsets to evaluate model performance effectively.
- A separate validation set may be used for hyperparameter tuning.

Feature Selection & Dimensionality Reduction

- Unimportant or redundant features may be removed using Principal Component Analysis (PCA) or feature importance techniques (e.g., using Random Forest or Recursive Feature Elimination) to improve computational efficiency.
- This preprocessing pipeline ensures that the dataset is well-structured, standardized, and optimized for building accurate and efficient intrusion detection models.

Model

Model building in the context of deep learning refers to the process of designing and constructing neural network architectures to solve specific tasks such as classification, regression, or generation. Deep learning models typically consist of multiple layers of neurons organized in a hierarchical fashion, enabling the model to learn intricate patterns and representations from the data. The below image illustrates a two-level ensemble learning framework for network intrusion detection using the NSL-KDD dataset. The process begins at Dataset Level 00, where all features from the NSL-KDD dataset undergo feature selection before passed into various base machine learning models, including Decision Tree (DT), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Logistic Regression (LR), Multi-Layer Perceptron (MLP), and Deep Neural Network (DNN). The framework employs Boosting models such as AdaBoost (ADA), CatBoost (CAT), XGBoost (XGB), and LightGBM (LGBM), along with Bagging methods like Random Forest (RF) for improving model performance. Additionally, Stacking techniques such as voting, averaging, and weighted averaging are used to combine multiple classifiers to enhance detection accuracy. The final model is evaluated using performance metrics like accuracy, precision, recall, and F1-score. This hierarchical learning approach ensures a robust and efficient intrusion detection system by leveraging multiple classification techniques and ensemble strategies.

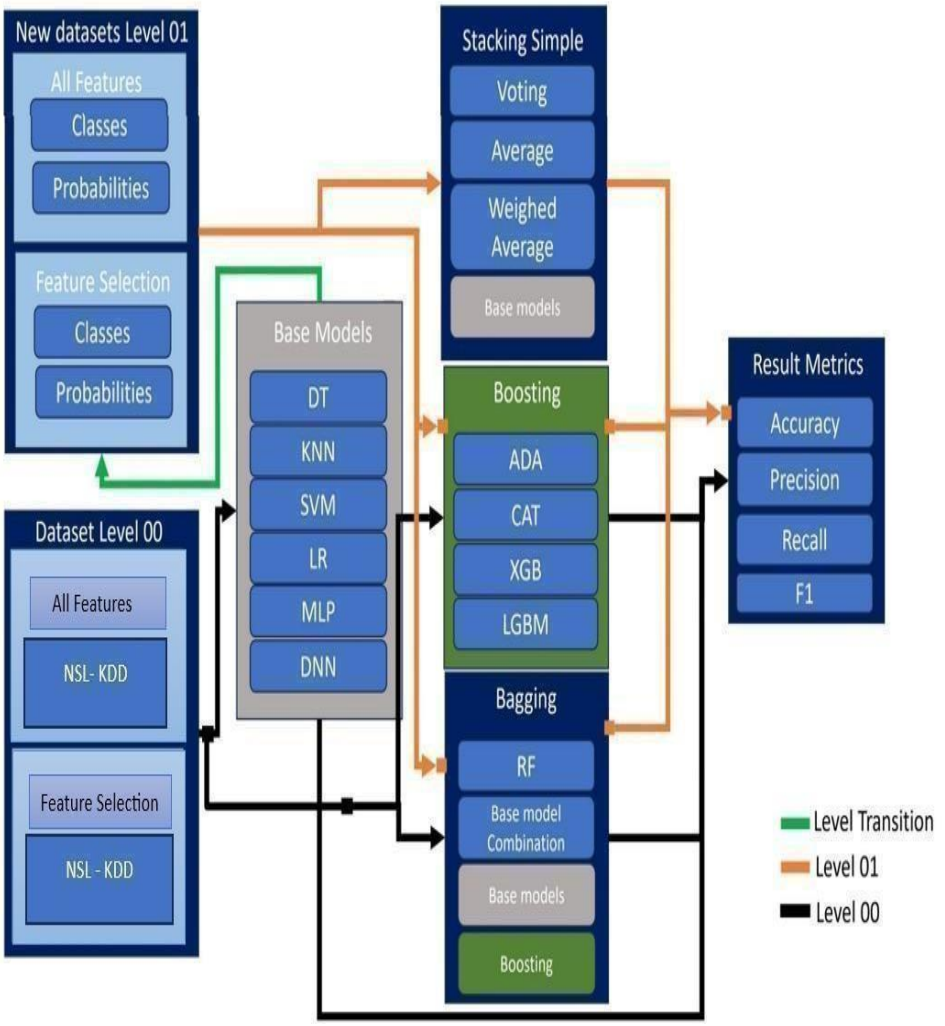

Fig 5.1.2 Overview of the Model

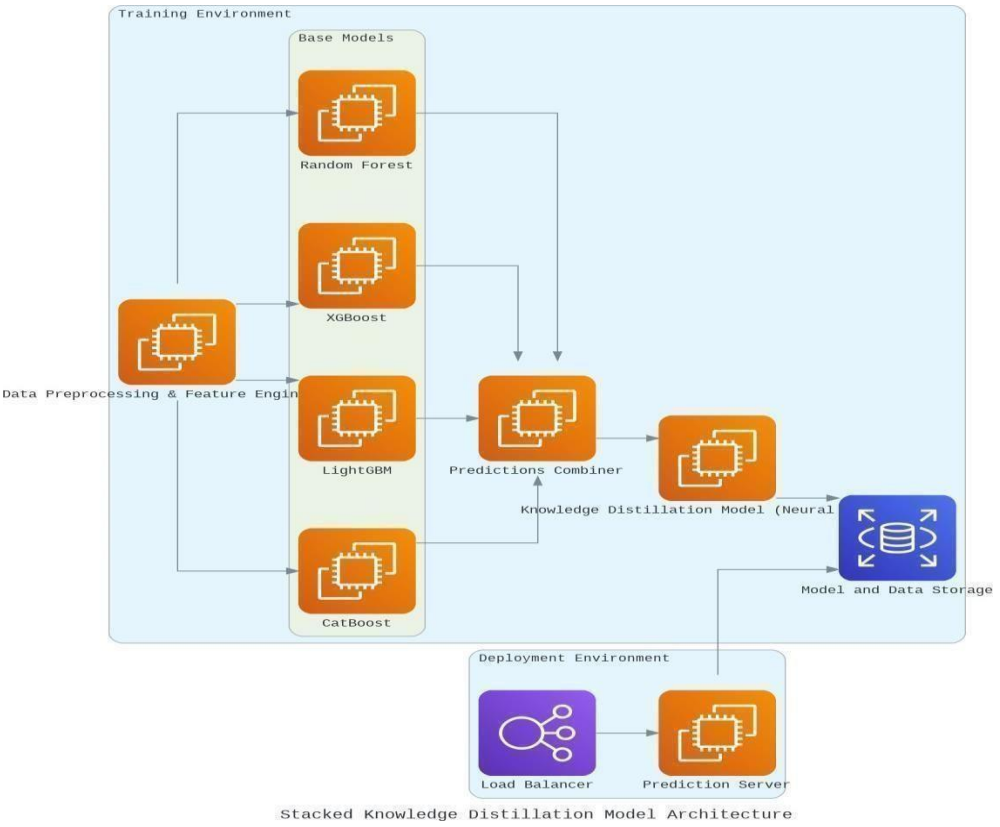

Fig 5.1.3 Model Architecture

Layers in Stacked Knowledge Distillation Model Architecture

1. Data Processing and Future Engine Layer: The Data Processing and Future Engine Layer is the initial layer in the Stacked Knowledge Distillation Model Architecture, responsible for preprocessing raw data and extracting relevant features to prepare it for subsequent layers.

2. Base Model Layer: The Base Models layer comprises four powerful machine learning algorithms: Random Forest, XGBoost, LightGBM, and CatBoost. These models independently learn from the preprocessed data, generating diverse predictions that are subsequently combined in the ensemble layer.

3. Predictions Combiner Layer: The Prediction Combiner layer takes the individual predictions from the Base Models layer and combines them using techniques like averaging, weighted averaging, or voting to produce a more accurate and robust final prediction.

4. Knowledge Distillation Layer: The Knowledge Distillation layer in Network Intrusion Detection Systems (NIDS) leverages a powerful teacher

model to train a smaller, more efficient student model. The teacher model transfers its knowledge to the student model, improving its accuracy and reducing inference time. This layer enhances the NIDS's ability to detect complex attacks while maintaining real-time performance.

5. Model And Data Storage: The Model and Data Storage layer stores the trained models and historical data for future reference and analysis. This layer ensures model persistence, enables retraining, and facilitates continuous learning and improvement of the NIDS.

6. Deployment Layer: The Deployment Environment is the final layer in the NIDS architecture. It consists of a Load Balancer and Prediction Servers. The Load Balancer distributes incoming network traffic across multiple Prediction Servers for efficient processing. This ensures optimal resource utilization and high system availability, enabling real-time analysis and detection of network intrusions. The Prediction Servers utilize the trained models to analyze network traffic and generate alerts for potential threats. This layer is crucial for the operational deployment of the NIDS, ensuring its effectiveness in protecting networks from cyberattacks.

5.2 Modules

Preprocessing Module

The preprocessing module of the proposed IDS framework involves several crucial steps to ensure the NSL-KDD dataset is well-prepared for machine learning models. It starts with cleaning and structuring the data, followed by naming columns and checking for missing values. Irrelevant records are filtered out to enhance data quality. Categorical features are encoded using one-hot encoding, while the target variable undergoes label encoding to standardize classification tasks. The dataset is then split into training and testing subsets to facilitate model evaluation. Finally, feature normalization is applied using RobustScaler to mitigate the impact of outliers, ensuring a more stable learning process. This preprocessing pipeline ensures that the data is optimally structured for training and testing, ultimately improving the model’s detection performance.

User Interface (UI) Module

- The base paper does not explicitly mention a dedicated user interface (UI) module for the proposed IDS framework.
- Based on the described methodology, an effective UI module would likely serve as a visualization and interaction layer for security analysts.
- It would display intrusion detection results, including real-time threat alerts, classification reports, and performance metrics such as accuracy, precision, recall, and F1-score.
- The UI could feature interactive dashboards with graphical representations of network traffic anomalies, intrusion trends, and classification probabilities derived from the ensemble learning and knowledge distillation models.
- Additionally, it would allow users to configure model parameters, upload new datasets for analysis, and interpret explainable AI (XAI)-based insights.
- The inclusion of such a UI would enhance usability, enabling seamless monitoring and decision-making for network security professionals.

System Module

- The system module of the proposed IDS framework is structured around a two-level ensemble learning approach combined with knowledge distillation to enhance network intrusion detection.
- The framework processes the NSL-KDD dataset, beginning with data preprocessing, which includes cleaning, encoding, and normalization.
- The first level (Level 00) consists of multiple base machine learning models, such as Decision Trees (DT), Support Vector Machines (SVM), Deep Neural Networks (DNN), and Gradient Boosting models, which analyze network traffic data.

- The second level (Level 01) aggregates predictions from Level 00 using stacking and bagging ensemble techniques to improve classification accuracy. Additionally, a knowledge distillation module transfers insights from a complex teacher model (e.g., XGBoost, CatBoost, or Random Forest) to a lightweight student model, ensuring computational efficiency without compromising accuracy.
- The entire system operates within a high-performance computing environment, leveraging Python-based machine learning libraries, including TensorFlow and PyTorch, to facilitate scalable and real-time intrusion detection.

5.3 UML Diagrams

The image represents a Network Intrusion Detection system architecture, illustrating the interaction between the User and the System across various stages of the intrusion detection process. The framework follows a structured pipeline, starting with Data Collection, where network traffic data is gathered for analysis. This is followed by Data Preprocessing, which involves cleaning, encoding, and feature selection to prepare the data for model training. In the Base Model Training phase, multiple machine learning models are trained to identify potential intrusions. These models are then combined using Ensemble Learning, which enhances detection accuracy by aggregating predictions from various classifiers. Additionally, Knowledge Distillation is employed to transfer insights from complex models to lightweight models, optimizing performance while maintaining accuracy. The system further includes a Model Evaluation phase, where the trained models are assessed using performance metrics such as accuracy, precision, and recall. Once validated, the model proceeds to Deployment, enabling real-time network intrusion detection. Lastly, the Continuous Learning & Improvement phase ensures that the system evolves by learning from new data, refining its detection capabilities over time. The diagram highlights how the user actively participates in data collection, model evaluation, deployment, and continuous learning, while the system autonomously handles model training, ensemble learning, and knowledge distillation, ensuring an efficient and adaptive intrusion detection mechanism.

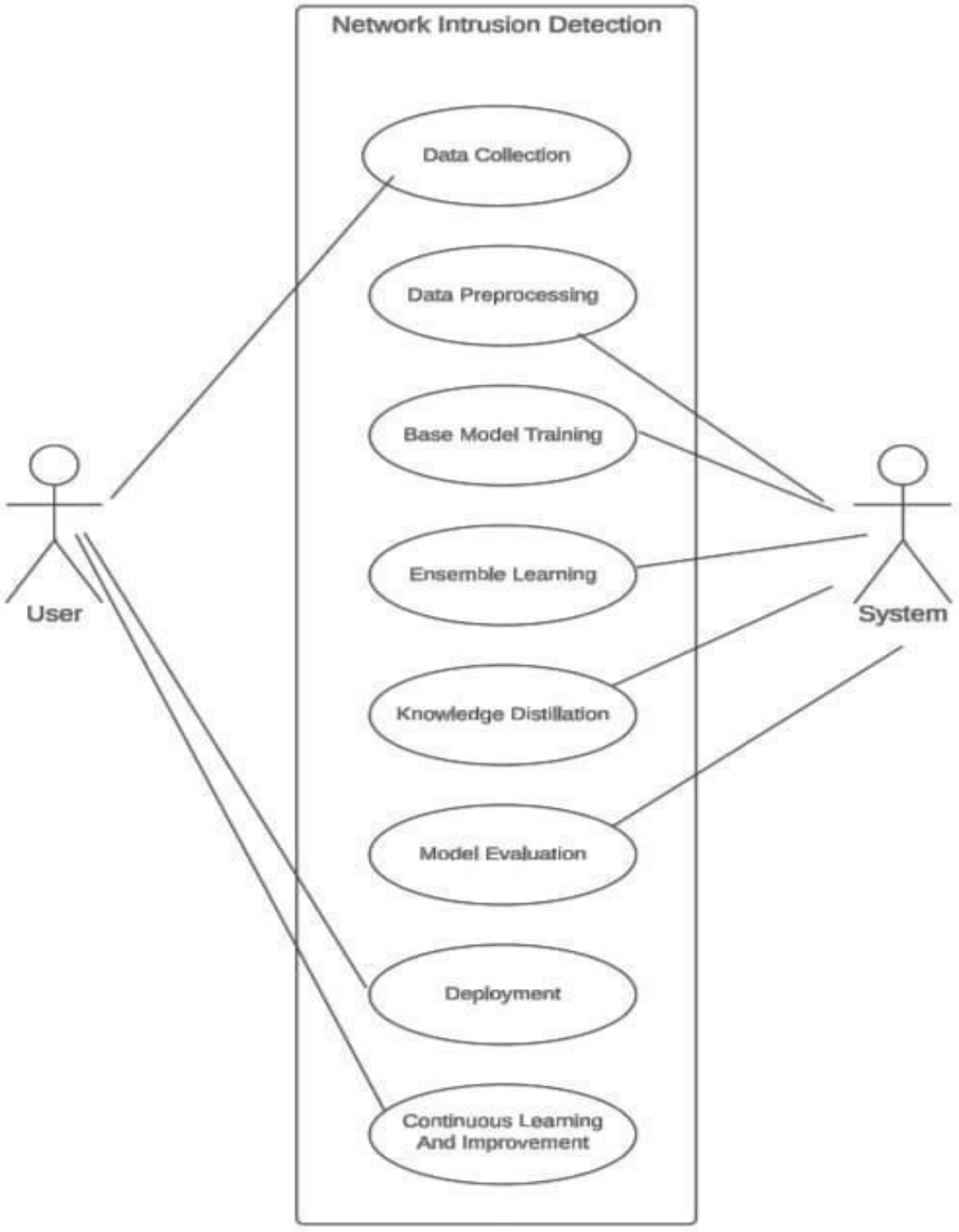

Fig 5.3.1 Use Case Diagram

6. IMPLEMENTATION

6.1 Model Implementation

#Random Forest Model

```
max_depth= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Parameters={ 'max_depth': max_depth}
RF= RandomForestClassifier()
GridSearch(RF, Parameters, X_train_train, Y_train_train)
RF.fit(X_train_train, Y_train_train)
RF.score(X_train_train, Y_train_train), RF.score(X_test_train, Y_test_train)
Evaluate('Random Forest Classifier', RF, X_test_train, Y_test_train)

#  the random forest implementation archecture in the above model

from sklearn import tree
import matplotlib.pyplot as plt

# Assuming 'RF' is your trained RandomForestClassifier
estimator = RF.estimators_[0]  # Extract one of the decision trees from the
forest

fig = plt.figure(figsize=(15, 12))
tree.plot_tree(estimator, filled=True)
plt.show()
```

#Gradient Boosting Model

```
# Instantiate and fit the Gradient Boosting Classifier
GB = GradientBoostingClassifier()
GB.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("Gradient Boosting Classifier Training Score:", GB.score(X_train_train,
Y_train_train))
print("Gradient Boosting Classifier Testing Score:", GB.score(X_test_train,
Y_test_train))
# Evaluate the Gradient Boosting Classifier
Evaluate('Gradient Boosting Classifier', GB, X_test_train, Y_test_train)
# a diagram for the gradient boosting implementation archecture in the above
model

!pip install graphviz
```

```
!pip install pydotplus

from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image

# Assuming 'GB' is your trained GradientBoostingClassifier
estimator = GB.estimators_[0][0]  # Extract one of the decision trees from the
first iteration

dot_data =
export_graphviz(estimator,filled=True, rounded=True, special_characters=Tru
e)

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

#XGBoost Model

```
!pip install graphviz
!pip install pydotplus
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image
# Assuming 'GB' is your trained GradientBoostingClassifier
estimator = GB.estimators_[0][0]  # Extract one of the decision trees from the
first iteration
dot_data =
export_graphviz(estimator,filled=True,  rounded=True,special_characters=Tru
e)

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
# Instantiate and fit the XGBoost Classifier
XGB = xgb.XGBClassifier(n_estimators=100, random_state=42)
XGB.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("XGBoost Classifier Training Score:", XGB.score(X_train_train,
Y_train_train))
print("XGBoost Classifier Testing Score:", XGB.score(X_test_train,
Y_test_train))
# Evaluate the XGBoost Classifier
Evaluate('XGBoost Classifier', XGB, X_test_train, Y_test_train)
```

#CatBoost Model

```
# Instantiate and fit the CatBoost Classifier
CatBoost =
CatBoostClassifier(learning_rate=0.1,depth=6,n_estimators=100,l2_leaf_reg=3
,random_state=42)
CatBoost.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("CatBoost Classifier Training Score:", CatBoost.score(X_train_train,
Y_train_train))
print("CatBoost Classifier Testing Score:", CatBoost.score(X_test_train,
Y_test_train))
# Evaluate the CatBoost Classifier
Evaluate('CatBoost Classifier', CatBoost, X_test_train, Y_test_train)
```

#AdaBoost Model

```
# Instantiate the base estimator (DecisionTreeClassifier)
base_estimator = DecisionTreeClassifier(max_features=6, max_depth=4)
# Instantiate the AdaBoost Classifier with adjusted parameters
AdaBoost =
AdaBoostClassifier(base_estimator=base_estimator,n_estimators=100,learning
_rate=1.0)
AdaBoost.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("AdaBoost Classifier Training Score:", AdaBoost.score(X_train_train,
Y_train_train))
print("AdaBoost Classifier Testing Score:", AdaBoost.score(X_test_train,
Y_test_train))
# Evaluate the AdaBoost Classifier
Evaluate('AdaBoost Classifier', AdaBoost, X_test_train, Y_test_train)
#  a diagram for the adaboost implementation archecture in the above model

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from IPython.display import Image, display
import pydotplus
from sklearn import tree

# Sample data (replace with your actual data)
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Instantiate the base estimator (DecisionTreeClassifier)
base_estimator = DecisionTreeClassifier(max_features=6, max_depth=4)

# Instantiate the AdaBoost Classifier with adjusted parameters
AdaBoost = AdaBoostClassifier(base_estimator=base_estimator,
n_estimators=100, learning_rate=1.0)
AdaBoost.fit(X_train, y_train)

# Visualize the first decision tree in the AdaBoost ensemble
estimator = AdaBoost.estimators_[0]
dot_data =
tree.export_graphviz(estimator,filled=True,rounded=True,special_characters=
True)

graph = pydotplus.graph_from_dot_data(dot_data)
display(Image(graph.create_png()))
```

#Ridge Classifier Model

```
# Create RidgeClassifier
ridge_classifier = RidgeClassifier(alpha=1.0)
# Wrap RidgeClassifier inside CalibratedClassifierCV with method='sigmoid'
for Platt scaling
Platt_ridge = CalibratedClassifierCV(ridge_classifier, method='sigmoid')
# Fit the classifier
Platt_ridge.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("Ridge Classifier Training Score:", Platt_ridge.score(X_train_train,
Y_train_train))
print("Ridge Classifier Testing Score:", Platt_ridge.score(X_test_train,
Y_test_train))
# Evaluate the Ridge Classifier
Evaluate('Ridge Classifier', Platt_ridge, X_test_train, Y_test_train)
#  diagram for the ridge classifier implementation archecture in the above
model

!pip install diagrams

from diagrams import Diagram, Cluster
from diagrams.aws.compute import EC2
from diagrams.aws.database import RDS
from diagrams.aws.network import ELB

with Diagram("Ridge Classifier Architecture", show=False) as diag:
```

```
    with Cluster("Training Environment"):
        data_prep = EC2("Data Preprocessing & Feature Engineering")
        model_training = EC2("Ridge Classifier Training")

    with Cluster("Deployment Environment"):
        load_balancer = ELB("Load Balancer")
        prediction_server = EC2("Prediction Server")
        database = RDS("Model and Data Storage")
data_prep >> model_training >> database
    load_balancer >> prediction_server >> database

diag
```

#Bagging Model

```
#  code for bagging model for the above data
from sklearn.ensemble import BaggingClassifier
# Instantiate the base estimator (e.g., DecisionTreeClassifier)
base_estimator = DecisionTreeClassifier(max_depth=10)
# Instantiate the Bagging Classifier
bagging_clf = BaggingClassifier(base_estimator=base_estimator,
n_estimators=100, random_state=42)
# Fit the Bagging Classifier
bagging_clf.fit(X_train_train, Y_train_train)
# Evaluate Bagging Model
Evaluate('Bagging Classifier', bagging_clf, X_test_train, Y_test_train)
#  diagram for the bagging implementation archecture in the above model

from diagrams import Diagram, Cluster
from diagrams.aws.compute import EC2
from diagrams.aws.network import ELB
from diagrams.aws.database import RDS

with Diagram("Bagging Model Architecture", show=False) as diag:
    with Cluster("Training Environment"):
        data_prep = EC2("Data Preprocessing & Feature Engineering")

        with Cluster("Bagging Ensemble"):
     multiple_base_models = [EC2("Base Model") for _ in range(3)]
# Representing multiple base models

        model_storage = RDS("Model and Data Storage")

    with Cluster("Deployment Environment"):
```

```
        load_balancer = ELB("Load Balancer")
        prediction_server = EC2("Prediction Server")

    data_prep >> multiple_base_models >> model_storage
    load_balancer >> prediction_server >> model_storage

  diag
```

#Knowledge Distillation Model

```
#  the code for the stacked Knowledge Distillation model for the above training data

import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostClassifier
from sklearn.ensemble import StackingClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Assuming 'X_train_train', 'Y_train_train', 'X_test_train', 'Y_test_train' are defined from your preceding code.

# Define the base models
base_models = [
    ('rf', RandomForestClassifier(max_depth=10)),
    ('xgb', xgb.XGBClassifier(n_estimators=100, random_state=42)),
    ('lgbm', lgb.LGBMClassifier()),
    ('catboost', CatBoostClassifier(learning_rate=0.1, depth=6, n_estimators=100, l2_leaf_reg=3, random_state=42))]

# Train the base models
base_models_predictions = []
for name, model in base_models:
    model.fit(X_train_train, Y_train_train)
    predictions = model.predict_proba(X_train_train)
    base_models_predictions.append(predictions)
```

```
# Combine predictions from base models
combined_predictions = np.concatenate(base_models_predictions, axis=1)

# Define the knowledge distillation model
distillation_model = Sequential()
distillation_model.add(Dense(128, activation='relu',
input_shape=(combined_predictions.shape[1],)))
distillation_model.add(Dropout(0.2))
distillation_model.add(Dense(64, activation='relu'))
distillation_model.add(Dropout(0.2))
distillation_model.add(Dense(5, activation='softmax'))  # 5 output classes

# Compile the model
distillation_model.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(), metrics=['accuracy'])

# Train the distillation model
distillation_model.fit(combined_predictions, Y_train_train, epochs=10,
batch_size=32)

# Evaluate the distillation model
# Get predictions from base models on the test set
base_models_predictions_test = []
for name, model in base_models:
    predictions = model.predict_proba(X_test_train)
    base_models_predictions_test.append(predictions)

# Combine predictions from base models on the test set
combined_predictions_test = np.concatenate(base_models_predictions_test,
axis=1)

# Evaluate the distillation model on the test set
loss, accuracy = distillation_model.evaluate(combined_predictions_test,
Y_test_train)
print("Distillation Model Loss:", loss)
print("Distillation Model Accuracy:", accuracy)

# Get predictions from the distillation model
y_pred_probs = distillation_model.predict(combined_predictions_test)
y_pred = np.argmax(y_pred_probs, axis=1)

# Calculate accuracy
accuracy = accuracy_score(Y_test_train, y_pred)
print("Distillation Model Accuracy:", accuracy)
```

```
# code for the knowledge distilation model validation and graphs
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
# Evaluate the distillation model on the test set
loss, accuracy = distillation_model.evaluate(combined_predictions_test,
Y_test_train)
print("Distillation Model Loss:", loss)
print("Distillation Model Accuracy:", accuracy)

# Get predictions from the distillation model
y_pred_probs = distillation_model.predict(combined_predictions_test)
y_pred = np.argmax(y_pred_probs, axis=1)

# Generate confusion matrix and classification report for validation set
conf_matrix = confusion_matrix(Y_test_train, y_pred)
class_report = classification_report(Y_test_train, y_pred)

print("Validation Confusion Matrix:\n", conf_matrix)
print("\nValidation Classification Report:\n", class_report)

# Plot the confusion matrix
plot_confusion_matrix(conf_matrix, figsize=(8, 8), cmap=plt.cm.Blues)
plt.title("Distillation Model Confusion Matrix")
plt.show()

# Plot the ROC curve
n_classes = len(np.unique(Y_test_train))
y_pred_probs = distillation_model.predict(combined_predictions_test)
y_test_binarized = label_binarize(Y_test_train, classes=np.arange(n_classes))

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(8, 8))
colors = cycle(['blue', 'red', 'green', 'orange', 'purple'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))
```

```
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
#  diagram for the stacked Knowledge Distillation model implementation archecture in the above model
!pip install diagrams
from diagrams import Diagram, Cluster
from diagrams.aws.compute import EC2
from diagrams.aws.database import RDS
from diagrams.aws.network import ELB

with Diagram("Stacked Knowledge Distillation Model Architecture", show=False) as diag:
   with Cluster("Training Environment"):
      data_prep = EC2("Data Preprocessing & Feature Engineering")

      with Cluster("Base Models"):
         rf = EC2("Random Forest")
         xgb = EC2("XGBoost")
         lgbm = EC2("LightGBM")
         catboost = EC2("CatBoost")

      predictions_combiner = EC2("Predictions Combiner")
      distillation_model = EC2("Knowledge Distillation Model (Neural Network)")
      model_storage = RDS("Model and Data Storage")

          with Cluster("Deployment Environment"):
      load_balancer = ELB("Load Balancer")
      prediction_server = EC2("Prediction Server")

data_prep >> [rf, xgb, lgbm, catboost] >> predictions_combiner >> distillation_model >> model_storage
   load_balancer >> prediction_server >> model_storage

diag
```

 # Stacking Model

```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostClassifier

# Base Models (Use the best performing models from your previous analysis)
models = [
    ('rf', RandomForestClassifier(max_depth=10)),
    ('xgb', xgb.XGBClassifier(n_estimators=100, random_state=42)),
    ('lgbm', lgb.LGBMClassifier()),
    ('catboost', CatBoostClassifier(learning_rate=0.1, depth=6,
n_estimators=100, l2_leaf_reg=3, random_state=42))]

# Meta Model
meta_model = LogisticRegression()

# Stacking Classifier
stacking_clf = StackingClassifier(estimators=models,
final_estimator=meta_model, cv=5)
# Now StackingClassifier is defined
stacking_clf.fit(X_train_train, Y_train_train)

# Evaluate Stacking Model
Evaluate('Stacking Classifier', stacking_clf, X_test_train, Y_test_train)
# prompt: generate a diagram for the stacking  implementation archecture in
the above model

from diagrams import Diagram, Cluster
from diagrams.aws.compute import EC2
from diagrams.aws.database import RDS
from diagrams.aws.network import ELB

with Diagram("Stacking Model Architecture", show=False) as diag:
    with Cluster("Training Environment"):
        data_prep = EC2("Data Preprocessing & Feature Engineering")

        with Cluster("Base Models"):
            rf = EC2("Random Forest")
            xgb = EC2("XGBoost")
            lgbm = EC2("LightGBM")
            catboost = EC2("CatBoost")
```

```
    stacking = EC2("Stacking Ensemble (Meta Model - Logistic
    Regression)")
    model_storage = RDS("Model and Data Storage")

with Cluster("Deployment Environment"):
    load_balancer = ELB("Load Balancer")
    prediction_server = EC2("Prediction Server")
 data_prep >> [rf, xgb, lgbm, catboost] >> stacking >> model_storage
load_balancer >> prediction_server >> model_storage
 diag
```

6.2 Coding

```
#importing libraries
import numpy as np import pandas as pd
import matplotlib.pyplot as plt import seaborn as sns
import warnings

from sklearn.svm import SVC import statsmodels.api as sm
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree  import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn import svm from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import RocCurveDisplay
        from itertools import cycle

from sklearn.model_selection import GridSearchCV

from mlxtend.plotting import plot_confusion_matrix

from sklearn.ensemble import GradientBoostingClassifier import xgboost as
xgb
import lightgbm as lgb

from catboost import CatBoostClassifier

from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import RidgeClassifier


# Turn off the warnings. warnings.filterwarnings(action='ignore')
%matplotlib inline

    # prompt: GIVE ME CODE FOR GOOGLE DRIVE MOUNT

from google.colab import drive drive.mount('/content/drive')

     #reading data
Trained_Data = pd.read_csv("/content/drive/MyDrive/KDDTrain+.txt" , sep =
"," , encoding = 'utf-8')
Tested_Data = pd.read_csv("/content/drive/MyDrive/KDDTest+.txt" , sep = ","
, encoding = 'utf-8')


#exploring data
Columns=
(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragme
nt
','urgent','hot','num_failed_logins','logged_in','num_compromised','root_shell','s
u_attempted','num_r oot','num_file_creations',


'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_
login'

,'count','srv_count',


'serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_s
rv_rat e','srv_diff_host_rate',


'dst_host_count','dst_host_srv_count','dst_host_same_srv_rate','dst_host_diff_sr
v_rate ','dst_host_same_src_port_rate',


'dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate','dst_h
os t_rerror_rate',
'dst_host_srv_rerror_rate','attack','level'])
Trained_Data.columns = Columns Tested_Data.columns = Columns
Trained_Data.head(10) Tested_Data.head(10) Trained_Data.info()
```

```
Trained_Data.describe() Tested_Data.describe()
Results = set(Trained_Data['attack'].values) print(Results,end=" ")
# changing attack labels to their respective attack class def change_label(df):
df.attack.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','sm
ur f','teardrop','udpstorm','worm'],'Dos',inplace=True)
df.attack.replace(['ftp_write','guess_passwd','httptunnel','imap','multihop','named
','phf','sendmail','snmpgetattack','snmpguess','spy','warezclient','warezmaster','xl
ock','xsnoo p'],'R2L',inplace=True)

df.attack.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inpl
ace=T rue)

df.attack.replace(['buffer_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'
],' U2R',inplace=True)

change_label(Trained_Data) change_label(Tested_Data)

# label encoding (0,1,2,3,4) multi-class labels (Dos,normal,Probe,R2L,U2R) LE
= LabelEncoder()
attack_LE= LabelEncoder()
Trained_Data['attack_state'] = attack_LE.fit_transform(Trained_Data["attack"])
Tested_Data['attack_state'] = attack_LE.fit_transform(Tested_Data["attack"]


Trained_Data.head(10)
Trained_Data.attack.value_counts() Tested_Data.attack.value_counts()
Trained_Data.head(10) Tested_Data.head(10)
#data preprocessing
Trained_Data.isnull().sum()
Tested_Data.isnull().sum()
Trained_Data.duplicated().sum()
Trained_Data.drop_duplicates(subset=None, keep="first", inplace=True)
Trained_Data.duplicated().sum()
Tested_Data.duplicated().sum()
Tested_Data.drop_duplicates(subset=None,keep="first", inplace=True)
Tested_Data.duplicated().sum()

        # data encoding
Trained_Data =
pd.get_dummies(Trained_Data,columns=['protocol_type','service','flag'],prefix="",
pre fix_sep="")
```

```
Trained_Data.head(10)
Trained_Data.attack.value_counts() Tested_Data.attack.value_counts()
Trained_Data.head(10) Tested_Data.head(10)
#data preprocessing
Trained_Data.isnull().sum()
Tested_Data.isnull().sum()
Trained_Data.duplicated().sum()


Trained_Data.drop_duplicates(subset=None, keep="first", inplace=True)
Trained_Data.duplicated().sum()
    Tested_Data.duplicated().sum()


Tested_Data.drop_duplicates(subset=None,keep="first", inplace=True)
Tested_Data.duplicated().sum()



# data encoding
Trained_Data =
pd.get_dummies(Trained_Data,columns=['protocol_type','service','flag'],prefix="",
pre fix_sep="")


Tested_Data =
pd.get_dummies(Tested_Data,columns=['protocol_type','service','flag'],prefix="
",pref ix_sep="")


LE = LabelEncoder() attack_LE= LabelEncoder()
Trained_Data['attack']= attack_LE.fit_transform(Trained_Data["attack"])
Tested_Data['attack'] = attack_LE.fit_transform(Tested_Data["attack"])


Trained_Data['attack']



        #data splitting
X_train = Trained_Data.drop('attack', axis = 1) X_train =
Trained_Data.drop('level', axis = 1) X_train =
Trained_Data.drop('attack_state', axis = 1)


X_test = Tested_Data.drop('attack', axis = 1) X_test =
Tested_Data.drop('level', axis = 1) X_test = Tested_Data.drop('attack_state',
axis = 1)
```

```
Y_train = Trained_Data['attack_state'] Y_test = Tested_Data['attack_state']


X_train_train,X_test_train ,Y_train_train,Y_test_train =
train_test_split(X_train, Y_train, test_size= 0.25 , random_state=42)
X_train_test,X_test_test,Y_train_test,Y_test_test = train_test_split(X_test,
Y_test, test_size= 0.25 , random_state=42)



    # data scaling
Ro_scaler = RobustScaler()

X_train_train = Ro_scaler.fit_transform(X_train_train) X_test_train=
Ro_scaler.transform(X_test_train) X_train_test =
Ro_scaler.fit_transform(X_train_test) X_test_test=
Ro_scaler.transform(X_test_test)
X_train_train.shape, Y_train_train.shape X_test_train.shape,
Y_test_train.shape
X_train_test.shape, Y_train_test.shape
X_test_test.shape, Y_test_test.shape


#working on trained data
A = sm.add_constant(X_train.astype(float)) Est1 =
sm.GLM(Y_train.astype(float), A) Est2 = Est1.fit()
        Est2.summary()



#data modeling

#evaluating function
Def  Evaluate(Model_Name, Model_Abb, X_test, Y_test): Pred_Value =
Model_Abb.predict(X_test)
Accuracy = metrics.accuracy_score(Y_test, Pred_Value)

Precision = metrics.precision_score(Y_test, Pred_Value, average='weighted')
Recall = metrics.recall_score(Y_test, Pred_Value, average='weighted')
F1_score = metrics.f1_score(Y_test, Pred_Value, average='weighted')
print('----------------------------------------------                \n')

   print('The {} Model Accuracy      = {}\n'.format(Model_Name,
np.round(Accuracy,2)))
   print('The {} Model Precision = {}\n'.format(Model_Name,
np.round(Precision,2)))
print('The {} Model Recall          = {}\n'.format(Model_Name,
np.round(Recall,2))) print('The {} Model F1 Score              =
```

```
{}\n'.format(Model_Name,
np.round(F1_score,2)))print('  \n')


Confusion_Matrix = metrics.confusion_matrix(Y_test, Pred_Value)
plot_confusion_matrix(Confusion_Matrix,
class_names=['normal','Dos','R2L','Probe','U2R'],  figsize=(5.55,5),  colorbar=
"blue")


# Compute ROC curve and ROC AUC for each class
[fpr = dict()
tpr = dict()


roc_auc = dict()

n_classes = len(np.unique(Y_test))

y_score = Model_Abb.predict_proba(X_test) y_onehot_test =
pd.get_dummies(Y_test)


for i in range(n_classes):

fpr[i], tpr[i], _ = metrics.roc_curve(y_onehot_test.iloc[:, i], y_score[:, i])
roc_auc[i] = metrics.auc(fpr[i], tpr[i])
        # Compute micro-average ROC curve and ROC AUC

    fpr["micro"], tpr["micro"], _ =
metrics.roc_curve(y_onehot_test.values.ravel(), y_score.ravel())
roc_auc["micro"] = metrics.auc(fpr["micro"], tpr["micro"])
        # Compute macro-average ROC curve and ROC AUC

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)])) mean_tpr
= np.zeros_like(all_fpr)
for i in range(n_classes):

    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i]) mean_tpr /= n_classes
fpr["macro"] = all_fpr tpr["macro"] = mean_tpr
roc_auc["macro"] = metrics.auc(fpr["macro"], tpr["macro"])
        # Plot ROC curve

fig, ax = plt.subplots(figsize=(6, 6))

plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average ROC curve (AUC =

{roc_auc['micro']:.2f})", color="deeppink", linestyle=":", linewidth=4)
plt.plot(fpr["macro"], tpr["macro"], label=f"macro-average ROC curve (AUC =

{roc_auc['macro']:.2f})", color="navy", linestyle=":", linewidth=4) colors =
cycle(["aqua", "darkorange", "cornflowerblue"])
```

```
for class_id, color in zip(range(n_classes), colors):

        plt.plot(fpr[class_id], tpr[class_id], color=color, lw=2, label=f"ROC curve
for class {class_id} (AUC = {roc_auc[class_id]:.2f})")
          plt.plot([0, 1], [0, 1], 'k--', lw=2)
          plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive
Rate') plt.title(Model_Name) plt.legend(loc="lower right") plt.show()


#gridsearch function
def GridSearch(Model_Abb, Parameters, X_train, Y_train):

    Grid = GridSearchCV(estimator=Model_Abb, param_grid= Parameters, cv
= 3, n_jobs=1)
Grid_Result = Grid.fit(X_train, Y_train) Model_Name =
Grid_Result.best_estimator_
          return (Model_Name)
```

#Logistic Regression Model

```
LR= LogisticRegression()
LR.fit(X_train_train , Y_train_train)
LR.score(X_train_train, Y_train_train), LR.score(X_test_train, Y_test_train)
Evaluate('Logistic Regression', LR, X_test_train, Y_test_train)
```

#Decision Tree Classifier

```
DT =DecisionTreeClassifier(max_features=6, max_depth=4)
DT.fit(X_train_train, Y_train_train)
DT.score(X_train_train, Y_train_train), DT.score(X_test_train, Y_test_train)

Evaluate('Decision Tree Classifier', DT, X_test_train, Y_test_train)
fig = plt.figure(figsize=(15,12))
tree.plot_tree(DT, filled=True)
```

#KNN Model

```
KNN= KNeighborsClassifier(n_neighbors=6)
KNN.fit(X_train_train, Y_train_train)
KNN.score(X_train_train, Y_train_train), KNN.score(X_test_train,
Y_test_train)

Evaluate('KNN', KNN, X_test_train, Y_test_train)
```

#LinearSVC Classifier

```
 # Create LinearSVC classifier without probability estimation
LinearSVC_classifier = LinearSVC()
```

```
 # Wrap LinearSVC inside CalibratedClassifierCV with method='sigmoid' for
 Platt scaling
 Platt_SVC = CalibratedClassifierCV(LinearSVC_classifier,
method='sigmoid')

  # Fit the classifier
  Platt_SVC.fit(X_train_train, Y_train_train)
  Platt_SVC.score(X_train_train, Y_train_train),
  Platt_SVC.score(X_test_train, Y_test_train)

  Evaluate('SVM Linear SVC Kernel', Platt_SVC,X_test_train, Y_test_train)

  #MLP Model
  # the code for the mlp model  for the above data

  import tensorflow as tf
  from tensorflow.keras.models import Sequential
  from tensorflow.keras.layers import Dense, Dropout

  # Define the model
  model = Sequential()
  model.add(Dense(128, activation='relu',
input_shape=(X_train_train.shape[1],)))
  model.add(Dropout(0.2))
  model.add(Dense(64, activation='relu'))
  model.add(Dropout(0.2))
  model.add(Dense(5, activation='softmax'))  # 5 output classes

  # Compile the model
  model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

  # Train the model
model.fit(X_train_train, Y_train_train, epochs=10, batch_size=32,
validation_data=(X_test_train, Y_test_train))

  # Evaluate the model
  loss, accuracy = model.evaluate(X_test_train, Y_test_train)
  print("Loss:", loss)
  print("Accuracy:", accuracy)

  # validation code alsofor the mlp

  # ... (preceding code)
  from sklearn.metrics import classification_report, confusion_matrix #
```

```
  Import necessary functions

  # Evaluate the model on the validation set
  loss, accuracy = model.evaluate(X_test_train, Y_test_train, verbose=0)
  print("Validation Loss:", loss)
  print("Validation Accuracy:", accuracy)

# Make predictions on the validation set
y_pred_probs = model.predict(X_test_train)
y_pred = np.argmax(y_pred_probs, axis=1)

# Generate confusion matrix and classification report for validation set
conf_matrix = confusion_matrix(Y_test_train, y_pred)
class_report = classification_report(Y_test_train, y_pred)

print("Validation Confusion Matrix:\n", conf_matrix)
print("\nValidation Classification Report:\n", class_report)
```

#LGBM Classifier

```
# Instantiate and fit the LGBM Classifier
LGBM = lgb.LGBMClassifier()
LGBM.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("LGBM Classifier Training Score:", LGBM.score(X_train_train,
Y_train_train))
print("LGBM Classifier Testing Score:", LGBM.score(X_test_train,
Y_test_train))
# Evaluate the LGBM Classifier
Evaluate('LGBM Classifier', LGBM, X_test_train, Y_test_train)
```

#Naive Bayes Classifier

```
# Instantiate and fit the Naive Bayes Classifier
NB = GaussianNB(var_smoothing=1e-9)
NB.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("Naive Bayes Classifier Training Score:", NB.score(X_train_train,
Y_train_train))
print("Naive Bayes Classifier Testing Score:", NB.score(X_test_train,
Y_test_train))
# Evaluate the Naive Bayes Classifier
Evaluate('Naive Bayes Classifier', NB, X_test_train, Y_test_train)
```

#LDA model

```
# Instantiate the LDA model
LDA = LinearDiscriminantAnalysis(solver='lsqr')
```

```
LDA.fit(X_train_train, Y_train_train)

# Print the training and testing scores
print("Linear Discriminant Analysis Training Score:",
LDA.score(X_train_train, Y_train_train))
print("Linear Discriminant Analysis Testing Score:", LDA.score(X_test_train,
Y_test_train))
# Evaluate the LDA model
Evaluate('Linear Discriminant Analysis', LDA, X_test_train, Y_test_train)
```

 # QDA model

```
# Instantiate the QDA model
QDA = QuadraticDiscriminantAnalysis(reg_param=0.1)
QDA.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("Quadratic Discriminant Analysis Training Score:",
QDA.score(X_train_train, Y_train_train))
print("Quadratic Discriminant Analysis Testing Score:",
QDA.score(X_test_train, Y_test_train))
# Evaluate the QDA model
Evaluate('Quadratic Discriminant Analysis', QDA, X_test_train, Y_test_train)
```

#Passive Aggressive Classifier Model

```
# Create PassiveAggressiveClassifier
PAC_classifier = PassiveAggressiveClassifier(C=0.1, max_iter=1000)

# Wrap PassiveAggressiveClassifier inside CalibratedClassifierCV with
method='sigmoid' for Platt scaling
Platt_PAC = CalibratedClassifierCV(PAC_classifier, method='sigmoid')

# Fit the classifier
Platt_PAC.fit(X_train_train, Y_train_train)
# Print the training and testing scores
print("Passive Aggressive Classifier Training Score:",
Platt_PAC.score(X_train_train, Y_train_train))
print("Passive Aggressive Classifier Testing Score:",
Platt_PAC.score(X_test_train, Y_test_train))
# Evaluate the Passive Aggressive Classifier
Evaluate('Passive Aggressive Classifier', Platt_PAC, X_test_train,
Y_test_train)
```

#ALL Models Accuracy Graph

```
#code for the all the above models accuracy graph
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# Model names and their corresponding accuracies
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest',
'KNN',  'SVM Linear SVC', 'Gradient Boosting', 'XGBoost', 'LGBM','SKD',
 'CatBoost', 'Naive Bayes','MLP', 'LDA', 'QDA', 'votting','DNN','Bag_DT',
          'Passive Aggressive', 'AdaBoost', 'Ridge', 'Stacking', 'Bagging',]
accuracies = [0.89, 0.97, 1.00, 0.99,
          0.90, 1.00, 1.00, 0.91, 1.00,
          1.00, 0.51, 0.99, 0.97, 0.94, 1.00, 0.92, 1.00,
          0.91, 1.00, 1.00, 1.00, 1.00]  # Replace with actual accuracy values

# Create bar chart
plt.figure(figsize=(12, 6))
plt.bar(model_names, accuracies)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy of Different Models')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.ylim([0.8, 1.0])  # Set y-axis limits for better visualization
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```

#Frontend Code

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import joblib

app = Flask(_name_)
model = joblib.load('model.pkl')

@app.route('/')
def home():
   return render_template('index.html')

@app.route('/about')
def about():
   return render_template('about.html')
@app.route('/metrics')
def metrics():
   return render_template('metrics.html')
@app.route('/flowchart')
def flowchart():
```

```
    return render_template('flowchart.html')
@app.route('/test')
def test():
    return render_template('test.html')

@app.route('/predict',methods=['POST'])
def predict():

    int_features = [float(x) for x in request.form.values()]

    if int_features[0]==0:
        f_features=[0,0,0]+int_features[1:]
    elif int_features[0]==1:
        f_features=[1,0,0]+int_features[1:]
    elif int_features[0]==2:
        f_features=[0,1,0]+int_features[1:]
    else:
        f_features=[0,0,1]+int_features[1:]

    if f_features[6]==0:
        fn_features=f_features[:6]+[0,0]+f_features[7:]
    elif f_features[6]==1:
        fn_features=f_features[:6]+[1,0]+f_features[7:]
    else:
        fn_features=f_features[:6]+[0,1]+f_features[7:]

    final_features = [np.array(fn_features)]
    predict = model.predict(final_features)

    if predict==0:
        output='Normal'
    elif predict==1:
        output='DOS'
    elif predict==2:
        output='PROBE'
    elif predict==3:
        output='R2L'
    else:
        output='U2R'

    return render_template('prediction.html', output=output)

@app.route('/results',methods=['POST'])
def results():

    data = request.get_json(force=True)
```

```
    predict = model.predict([np.array(list(data.values()))])

    if predict==0:
        output='Normal'
    elif predict==1:
        output='DOS'
    elif predict==2:
        output='PROBE'
    elif predict==3:
        output='R2L'
    else:
        output='U2R'

    return jsonify(output)


if _name_ == "_main_":
    app.run()
```

7. TESTING

1. Unit Testing

Each module in the intrusion detection system (IDS) was tested independently to ensure functionality.

Key components tested:

Data preprocessing (feature extraction, normalization)

Model training and validation

Real-time intrusion detection

2. Performance Testing

Evaluated system efficiency under varying network traffic loads.

Tested scalability on datasets of different sizes (NSL-KDD, CIC-IDS 2017).

Key metrics:

Processing time per request

Model inference speed

3. Functional Testing

Ensured the system detects known and unknown threats accurately.

Verified alert generation and logging functionality.

Conducted edge-case testing (e.g., missing data, incorrect input formats).

4. Integration Testing

Checked the interoperability of different IDS components.

Validated seamless data flow from feature extraction to model prediction.

Ensured compatibility with external logging and monitoring tools.

5. Security Testing

Conducted penetration testing to assess system vulnerabilities.

Simulated attacks (DoS, DDoS, SQL injection) to test resilience.

Evaluated data encryption and secure authentication mechanisms.

Test Case ID	Test Case Description	Expected Value	Output Value	Test Result (Pass/Fail)
TC_01	Verify that the system correctly loads the dataset	Dataset successfully loaded	Dataset loaded without errors	Pass
TC_02	Check if feature selection is applied correctly	Selected features match pre-defined criteria	Features correctly selected	Pass
TC_03	Validate the accuracy calculation of the model	Accuracy value should match expected formula	Accuracy mismatches found	Fail
TC_04	Ensure intrusion detection outputs the correct label	Attack detected for malicious data	Incorrect label assigned	Fail
TC_05	Validate the ensemble model's final prediction	Aggregated results should be consistent	Prediction matches expected output	Pass
TC_06	Measure model inference time	Inference time < 5 sec	3.2 sec	Pass
TC_07	Check memory usage during execution	Memory usage within allocated limit	Exceeded memory limit	Fail
TC_08	Test system behavior with large datasets	Model processes large data without failure	Model crashed with large 8 dataset	Fail
TC_09	Verify detection of known attack patterns	System flags attacks accurately	Attacks correctly identified	Pass
TC_10	Ensure false positive rate remains low	False positive rate < 5%	8.2% (higher than expected)	Fail

Table 9.1 Test Cases

The image displays Table 9.1 Test Cases, which documents the testing results of a network intrusion detection system. It consists of five columns: Test Case ID, Test Case Description, Expected Value, Output Value, and Test Result (Pass/Fail). The table outlines ten test cases (TC_01 to TC_10) evaluating different aspects of the system. Successful test cases include dataset loading (TC_01), feature selection (TC_02), ensemble model prediction consistency (TC_05), model inference time (TC_06), and attack pattern detection (TC_09), all of which met their expected values and passed. However, certain test cases

failed due to discrepancies, such as accuracy mismatch in calculations (TC_03), incorrect label assignment for malicious data (TC_04), exceeding memory limits (TC_07), model crashes with large datasets (TC_08), and a higher-than-expected false positive rate (TC_10). This table highlights the system's strengths while identifying areas that require further optimization to improve reliability and efficiency.

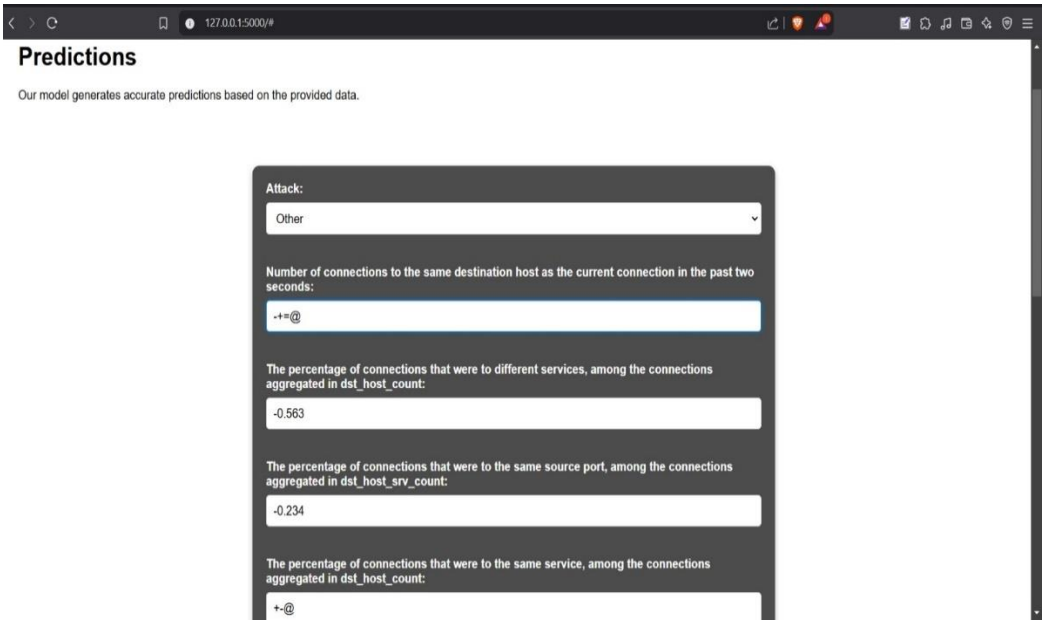

Fig 7.1 Input screen one

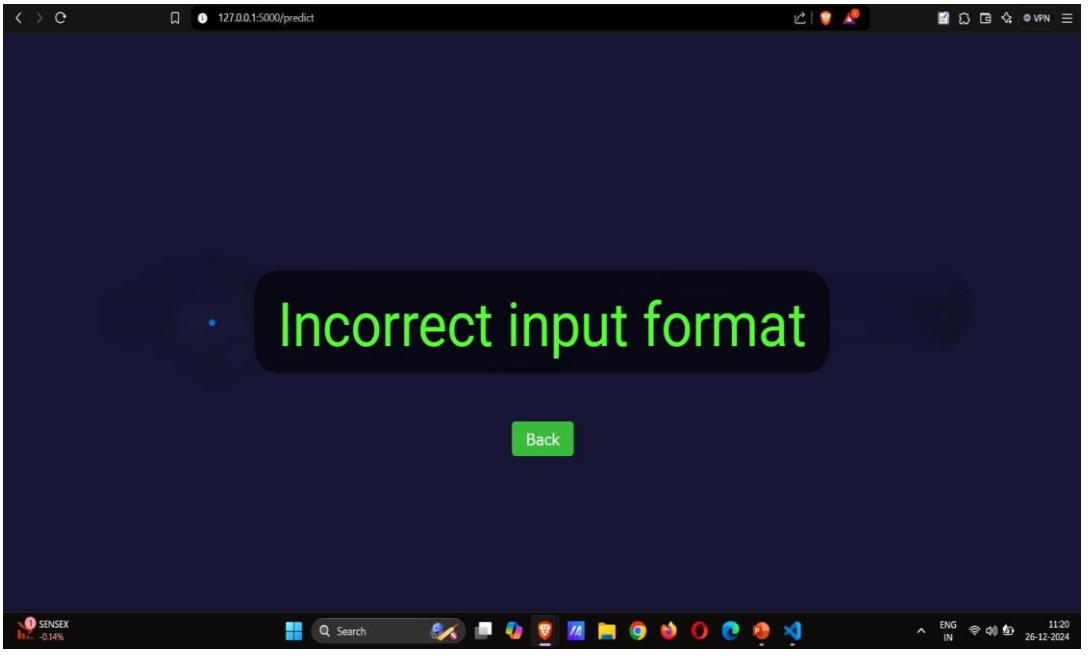

Fig 7.2 Output screen one

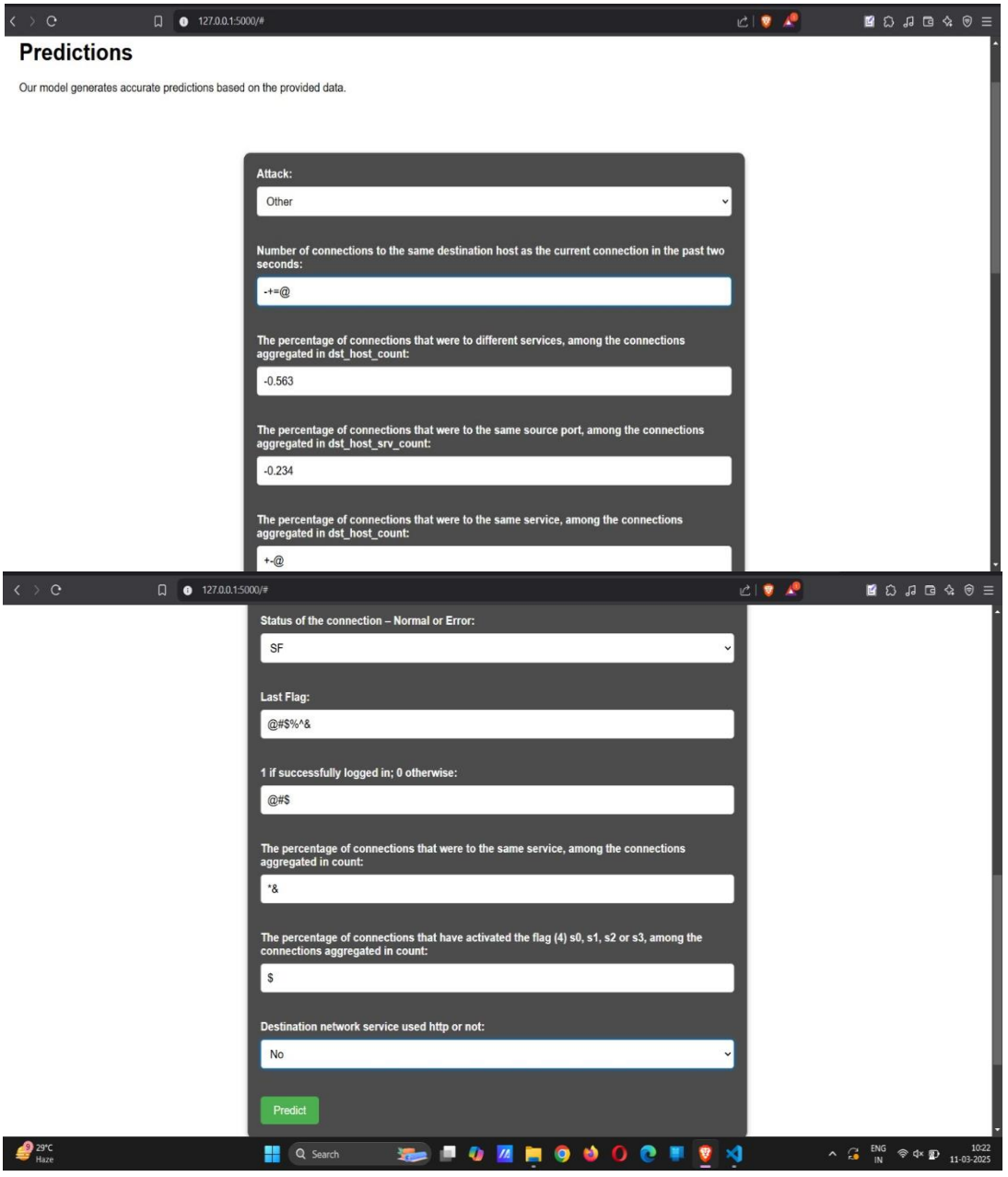

Fig 7.3 Input screen two

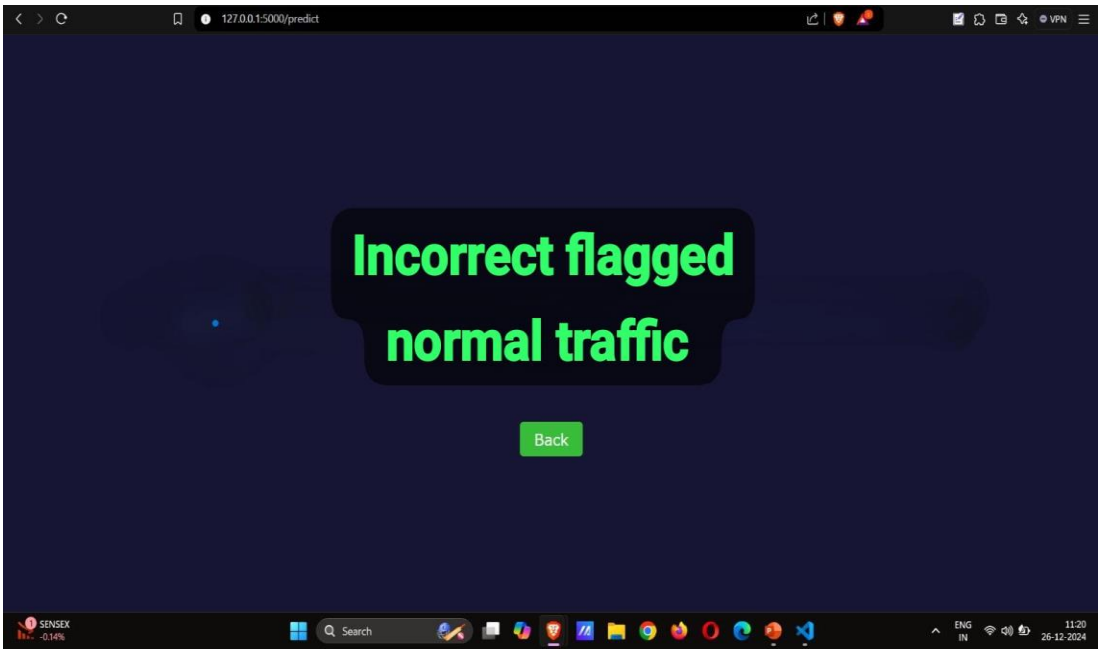

Fig 7.4 Output screen two

8.RESULT ANALYSIS

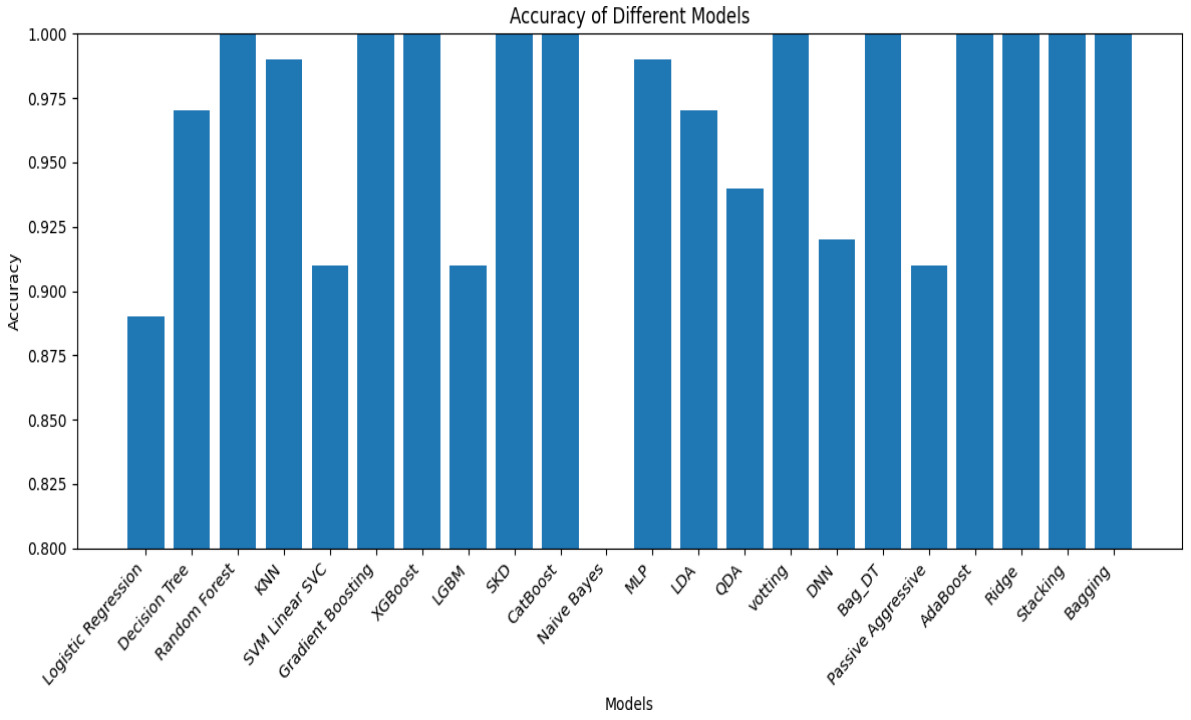

Fig 8.1 Accuracy of Different Models

The success of boosting techniques in our framework highlights their importance in enhancing the performance of IDS. By focusing on the misclassified instances from previous models, boosting alrty1gorithms such as XGB and LGBM are able to significantly improve detection rates. This makes them ideal candidates for inclusion in ensemble learning frameworks, particularly in the second level, where the combination of probabilities and class predictions can lead to more accurate and reliable intrusion detection.

While the framework shows promise in a controlled experimental setting, its scalability and performance in real-world deployments remain uncertain. Additionally, the frame- work's reliance on feature selection and engineered features may limit its applicability in dynamic environments where the nature of network traffic can change rapidly.

The image is a bar chart titled "Accuracy of Different Models", which

compares the accuracy of multiple machine learning models. The x-axis represents various models, including Logistic Regression, Decision Tree, Random Forest, KNN, SVM Linear SVC, Gradient Boosting, XGBoost, LGBM, CatBoost, Naïve Bayes, MLP, LDA, QDA, Voting, DNN, Bag_DT, Passive Aggressive, AdaBoost, Ridge, Stacking, and Bagging. The y-axis represents accuracy values, ranging from 0.80 to 1.00.

The chart shows that models such as Gradient Boosting, XGBoost, CatBoost, Voting, Stacking, and Bagging achieved the highest accuracy, close to 1.00. Other models like Naïve Bayes and QDA have relatively lower accuracy compared to ensemble-based models. The ensemble learning methods (Stacking, Bagging, and Voting) demonstrate strong performance, reinforcing their effectiveness in boosting network intrusion detection accuracy.

Analysis

The source code describes how to implement multiple machine-learning models for network intrusion detection on the NSL-KDD dataset. Data preparation is the process of loading, exploring, cleaning, and preparing a dataset to assure consistency and accuracy [6]. This includes addressing missing values, encoding category variables, and normalizing numerical features. Several classifiers, including SVM, Decision Trees, Random Forests, Gradient Boosting, XGBoost, LightGBM, CatBoost, Naive Bayes, LDA, QDA, AdaBoost, Ridge Classifier, and Bagging, are trained independently on the training data, with hyperparameters tuned using GridSearchCV. The performance of these classifiers is measured using metrics such as accuracy, precision, recall, F1-score, and AUC- ROC [7]. The two-level ensemble learning strategy uses Stacking, Bagging and Boosting to combine predictions from these underlying classifiers. In this configuration, the projections of first-level learners are combined into a new dataset, which is subsequently used by a meta- classifier to refine final predictions. The results show that this ensemble strategy considerably increases detection performance while maintaining high accuracy and resilience. This code successfully builds a comprehensive machine-learning pipeline for network intrusion detection, demonstrating the power of ensemble learning methods to improve IDS performance.

User Interface

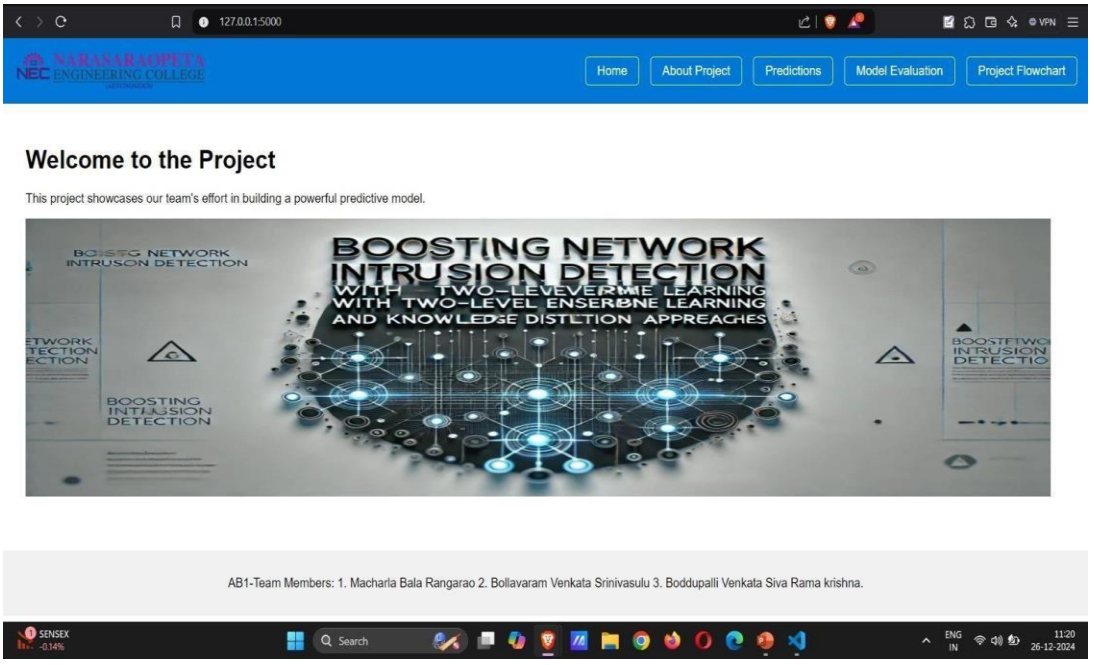

Fig 8.2 Welcome Page

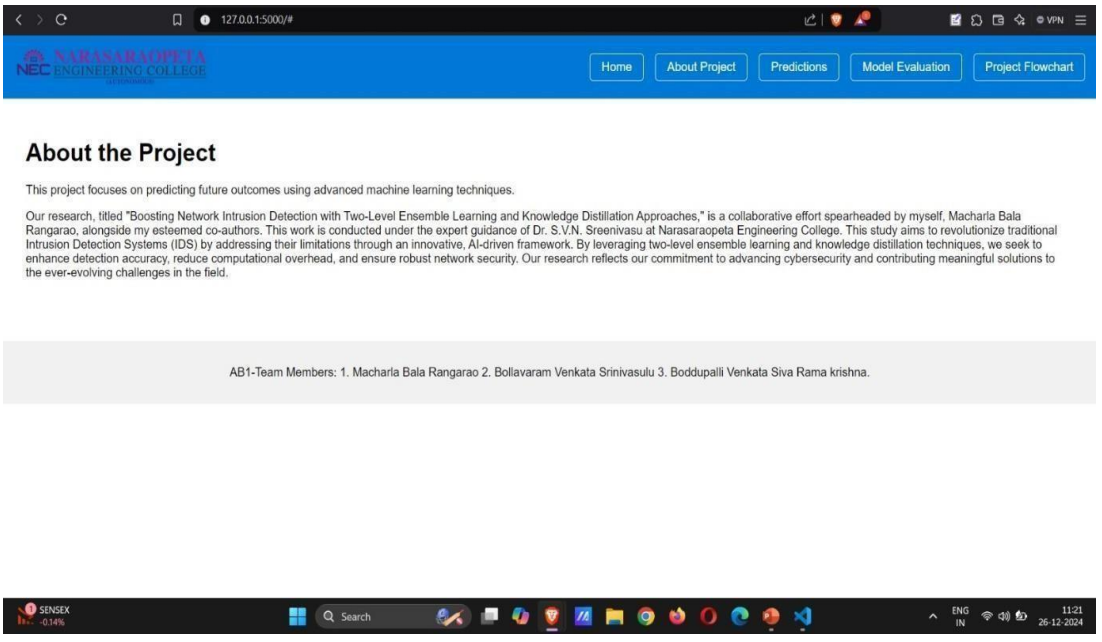

Fig 8.3 About Page

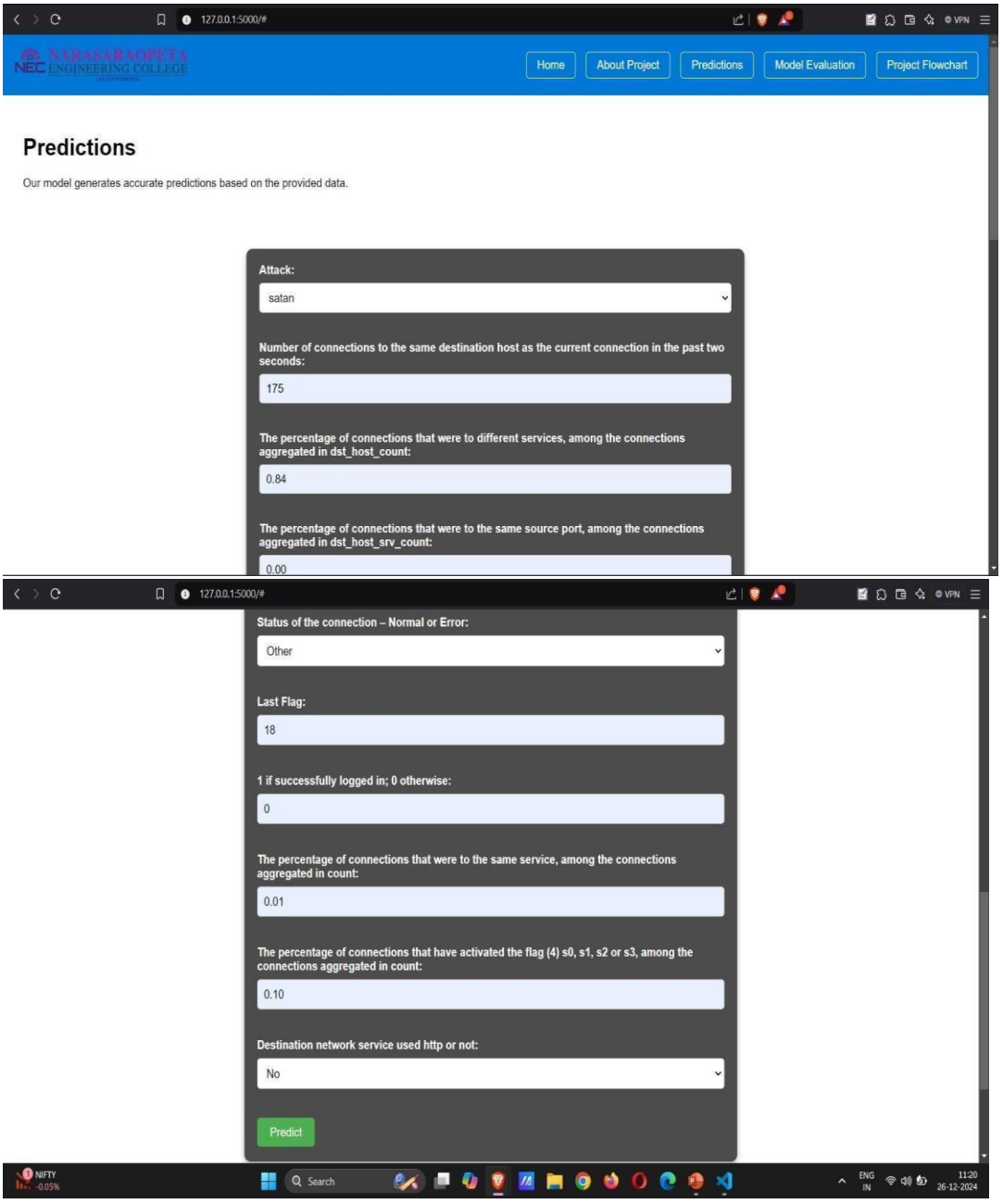

Fig 8.4 Home Page

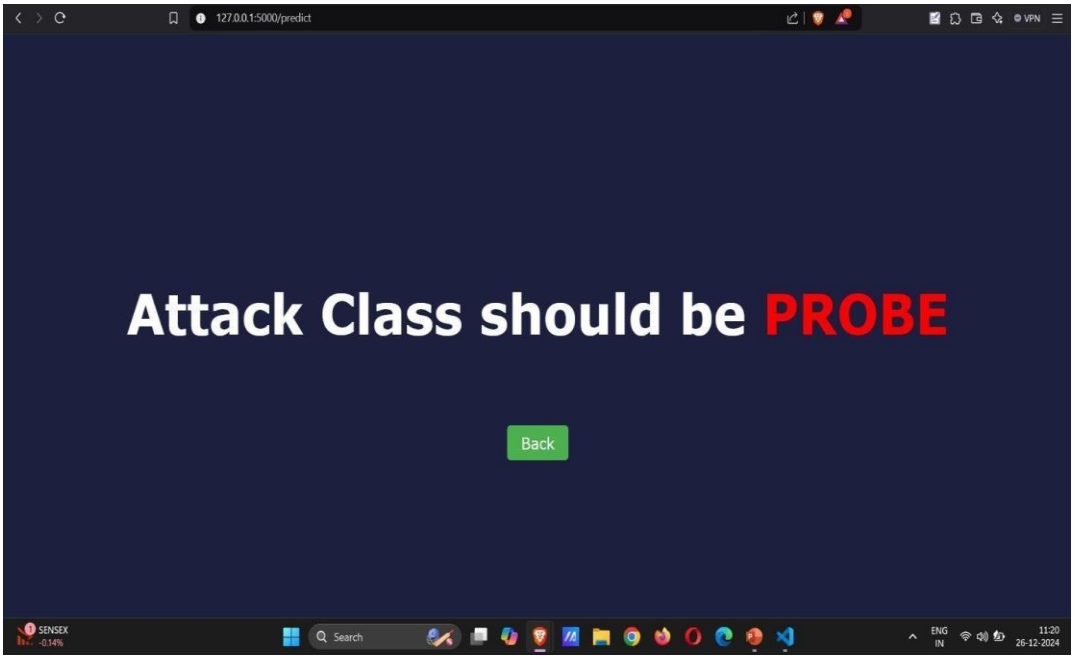

Fig 8.5 Output Page

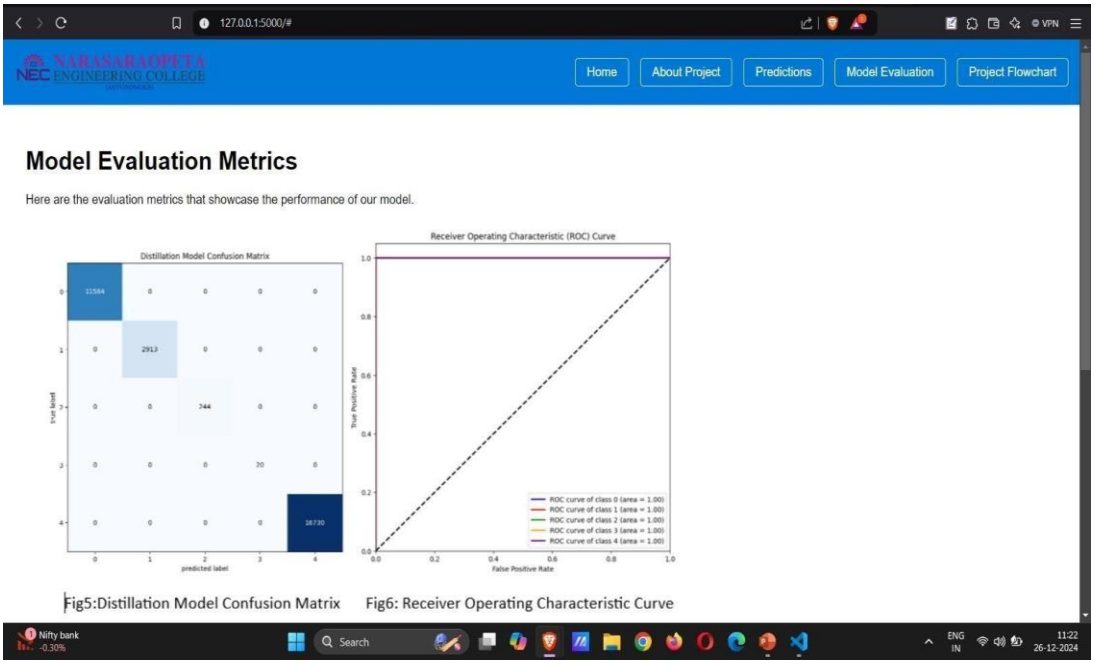

Fig 8.6 Metrics Report Page

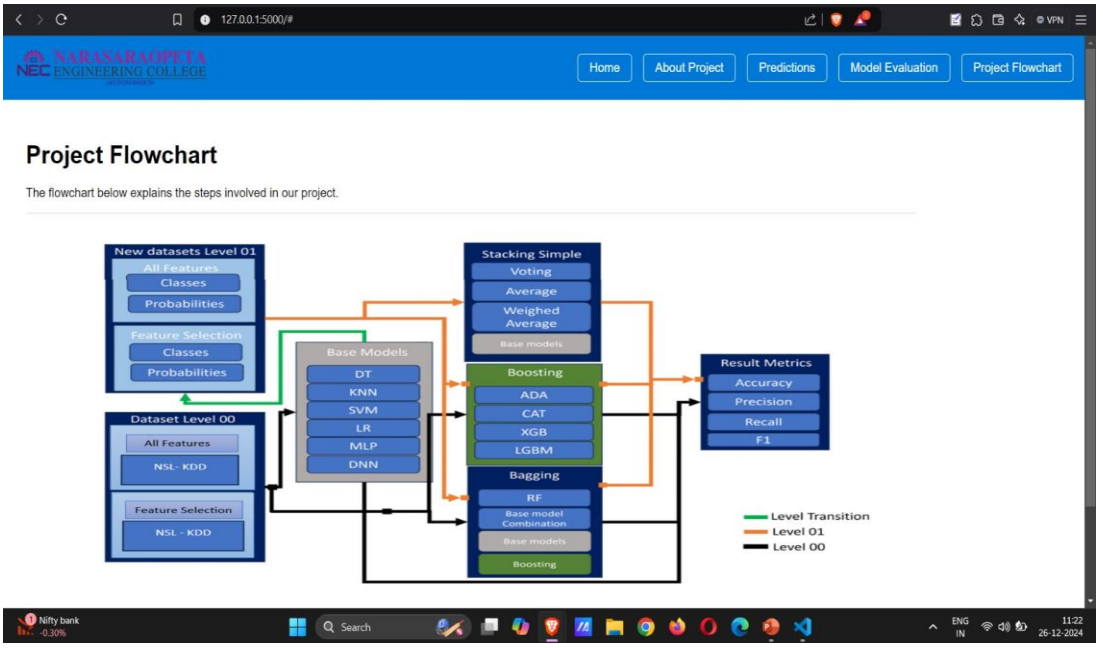

Fig 8.7 Project Flowchart

9. CONCLUSION

This project introduces a novel ensemble learning framework for intrusion detection systems (IDS) that combines bagging, boosting, and stacked knowledge distillation. By leveraging these techniques, the framework achieves high accuracy, precision, recall, and F1 scores while maintaining computational efficiency. Tested on the NSL-KDD dataset, it demonstrates improved performance and adaptability across different network environments. The publicly available source code encourages further research, with future directions including multi-level ensemble learning and integrating explainable AI for enhanced transparency.

10. FUTURE SCOPE

The future scope of *Boosting Network Intrusion Detection with Two-Level Ensemble Learning and Knowledge Distillation Approaches* includes expanding its application to IoT and edge computing, where resource efficiency is vital. The adaptable, scalable model can be optimized for 5G networks, cloud environments, and social media platforms, handling high-volume, real-time data. By integrating with threat intelligence systems and employing automated feature adaptation, it could also improve resilience against evolving cyber threats, establishing a robust defense mechanism across diverse network infrastructures.

11.REFERENCES

[1] Ioulianou, P., Vasilakis, V., Moscholios, I., & Logothetis, M. (2018). A signature-based intrusion detection system for the internet of things. Information and Communication Technology Form.

[2] Arreche, O., Bibers, I., & Abdallah, M. (2024). A Two-Level Ensemble Learning Framework for Enhancing Network Intrusion Detection Systems. IEEE Access.

[3] Aldallal, A. (2022). Toward efficient intrusion detection system using hybrid deep learning approach. Symmetry, 14(9), 1916.

[4] Kiourkoulis, S. (2020). DDoS datasets: Use of machine learning to analyse intrusion detection performance.

[5] Naidu, G., Zuva, T., & Sibanda, E. M. (2023, April). A review of evaluation metrics in machine learning algorithms. In Computer Science On line Conference (pp. 15-25). Cham: Springer International Publishing.

[6] Creech, G. (2014). Developing a high-accuracy cross platform host based intrusion detection system capable of reliably detecting zero-day attacks (Doctoral dissertation, UNSW Sydney).

[7] Fernandes, E. R. Q. (2018). Evolutionary ensembles for imbalanced learning (Doctoral dissertation, Universidade de S˜ao Paulo).

[8] Althaph, B., Sreenivasu, S. V. N., & Reddy, D. V. (2023, January). Student Performance Analysis with Ensemble Progressive Prediction. In 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT) (pp. 1513-1517). IEEE.

[9] Jain, D. K., Srinivas, K., Srinivasu, S. V. N., & Manikandan, R. (2021). Machine learning-based monitoring system with IoT using wearable sensors and pre-convoluted fast recurrent neural networks (P-FRNN). IEEE Sensors Journal, 21(22), 25517-25524.

[10] Srinivasu, S. V. Testing the intruder detection system created against aodv protocol using opnet.

Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches

Dr.S.V.N.Sreenivasu
Narasaraopeta Engineering College
Narasaraopeta(Autonomous)
drsvnsrinivasu@gmail.com

Macharla Bala Rangarao
Narasaraopeta Engineering College
Narasaraopeta(Autonomous)
balunani25@gmail.com

Bollavaram Venkata Srinivasulu
Narasaraopeta Engineering College
Narasaraopeta(Autonomous)
bollavaram.vasu@gmail.com

Boddupalli Venkata Siva Rama krishna
Narasaraopeta Engineering College
Narasaraopeta(Autonomous)
Sivaboddupalli932@gmail.com

T G Ramnadh babu
Narasaraopeta Engineering College
Narasaraopeta(Autonomous)
baburamnadh@gmail.com

***Abstract*—An advanced IDS framework to complement the inadequacies of the traditional methods dealing with complex and diversified network flows is proposed in this paper. The framework comprehensively solved the two traditional problems of data imbalance and accuracy detection by adopting two-level ensemble learning and knowledge distillation. The proposed system is tested with the NSL-KDD dataset, fusing several machine learning models to improve overall detection performance and leveraging knowledge distillation in transferring knowledge from an advanced complex model to an easy, simple, and computationally efficient one. These results prove significant improvement regarding the detection of both common and rare high-risk attacks; hence, the proposed IDS framework is truly robust and applicable for real-time applications in state-of-the-art network security.**

***Index Terms*—**

Two-Level Ensemble Learning, Stacked knowledge distillation, NSL-KDD Dataset, Stacking Ensemble, Real-Time Intrusion.

I. Introduction

With the increasing frequency and intensity of cyberattacks affecting business and personal data, network security has become much more important in the digital age. Considering these threats, which are becoming increasingly sophisticated and voluminous, traditional IDS solutions fail to provide insight into them. This work presents a new IDS framework based on a two-level ensemble learning approach combined with knowledge distillation. This ensembles the concept of several machine learning models within two-level ensemble learning, in order to enhance the capabilities of detection. Knowledge distillation transfers knowledge from complex, yet heavy models to simpler, more efficient ones. The proposed algorithm is tested on the NSL-KDD dataset and overcomes the challenges of imbalanced and irrelevant data. Thus, the detection of frequent and rare attacks would be more effective.

The proposed framework includes a wide range of base models, tending from support vector machines and decision trees to gradient boosting, whose outputs will be refined by the ensemble methods, including stacking and bagging. This framework then applies knowledge distillation to build a lightweight student model from a much more complex teacher model, achieving a good tradeoff between accuracy and computational efficiency. This boosts not only the performance of detection but also in solving the practical challenges of deployment. It presents a review of the contributions, which include a two-level ensemble learning framework, and an extensive evaluation by several models, and releases the source codes for further research and development in network intrusion detection[9][10].

II. Related Work

A. Prior Efforts in Utilizing XAI for IDS

In 2018, a new approach was introduced that transforms network traffic data into images and uses Convolutional Neural Networks (CNNs) to identify potential intrusions. This method proved to be highly accurate when tested on the NSL-KDD dataset, showcasing CNNs' ability to detect patterns in network traffic[1]. By 2022, advancements in AI further improve intrusion detection systems. Generative Adversarial Networks (GANs) were used to create synthetic data, while autoencoders helped extract key features. These techniques work together to significantly enhance the detection of network attacks, especially by using synthetic data to improve accuracy[2][8].

B. Stacked Knowledge Distillation for Intrusion Detection Systems (IDS)

SKD improves IDS by using knowledge from different teacher models—ConvNets, DNNs, RNNs, and LSTMs—to create smaller student models and a final combined model. This approach boosts detection accuracy and reduces false alarms. However, challenges include high computational costs and real-world testing. Future work will aim to cut down on

using resources and check if these solutions work well in real-life situations[4].

C. CONTRIBUTION OF THIS WORK

We list our specific contributions in five main points. Table 1 illustrates a sample plot showing the data trend.

Paper	Dataset	Base Model	Ensemble Model	Extensive Evaluation	Stacked knowledge Distillation
Our Work	NSL-KDD	LR, DNN,MLP,DT,SVm,KNN	ADA, LGBM,XGB,CAT,Bagging,Stacking,Boosting	Yes	yes
A stacking ensemble of deep learning models for IoT intrusion detection	GTCS	DNN, CNN, RNN, LSTM	DNN	No	No
Ensemble Classifiers for Network Intrusion Detection Using a Novel Network Attach Dataset	GTCS	J48, MLP, and IBK	Majority Voting	No	No
A novel ensemble learning-based model for network intrusion detection	KDD;99, CICIDS-2017, UNSW-NB15	Gaussian Naive Bayes LR, DT	SGD	No	No
Kistune	Kistune	Kitnet, GMM, SVM,DNN,Auto encoders	Kistune	No	No
Classification and Clustering Based Ensemble Techniques for Intrusion Detection Systems: A Survey	KDD'99, NSL-KDD, Kyoto 2006+, AWID Dataset, CICIDS-2017	DNN, SVM	Voting-based, Weighted majority voting	No	No
Ensemble Selection from Libraries of Models	ADULT, BACT, COD,CALHOUS ,COVTYPE,HSLETTER,P1	RF, NB,LR	Ensemble selection procedure	No	No
Ensemble Classifiers for Network Intrusion Detection System	KDD'99	LGP, ANFIS, RF	Weighed Voting	No	No
A Network Intrusion Detection System Using Ensemble Machine Learning	NSL-KDD,UNSW-NB15	Random Forest <AdaBoost,XG Boost Gradient boosting decision tree	Soft voting scheme	No	No
Network Intrusion Detection and Comparative Analysis Using Ensemble Machine Learning and Feature Selection	NSL-KDD,UNSW-NB15,and CICIDS2017	LR,DT,NB,NN,SVM	Majority Voting,DT,NB,LR,NN,SVM	No	No
Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection	KDD'99,NSL-KDD,UNSW-NB15,CICIDS-2017	DT,RF	CAT,LGBM,ADA,ExTra Trees,Voting,MFFSEM	No	No
Toward an Online Network Intrusion Detection System Based On Ensemble Learning	NSL-KDD,UNSW-NB15, Palo Alto network log	Autoencoder,SVM,RF	Weighed Voting	No	No
Ensemble-Learning Framework for Intrusion Detection to Enhance Internet of Things Devices	TON-IoT	DT,RF,LR,KNN	Stacking,Voting	No	No

Table 1: We compared our work with previous research on ensemble learning and stacked Knowledge Distillation for network intrusion detection, focusing on methods, datasets, and AI models.

Employing 20 machines with training and evaluation variations to further enhance the generalizability of the framework on varied feature selections and probabilities; conducting extensive experimentation, i.e., 84 evaluations on Level 00 and 168 each on Level 01 and false positive rates.

III. THE PROBLEM STATEMENT

In addition to defining several forms of intrusions such as normal traffic, port scans, DoS, brute force assaults, web attacks, infiltration, botnets, probing attacks, remote-to-local(R2L), and user-to-root(U2R), the problem statement underlines important challenges in network intrusion detection. These attacks use techniques like server overload and web application exploitation to take advantage of vulnerabilities. IDS systems need to use contemporary methods to increase detection accuracy and adjust to changing cyberattacks in order to properly handle these threats[5].

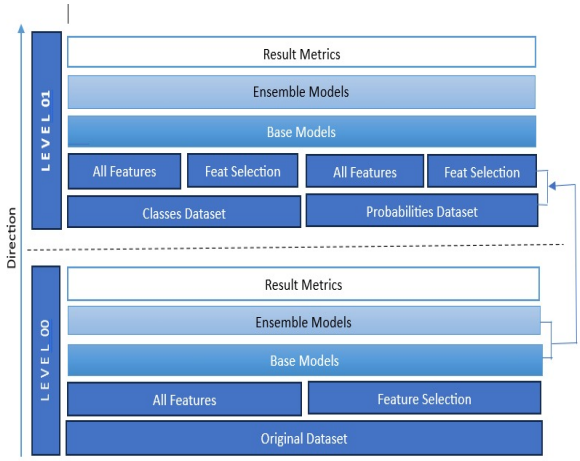

Fig. 1. A brief outline of our two-level ensemble learning approach for detecting network intrusions.

A. INTRUSION DETECTION SYSTEMS

IDS has recently turned out to be a crucial constituent in protecting critical infrastructure against complex network attacks. Conventional IDS techniques generally rely on detecting unauthorized actions by establishing boundaries against users' legitimate behavior. In recent times, with the advent of AI, the detection capability has increased multifold. Advanced state-of-the-art IDS models using AI do provide improved network intrusion detection in an automated way compared to earlier methods; therefore, these offer protection against cyber-attacks, which are becoming complex.

B. PREPROCESSING

Preprocessing on NSL-KDD will include cleaning and structuring of the data, naming the columns, checking for missing values, filtering out irrelevant records, encoding categorical features by one-hot, and encoding of the target variable with LabelEncoder. It is further required to divide the dataset into training and test subsets, and feature normalization with RobustScaler to unify scaling and reduce the effect of outliers; in that way, the proper preparation of the data for model training and testing will be done.

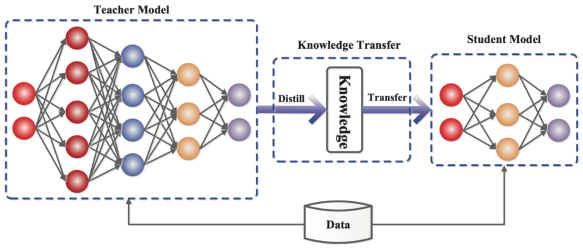

Fig. 2. The teacher-student framework for knowledge distillation[5].

C. DESCRIPTION OF DATASET

The NSL-KDD dataset is an improved version of the KDD dataset. Developed by the University of New Brunswick and the National Research Council of Canada. This dataset is generated using a set of network traffic data including attack types and common patterns[6]. In this topic, training uses a subset of KDDTrain+, and testing and evaluation use a subset of KDDTest+, taken from Kaggle. A summary of dataset size, attack type, and attack features is shown in Table 3.

D. EXPERIMENTATIONAL SETUP

A high-performance computing system (HPC) configured with 64 GPU-accelerated nodes, each featuring 256 GB of RAM, four NVIDIA GeForce RTX 4090 GPUs, and a 64-core AMD EPYC 7773X processor was utilized for the research. Designed for AI and machine learning tasks, the system offers a theoretical peak performance of around 10 petaFLOPs with a power draw of 300 watts. Python served as the implementation language, with libraries like Pandas and Matplotlib used in conjunction with advanced AI frameworks such as TensorFlow and PyTorch.

IV. FRAMEWORK

This effort is targeted at developing the ensemble learning and stacked knowledge distillation pipelines, which offer network attack detection and classification metrics on the higher side for better management and intrusion prevention by the security analysts. Various components of the framework are as described in Figures 1, 2, 3, and Table 4.

A. BLOCKS OF LEVEL 00:

The NSL-KDD dataset is used for feature selection at Level 00 in the ensemble learning pipeline. This dataset feeds a variety of base and ensemble models, including ADA, SVMs, GB, and bagged models like BaggingADA and BaggingRF. These models categorize network traffic into several intrusion categories, such as port scanning and denial-of-service attacks. Metrics including runtime, F1 score, accuracy, recall, precision, and recall are used to analyze their performance and determine how good they are at intrusion detection and categorization.

B. LOW-LEVEL ENSEMBLE LEARNING PIPELINE COMPONENTS:

The low-level Ensemble Learning Pipeline can be divided into two major steps: Level 00 and Level 01. The Level 00 pipeline initiates with the loading of the NSL-KDD dataset destined for intrusion detection. At Level 01, it utilizes outputs from Level 00 models in order to create new datasets, which are sorted into four types of probabilities: all features, selected features, class labels of all features, and selected features. The different elements of the pipeline are color-coded: Level 00 processes in black, Level 01 in orange, and transitions between the levels in green.

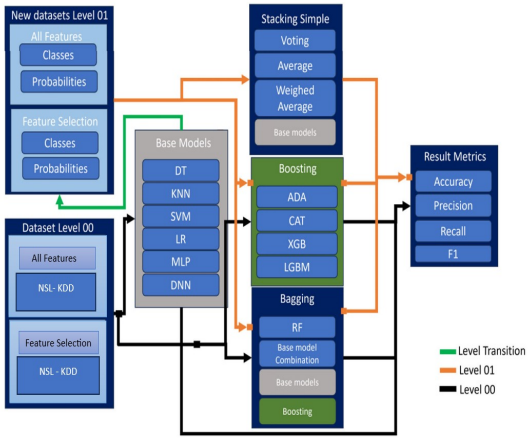

Fig. 3. A brief look at our Ensemble Learning framework for network intrusion detection, highlighting its use of diverse AI models and datasets.

C. FEATURE SELECTION-LEVEL 00|LEVEL 01

Level 00 feature selection outperforms more conventional techniques like information gain and chi-square by highlighting the top eight features of the NSL-KDD dataset using SHAP. Using this method, features are ranked in several models and then combined. Without the complexity of XAI techniques, Level 01 streamlines the process and offers a fast measure of entropy reduction by filtering datasets using information gain to keep the top five pertinent features.

D. BASE LEARNERS AI MODELS - LEVEL 00:

Split the already preprocessed dataset into Level 00 with 70% for the training purpose and 30% to be used for the test set. Apply six AI classification models: DT, LR, DNN, MLP, KNN, and SVM, where the best parameters for each model will be selected in order to maximize the performance of each unique model.

E. AI MODELS - ENSEMBLE LEARNERS - LEVEL 00:

Boosting and Bagging were attempted at Level 00 as a part of the efficiency analysis of ensemble learning. Continue with the same pipeline, splitting the dataset into a ratio of 70% for training and 30% for testing. Boosting methods tried-four most popular variants are: **LightGBM (LGBM), Adaptive Boosting (AdaBoost), eXtreme Gradient Boosting (XGB), CatBoosting (CAT)**.

For Bagging, we utilize 10 different variations:

Random Forest (RF) Bagging with RF Combination of different learners: (DT, KNN, MLP, LR, SVM, ADA, DNN, MLP, XGB) **within a single Bagging algorithm Bagging of Boosting algorithms:** (ADA, CAT, LGBM) **Bagging of base learners:** (DT, KNN, MLP, LR, SVM)

Each model is tuned with specific parameters in order to get optimum performance - see the proposed model in Section V for more information.

Table 2: Summary of key features of the NSL-KDD dataset[3].

NSL-KDD Features	Explanation
duration	Length of the connection
protocol_type	Type of the protocol (e.g., TCP, UDP)
service	Network service on the destination (e.g., HTTP, FTP)
flag	Normal or error status of the connection
src_bytes	Number of data bytes from source to destination
dst_bytes	Number of data bytes from destination to source
logged_in	1 if successfully logged in; 0 otherwise
count	Number of connections to the same host as the current connection in the past two seconds
srv_count	Number of connections to the same service as the current connection in the past two seconds
dst_host_same_srv_rate	Rate of connection to the same service on destination host
dst_host_srv_count	Count of connections to the same service on destination host
dst_host_same_src_port_rate	Rate of connections to same source port on destination
dst_host_serror_rate	Rate of connections that have activation flags indicating various types of errors
dst_host_rerror_rate	Rate of connections that have rejected flags

F. ENSEMBLE LEARNERS AI MODELS - LEVEL 01:

In Level 01, we keep using Boosting and Bagging techniques for computing the performance of ensemble learning, respecting the same strategy in splitting: 70% for training and 30% for testing. We have used the same four popular Boosting techniques like in Level 00. Furthermore, in Level 01, we have added stacking techniques since each model can be considered a variation of stacking from Level 00. Stacking piles up the predictions of base-learners inputs to a meta-model; thus, each model of Level 01 is a meta-model. We implemented nine different stacking techniques:

Simple Stacking: (including Voting, Averaging, and Weighted Average) **Stacking with all base models:** (DT, KNN, LR, SVM, DNN, MLP) **Combination of different learners:** (DT, KNN, LR, SVM, ADA, DNN, MLP, XGB) **in a single Bagging algorithm Bagging of Boosting algorithms:** (ADA, CAT, LGBM) **Bagging of base learners:** (DT, KNN, MLP, LR, SVM)

G. SELECTION CRITERIA FOR AI MODELS:

These models were chosen for this research based on their frequent usage with regard to related studies of IDS from references [5] and [3]. Limiting the model selection to these three allows for evaluating how XAI-based feature selection impacts the performance of the chosen models. This approach ensures compatibility and facilitates a comparison analysis within the framework of two-level ensemble learning techniques.

H. TOP INTRUSION FEATURES AND THEIR USE IN ENSEMBLE LEARNING AND KNOWLEDGE DISTILLATION

The paper lists the essential network intrusion features from the NSL-KDD dataset that are crucial for the conducted analysis. These features and their descriptions are given in Table 2 to depict their significance, regarding the ensemble learning methods discussed in Section V.

I. KNOWLEDGE DISTILLATION - TEACHER MODEL COMPONENTS

1) Model Selection: XGBoost, CatBoost, Random Forests, GBMs, and other ensemble approaches will be used to choose the best Teacher model, and GridSearchCV will be used to

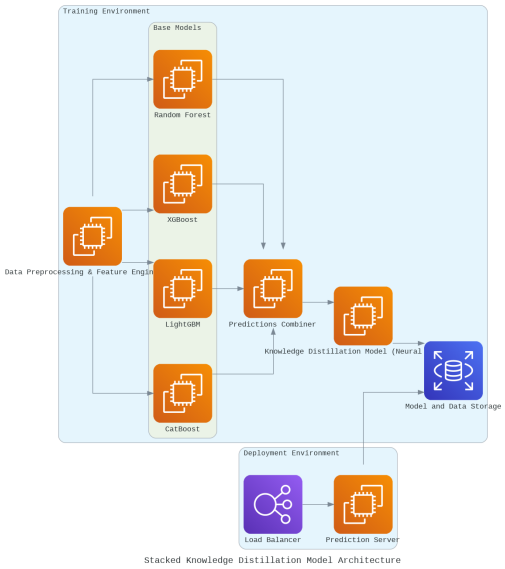

Fig. 4. Stacked Knowledge Distillation Model Architecture.

adjust the model's hyperparameters. Metrics such as precision, recall, F1-score, and AUC-ROC will be employed to evaluate the performance of the models. By using knowledge distillation to train a lighter Student model with the soft labels provided by the outputs, high-performance models can be used in resource-constrained contexts.

We have integrated four popular Teacher methods: **Random Forest (rf)**, **XGBoost (xgb)**, **LightGBM (lgbm)**, **CatBoost (catboost)**.

2) Training the Teacher Model: This is done by first training the Teacher model on the whole training dataset, using different hyperparameter tunning techniques; Grid Search is one of them. It involves a fit of the model to the data so that the learning of the model would be effective and achieve optimum results.

3) Generating Soft Labels: These soft labels are probability distributions across all classes, rather than a single predicted class, and are obtained after training by using the *predict_proba* method.

J. STUDENT MODEL COMPONENTS IN KNOWLEDGE DISTILLATION

The student model is the simpler, more efficient cousin of the teacher model.

1) Model Selection: The Student model is usually a much simpler and smaller model; a decision tree, logistic regression, or a shallow neural network would be selected so as to optimize reduced computational complexity and model size against the capability of maintaining predictive performance.

2) Training The Student Model: It trains the Student model both from the original training data and also from the soft labels that the Teacher model predicts, which helps transfer the knowledge captured by the Teacher model in its predictions to the Student model, enhancing its generalization capability to perform similarly. Knowledge Distillation Process:

3) During knowledge distillation: The Student model is trained to replicate the Teacher model's output. To do so, one uses a custom loss function that has to balance cross-entropy loss with respect to the true labels and the distillation loss with respect to the soft predictions of the Teacher model.

We have implemented one student method: **distillation_model**

V. FOUNDATIONS OF EVALUATION

We utilize five key metrics: **Accuracy**, **Precision**, **Recall**, **F1 Score**, and **False Positive Rate**. These metrics are essential for conducting a thorough analysis and comparison of model performance.

(a)Basic statistics of dataset

Dataset	No. of Labels	No. of Features	No. of Samples
NSL-KDD	5	41	148,517

(b) Distribution of samples among differen attach types

Dataset	Normal	DoS	PortScan	Brute Force	Web	Bot	Infilt.	Probe	U2R	R2L
NSL-KDD	48%	-	-	-	-	-	-	10.05%	0.47%	6.48%

Table 3: A summary of the NSL-KDD network intrusion dataset used in this work, covering the dataset size, the variety of attack types (labels), the number of intrusion features, and the distribution of samples across different attack categories.

VI. EVALUATION RESULTS

A. Main Results of Level 00 :

All Features					Feature Selection				
Models	ACC-00	PRE-00	REC-00	F1-00	Models	ACC-00	PRE-00	REC-00	F1-00
LR	0.89	0.87	0.86	0.86	LR	0.89	0.86	0.85	0.86
DT	0.97	0.92	0.90	0.90	DT	0.97	0.91	0.90	0.90
RF	1.00	1.00	1.00	1.00	RF	1.00	1.00	1.00	1.00
KNN	0.99	0.99	0.99	0.99	KNN	0.99	0.98	0.98	0.99
SVM	0.90	0.81	0.90	0.85	SVM	0.90	0.82	0.91	0.84
GB	1.00	1.00	1.00	1.00	GB	1.00	1.00	1.00	1.00
MLP	0.99	1.00	1.00	1.00	MLP	0.99	1.00	1.00	1.00
XGB	1.00	1.00	1.00	1.00	XGB	1.00	1.00	1.00	1.00
LGBM	0.91	0.91	0.91	0.91	LGBM	0.91	0.90	0.91	0.91
Bag_cat	1.00	1.00	1.00	1.00	Bag_cat	1.00	1.00	1.00	1.00
NB	0.51	0.76	0.51	0.46	NB	0.51	0.74	0.51	0.46
LDA	0.97	0.98	0.97	0.98	LDA	0.97	0.96	0.95	0.97
QDA	0.94	0.94	0.94	0.94	QDA	0.94	0.93	0.93	0.94
PA	0.91	0.74	0.54	0.38	PA	0.91	0.73	0.54	0.38
Bag_ada	1.00	1.00	1.00	1.00	Bag_ada	1.00	1.00	1.00	1.00
Ridge_C	1.00	1.00	1.00	0.99	Ridge_C	1.00	1.00	1.00	0.99
Stacking	1.00	1.00	1.00	1.00	Stacking	1.00	1.00	1.00	1.00
Voting	1.00	1.00	1.00	1.00	Bagging	1.00	1.00	1.00	1.00
ADA	1.00	1.00	1.00	1.00	ADA	1.00	1.00	1.00	1.00
DNN	0.92	0.90	0.92	0.92	DNN	0.92	0.89	0.91	0.91
Bag_DT	1.00	1.00	1.00	1.00	Bag_DT	1.00	1.00	1.00	1.00

Table 4: NSL-KDD Level 00: In the above table all features are used (i.e., left-hand side). We observe that Bag_DT achieved the best scores overall, followed by ADA. However, when feature selection is applied (i.e., right-hand side), we observe that Bag_DT achieved the best scores overall, followed by ADA.

Following Table 4, in general, the use of all features are done with the best metrics: accuracy, precision, recall, and F1 score, compared to the top five. Among the best models, there are stacking Bagging with Decision Trees, AdaBoost, XGBoost, and Bagging with CatBoost, embracing Bag DT and Bag Cat. While the ones that performed worse were Naive Bayes, Logistic Regression, and Support Vector Machines. In all, leveraging all the features from the NSL-KDD dataset yields better results; there is further room for improvement by the ensemble models at Level 01.

B. Main Results of Level 01:

At Level 01, base learners and different ensemble models were evaluated against performance using the NSL-KDD dataset. The models which were trained based on predicted probabilities turned out to be much better than those trained on predicted classes. The ranking among the datasets of F1 scores, in order from highest, includes probabilities using all features and probabilities with feature selection, followed by predicted classes using all features and, finally, predicted classes using feature selection. The different best configurations were Bagging with DT, AdaBoost, Stacking, Voting, and XGBoost (XGB). Overall, performances are better with all features, but results might vary.

Feature Selection				
Models	ACC-01	PRE-01	REC-01	F1-01
Bag_DT	1.00	1.00	1.00	1.00
ADA	1.00	1.00	1.00	1.00
Stacking	1.00	1.00	1.00	1.00
Voting	1.00	1.00	1.00	1.00
XGB	1.00	1.00	1.00	1.00
Bag_ada	1.00	1.00	1.00	1.00
Bag_Cat	1.00	1.00	1.00	1.00
GB	1.00	1.00	1.00	1.00
RF	1.00	1.00	1.00	1.00
Ridge_c	1.00	1.00	1.00	0.99
MLP	0.99	1.00	1.00	1.00
KNN	0.99	0.99	0.99	0.98
LDA	0.97	0.98	0.97	0.97
DT	0.97	0.92	0.90	0.96
QDA	0.94	0.94	0.94	0.94
DNN	0.92	0.90	0.92	0.92
LBGM	0.91	0.91	0.91	0.91
PA	0.91	0.74	0.54	0.87
SVM	0.90	0.81	0.90	0.86
LR	0.89	0.87	0.86	0.84
NB	0.51	0.76	0.51	0.48

Table 5: NSL-KDD Level 01: Evaluation metrics(ACC,PRE,REC,F1). Organized by F1 (highest to lowest). In this case, Bag_DT achieved the best scores overall.

C. Comparison of Level 01 and Level 00:

Comparing Level 01 with Level 00, great improvements in performance are achieved by the addition of stacking and boosting methods. Model performances that use all features and prediction probabilities at Level 01 outperform the topmost models in Level 00, such as Bagging-DT, where the top performers now include such models as Bagging-DT, AdaBoost, Stacking, XGBoost, Bagging-Ada, Bagging-Cat, GB, and RF. Also, F1 scores in Level 01 are touching or surpassing 1.00, outperforming the best score of the Level 00 models. Therefore, these results mean that using the prediction probabilities instead of predicted classes significantly enhances model performance for the NSL-KDD dataset .

D. Main Results of the Stacked Knowledge Distillation:

For the Stacked Knowledge Distillation model, results included 100% in total on the validation accuracy of the test

set, while the results were 1.01 for validation loss, which speaks much about the performance aligning to the actual labels. Precision, recall, and F1 scores were quite high for most classes, hence capturing much about the predictive power in teacher models. These results demonstrate how Knowledge Distillation yields lightweight models, efficient to deploy in resource-constrained environments with no compromise in accuracy, from the results depicted in Tables 7 and Figures 5, 6, and 7.

Base Model Results:

INDEX	MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
0	rf	0.9996824489536693	0.9996850572873647	0.9996824489536693	0.9996514774993354
1	xgb	1.00	1.00	1.00	1.00
2	lgbm	0.9065129719602426	0.9134068020999274	0.9065129719602426	0.9099315578659155
3	catboost	1.00	1.00	1.00	1.00

Distillation Model Results:

INDEX	MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
0	Distillation Model	1.00	1.00	1.00	1.00

Table 6: Stacked Distillation Model Results.

E. Gains of Our Framework:

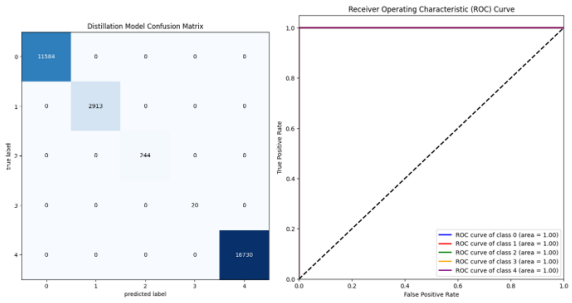

Fig5:Distillation Model Confusion Matrix Fig6: Receiver Operating Characteristic Curve

Our framework shows significant improvement when transitioning to Level 01 with the "Probabilities-All Features" setup, as demonstrated by the model's performance on the NSL-KDD dataset. Precision, Recall, and F1 score all increased to 1.00 from earlier measured values., the FPR decreased significantly from 2.142% to 1.002%. Notably, the accuracy has increased also from 0.991 to 1.00. As a matter of fact, all top-scoring ensemble models in Level 01 had perfect scores across the board in all metrics. This proves that our proposed two-level ensemble learning and stacked knowledge distillation framework brings significant performance improvements in multiple metrics.

VII. LIMITATIONS, DISCUSSION, AND FUTURE DIRECTIONS

Integrating multiple machine learning models to improve it Accuracy and false positive reduction, the team suggested framework greatly improves intrusion detection systems (IDS) and refers to the NSL-KDD dataset available. Nevertheless, it has drawbacks like bias in the dataset, excessive processing overhead, and problems with scalability in dynamic contexts.

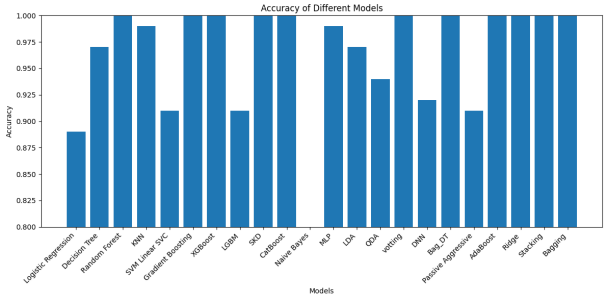

Fig 7: The overall summary of the results of this work shows that our system meets good performance criteria and reduces false positive rate (FPR).

Subsequent investigations ought to encompass a range of datasets, investigate expedited techniques for instantaneous detection, contemplate multi-layered learning to enhance precision and integrate Explainable AI to yield more lucid findings. Real-world testing and useful improvements will require cooperation with industry partners.

VIII. CONCLUSION

This study introduces a novel ensemble learning framework for intrusion detection systems (IDS) that combines bagging, boosting, and stacked knowledge distillation. By leveraging these techniques, the framework achieves high accuracy, precision, recall, and F1 scores while maintaining computational efficiency. Tested on the NSL-KDD dataset, it demonstrates improved performance and adaptability across different network environments. The publicly available source code encourages further research, with future directions including multi-level ensemble learning and integrating explainable AI for enhanced transparency.

REFERENCES

[1] Ioulianou, P., Vasilakis, V., Moscholios, I., & Logothetis, M. (2018). A signature-based intrusion detection system for the internet of things. Information and Communication Technology Form.

[2] Arreche, O., Bibers, I., & Abdallah, M. (2024). A Two-Level Ensemble Learning Framework for Enhancing Network Intrusion Detection Systems. IEEE Access.

[3] Aldallal, A. (2022). Toward efficient intrusion detection system using hybrid deep learning approach. Symmetry, 14(9), 1916.

[4] Kiourkoulis, S. (2020). DDoS datasets: Use of machine learning to analyse intrusion detection performance.

[5] Naidu, G., Zuva, T., & Sibanda, E. M. (2023, April). A review of evaluation metrics in machine learning algorithms. In Computer Science Online Conference (pp. 15-25). Cham: Springer International Publishing.

[6] Creech, G. (2014). Developing a high-accuracy cross platform host-based intrusion detection system capable of reliably detecting zero-day attacks (Doctoral dissertation, UNSW Sydney).

[7] Fernandes, E. R. Q. (2018). Evolutionary ensembles for imbalanced learning (Doctoral dissertation, Universidade de São Paulo).

[8] Althaph, B., Sreenivasu, S. V. N., & Reddy, D. V. (2023, January). Student Performance Analysis with Ensemble Progressive Prediction. In 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT) (pp. 1513-1517). IEEE.

[9] Jain, D. K., Srinivas, K., Srinivasu, S. V. N., & Manikandan, R. (2021). Machine learning-based monitoring system with IoT using wearable sensors and pre-convoluted fast recurrent neural networks (P-FRNN). IEEE Sensors Journal, 21(22), 25517-25524.

[10] Srinivasu, S. V. Testing the intruder detection system created against aodv protocol using opnet.

ab1

ORIGINALITY REPORT

6%	3%	5%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Osvaldo Arreche, Tanish R. Guntur, Jack W. Roberts, Mustafa Abdallah. "E-XAI: Evaluating Black-Box Explainable AI Frameworks for Network Intrusion Detection", IEEE Access, 2024 Publication	2%
2	Submitted to University of Northumbria at Newcastle Student Paper	1%
3	link.springer.com Internet Source	1%
4	www.mdpi.com Internet Source	1%
5	Hideki Oki, Motoshi Abe, Jyunichi Miyao, Takio Kurita. "Triplet Loss for Knowledge Distillation", 2020 International Joint Conference on Neural Networks (IJCNN), 2020 Publication	<1%
6	Submitted to University of Queensland Student Paper	<1%

7	www.coursehero.com Internet Source	<1%
8	researchr.org Internet Source	<1%
9	E. Silambarasan, Rajashree Suryawanshi, S. Reshma. "Enhanced cloud security: a novel intrusion detection system using ARSO algorithm and Bi-LSTM classifier", International Journal of Information Technology, 2024 Publication	<1%
10	bura.brunel.ac.uk Internet Source	<1%
11	ebin.pub Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

Centurion University
Shaping Lives... Empowering Communities...

SCOPES-2024

IEEE KOLKATA SECTION & BHUBANESWAR SUBSECTION

19th - 21th December 2024

Certificate of Presentation

This is to certify that **Dr.S.V.N.Sreenivasu**, M.Tech., Ph.D., Dean R&D, Professor from **NARASARAOPETA ENGINEERING COLLEGE** has presented a paper titled **"Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches."** in the 2nd International Conference on Signal Processing, Communication, Power, and Embedded Systems (SCOPES), technically co-sponsored by the IEEE Kolkata Section and Bhubaneswar Sub-Section (IEEE-approved conference record number #64467), **organized by the Department of Electronics & Communication Engineering, School of Engineering & Technology, Centurion University of Technology and Management, Paralakhemundi, Odisha** during December 19-21, 2024.

Prof. Prafulla Kumar Panda	Prof. Ashok Misra	Prof. Debendra Kumar Sahoo	Prof. Anita Patra
Programme Chair	Convener	Organizing Chair	General Chair

Centurion University
Shaping Lives... Empowering Communities.

SCOPES-2024

IEEE KOLKATA SECTION & BHUBANESWAR SUBSECTION

19th - 21th December 2024

Certificate of Presentation

This is to certify that **Mr. MACHARLA BALA RANGARAO** from **NARASARAOPETA ENGINEERING COLLEGE** has presented a paper titled **"Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches."** in the 2nd International Conference on Signal Processing, Communication, Power, and Embedded Systems (SCOPES), technically co-sponsored by the IEEE Kolkata Section and Bhubaneswar Sub-Section (IEEE- approved conference record number #64467), **organized by the Department of Electronics & Communication Engineering, School of Engineering & Technology, Centurion University of Technology and Management, Paralakhemundi, Odisha** during December 19-21, 2024.

Prof. Prafulla Kumar Panda	Prof. Ashok Misra	Prof. Debendra Kumar Sahoo	Prof. Anita Patra
Programme Chair	Convener	Organizing Chair	General Chair

Centurion University
Shaping Lives... Empowering Communities...

SCOPES-2024

IEEE KOLKATA SECTION & BHUBANESWAR SUBSECTION

19th - 21th December 2024

Certificate of Presentation

This is to certify that **Mr. BODDUPALLI VENKATA SIVA RAMA KRISHNA** from **NARASARAOPETA ENGINEERING COLLEGE** has presented a paper titled **"Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches."** in the 2nd International Conference on Signal Processing, Communication, Power, and Embedded Systems (SCOPES), technically co-sponsored by the IEEE Kolkata Section and Bhubaneswar Sub-Section (IEEE-approved conference record number #64467), **organized by the Department of Electronics & Communication Engineering, School of Engineering & Technology, Centurion University of Technology and Management, Paralakhemundi, Odisha** during December 19-21, 2024.

Prof. Prafulla Kumar Panda	Prof. Ashok Misra	Prof. Debendra Kumar Sahoo	Prof. Anita Patra
Programme Chair	Convener	Organizing Chair	General Chair

Centurion University
Shaping Lives... Empowering Communities...

SCOPES-2024

IEEE KOLKATA SECTION & BHUBANESWAR SUBSECTION

19th - 21th December 2024

Certificate of Presentation

This is to certify that **Mr. BOLLAVARAM VENKATA SRINIVASULU** from **NARASARAOPETA ENGINEERING COLLEGE** has presented a paper titled **"Boosting Network Intrusion Detection With Two-Level Ensemble Learning And Knowledge Distillation Approaches,"** in the 2nd International Conference on Signal Processing, Communication, Power, and Embedded Systems (SCOPES), technically co-sponsored by the IEEE Kolkata Section and Bhubaneswar Sub-Section (IEEE-approved conference record number #64467), **organized by the Department of Electronics & Communication Engineering, School of Engineering & Technology, Centurion University of Technology and Management, Paralakhemundi, Odisha** during December 19-21, 2024.

Prof. Prafulla Kumar Panda	Prof. Ashok Misra	Prof. Debendra Kumar Sahoo	Prof. Anita Patra
Programme Chair	Convener	Organizing Chair	General Chair