Enhancing Cloud Security with Deep Learning - An ANN Approach for Cyber Threat Detection

A project Report submitted in the partial fulfilment of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Bojja Sobharani	(21471A0509)
Gattu Thanuja	(21471A0522)
Thalapaneni Suma Durga	(21471A0564)

Under the esteemed guidance of

M Sathyam Reddy, M.Tech

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASARAOPET (AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under Tier -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, YALAMAND VILLAGE, NARASARAOPET – 522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE: NARASARAOPET

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"Enhancing Cloud security with Deep Learning- An ANN Approach for Cyber Threat Detection"** is a bonafide work done by the team **B. Sobharani(21471A0509), G. Thanuja(21471A0522), T. Suma Durga(21471A0564),** in partial fulfillment of requirements for the award of the degree of Bachelor of Technology in the Department of COMPUTER SCIENCE AND ENGINEERING, during the academic year 2024- 2025.

PROJECT GUIDE   **PROJECT CO-ORDINATOR**

M Sathyam Reddy,M.Tech   **D Venkata Reddy,**M.Tech,(Ph.D)

Assistant Professor   **Assistant Professor**

HEAD OF THE DEPARTMENT   **EXTERNAL EXAMINER**

Dr.S.N. Tirumala Rao,M.Tech, Ph.D

Professor & HOD

DECLARATION

We declare that this project work titled "Enhancing Cloud security with Deep Learning-An ANN Approach for Cyber Threat Detection" 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.

B. Sobharani   (21471A0509)

G. Thanuja   (21471A0522)

T. Suma Durga (21471A0564)

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 this course. 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 **M Sathyam Reddy, M.Tech.,** of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **D Venkata Reddy,M.Tech.,(Ph.D.),** Assistant 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

B. Sobharani (21471A0509)

G. Thanuja   (21471A0522)

T. Suma Durga(21471A0564)

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

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.

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.

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.

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.

NARASARAOPETA ENGINEERING COLLEGE (AUTONOMOUS)

Project Course Outcomes (CO'S):

CO421.1: Analyse 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	PSO1	PSO2	PSO3
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	PSO1	PSO2	PSO3
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 recognizing image manipulations using CNN and ELA	PO1, PO3
CC421.1, C2204.3, C22L3.2	Each and every requirement is critically analyzed, the process model 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	Implementation is done and the project will be handled by the social media users and in future updates in our project can be done based on detection of forged videos	PO4, PO7
C32SC4.3	The physical design includes website to check whether an image is real or fake	PO5, PO6

ABSTRACT

Due to the increasing role of cloud computing in the growth of trustable digital services, secure and efficient solutions will be a must in the mitigation of cyber risks. This paper presents an approach to the enhancement of cloud security through the application of deep learning, focusing on Artificial Neural Networks. Moderate segments of ANN algorithms have been performed that are Levenberg-Marquardt (LM), Scaled Conjugate Gradient (SCG), and Bayesian Regularization (BR), to build an enhanced threat detection system which has the capability of detecting highly complex attacking patterns in the cloud. These algorithms are selected for their efficiency in training, scalability, and robustness in handling noisy data. The system is designed to adapt to evolving cyberattacks, offering a proactive security approach. The incorporation of such models ensures real-time threat detection and prevention. The paper also looks at the problem of various types of cyber threat detection—malware, phishing, distributed denial of service (DDoS) attacks, etc. Finally, it can be concluded that employing the knowledge of ANN, and other, deep learning approaches, can facilitate the development of persistent cyber defense structures in the cloud. The research shows how ANN models can contribute to a faster and easier detection of new, unclassified threats, therefore increasing the overall security of cloud-related activities.

INDEX

LIST OF FIGURES

TABLE   **TABLE CAPTION**   **PAGE NO**

1. INTRODUCTION

The emergence of cloud computing has revolutionized the way individuals and organizations store, manage, and access information. By providing unmatched scalability, flexibility, and cost- efficiency,[1] cloud computing has become the backbone of modern digital infrastructure. Its ability to seamlessly integrate and share resources across the globe has driven innovation across various industries, including healthcare, finance, education, and e-commerce. The benefits of cloud computing include reduced operational costs, enhanced collaboration, improved data accessibility, and the ability to scale resources dynamically based on demand[2].

However, with these significant advancements come notable challenges, particularly in terms of security. As sensitive data, intellectual property, and critical operations increasingly migrate to cloud systems, these centralized architectures become prime targets for cybercriminals. The dynamic and interconnected nature of the cloud introduces various vulnerabilities that attackers can exploit [3], including unauthorized data access, data breaches, denial-of-service (DoS) attacks, and sophisticated malware infiltrations. Addressing these security concerns requires robust, adaptive, and intelligent solutions that can predict, prevent, and respond to evolving cyber threats in real time [4].

The centralized architecture of cloud computing systems makes them attractive targets for attackers. Cybercriminals constantly devise new strategies to exploit weaknesses in cloud infrastructure. Common attack methods include phishing attacks, DDoS attacks, ransomware, and data leakage incidents. Phishing attacks involve deceptive emails or messages that trick users into revealing sensitive information [5]. DDoS attacks flood a cloud network with excessive traffic, overwhelming the system and rendering it inaccessible. Ransomware encrypts critical data, forcing victims to pay a ransom to regain access, while data leakage can occur through misconfigurations or insider threats, exposing sensitive information to unauthorized users [6].

The shared responsibility model in cloud security further complicates the task of mitigating these threats. While cloud service providers manage infrastructure security, customers are responsible for securing their data, applications, and access controls [7]. This division often leads to gaps in security coverage, creating opportunities for attackers to exploit vulnerabilities. Traditional security approaches such as firewalls, antivirus software, and signature-based detection systems often struggle to identify zero-day exploits and polymorphic malware, which

use advanced techniques to bypass detection. Consequently, there is an increasing demand for innovative and intelligent security solutions capable of mitigating advanced cyber threats in cloud environments [8].

Artificial Neural Networks (ANNs) have emerged as a powerful tool in the realm of cybersecurity, particularly for enhancing cloud security. ANNs are inspired by the structure and functionality of the human brain, consisting of interconnected layers of neurons that process data and extract meaningful patterns [9]. This unique architecture makes ANNs highly effective in identifying complex and non-linear relationships within large datasets, which is crucial for detecting cyber threats. Unlike traditional rule-based security systems, ANNs can adapt by learning from previous attack patterns, allowing them to generalize and detect previously unseen threats, such as zero-day attacks [10]. Their ability to recognize subtle anomalies in network traffic, log files, and user behavior enhances their ability to detect malicious activities. By continuously analyzing data streams from cloud environments, ANNs can identify suspicious behavior and trigger preventive measures in real time. Additionally, ANNs reduce reliance on manual intervention by automating threat detection and response processes, enhancing security operations' efficiency [11].

ANNs have several practical applications in cloud security. For instance, they can power Intrusion Detection Systems (IDS) that analyze network traffic patterns to detect potential intrusions or unauthorized access attempts. They are also valuable in malware detection, where they can identify both known and unknown malware variants based on their behavior. Furthermore, ANNs are effective in anomaly detection, where they monitor cloud infrastructure and identify deviations from normal behavior, helping to prevent data breaches and insider threats [12].

Deep learning, a specialized subset of machine learning, significantly enhances the capabilities of ANNs by incorporating multiple hidden layers[13]. Deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have proven highly effective in analyzing vast volumes of data for threat detection. CNNs are particularly suitable for detecting malicious patterns in network traffic data, packet sequences, and file content. On the other hand, RNNs excel at identifying behavioral anomalies over time by analyzing sequential data, such as user access logs and server interactions. Autoencoders, an unsupervised learning technique, are effective for anomaly detection by learning data patterns and identifying deviations from typical behavior [14]. By combining these techniques with supervised

2

and unsupervised learning approaches, deep learning models can detect both known attack types and emerging threats that bypass traditional security mechanisms [15].

The evolving nature of cyber threats necessitates continuous research and development in cloud security. Future advancements may include the development of hybrid AI models that combine ANNs with other machine learning techniques such as Support Vector Machines (SVMs), Decision Trees, and Random Forests to enhance threat detection accuracy. Integrating real-time threat intelligence feeds can further improve the effectiveness of deep learning models by keeping them updated on emerging attack patterns[16]. Enhanced data privacy solutions such as homomorphic encryption and differential privacy can provide additional security for AI-driven systems. Moreover, ethical considerations must be addressed to ensure transparency, fairness, and accountability in AI decision-making, ensuring wider adoption and trust in these security models [17].

In addition to technical advancements, organizations must also focus on building a security-conscious culture among employees. Regular training sessions on cybersecurity best practices, secure coding techniques, and threat awareness can significantly reduce human errors that often lead to security breaches. By promoting security awareness and ensuring that employees adhere to secure protocols, organizations can create an added layer of protection against cyber threats, complementing the efforts of AI-based security frameworks[18].

The integration of Artificial Neural Networks and deep learning into cloud security frameworks presents a transformative approach to safeguarding cloud environments [19]. By leveraging their adaptability, pattern recognition capabilities, and automated threat response mechanisms, these intelligent models provide organizations with the tools to proactively defend against sophisticated cyber threats. As cloud computing continues to expand, investing in AI-driven security solutions will be vital in ensuring the integrity, confidentiality, and availability of data in the digital landscape. Ultimately, combining deep learning techniques with existing security frameworks can create a robust defense mechanism, protecting businesses and individuals from the ever-growing threat of cyberattacks [20].

To further strengthen cloud security, collaboration between cloud service providers, cybersecurity experts, and regulatory bodies is crucial. Establishing comprehensive security standards, developing best practices, and sharing threat intelligence can help organizations stay

ahead of emerging threats [21]. Encouraging open communication and cooperation can foster a proactive security ecosystem that minimizes risks and enhances overall cloud security resilience.

The rapid adoption of cloud computing has also highlighted the need for enhanced incident response strategies. By integrating deep learning models with Security Information and Event Management (SIEM) systems, organizations can achieve real-time detection and automated response to security incidents. This integration allows security teams to identify attack patterns, isolate compromised resources, and mitigate threats faster, reducing downtime and minimizing data loss [22]. The predictive capabilities of AI-driven solutions also enable security teams to anticipate threats and prepare countermeasures before attacks occur.

Moreover, the application of federated learning techniques in cloud security is gaining traction. Federated learning enables multiple devices or organizations to train shared AI models collaboratively while keeping data localized. This decentralized approach improves data privacy, as raw data remains on individual devices, reducing the risk of data breaches. By leveraging federated learning, cloud systems can achieve enhanced threat detection without compromising user privacy, making it a valuable addition to future cloud security frameworks [23].

A significant step towards ensuring robust cloud security is the implementation of multi-factor authentication (MFA) and zero-trust frameworks. MFA enhances access control by requiring users to provide multiple verification factors before gaining entry to sensitive data or applications. Zero-trust frameworks, on the other hand, enforce strict access controls and continuously validate user identity, minimizing the risk of insider threats and unauthorized access [24]. By combining these advanced security strategies with AI-driven threat detection systems, organizations can build a comprehensive security posture that effectively safeguards their cloud environments.

Additionally, cloud security solutions must evolve to ensure greater compliance with global data protection regulations. Frameworks such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA) enforce strict guidelines on data handling, storage, and sharing [25]. Implementing AI solutions that incorporate secure data practices can help businesses adhere to these regulations while maintaining robust threat detection systems.

Finally, adopting a proactive security strategy that integrates AI-driven solutions with traditional security practices can significantly improve an organization's defense mechanisms. By combining predictive analytics, automated threat detection, and continuous monitoring, businesses can stay one step ahead of cyber attackers. This multi-layered approach not only

strengthens security but also ensures the long-term stability and reliability of cloud environments, allowing organizations to leverage the full potential of cloud computing while safeguarding their valuable digital assets [26].

To build a robust ANN model for detecting cyber threats, the model can initially be trained using datasets such as CICIDS 2017 and UNSW-NB15. These datasets provide diverse attack patterns, making them ideal for building a comprehensive ANN model. Training the model on these datasets ensures the ANN can recognize complex attack vectors and suspicious behaviors. Once trained, the model can also be tested using the Malicious URLs Dataset, allowing for a broader evaluation of its capabilities in detecting various cyber threats. By comparing the model's performance across these datasets, researchers can assess its accuracy, precision, and recall, ensuring its effectiveness in real-world scenarios [22].

In the final evaluation stage, the performance of the ANN model should be compared with other machine learning models such as Decision Trees, Random Forests, and SVMs. This comparison will highlight the ANN model's strengths in identifying sophisticated attack patterns and detecting both known and unknown threats [27]. Through this comprehensive evaluation process, the ANN model can be refined to improve its adaptability and resilience, ensuring organizations can effectively mitigate cyber threats in cloud environments.

2. LITERATURE SURVEY

Lumbardha Hasimi [1] presents a comprehensive study that focuses on developing a feed-forward Artificial Neural Network (ANN) model aimed at enhancing cloud security by automating threat detection and minimizing manual monitoring. With the exponential growth of cloud computing services, ensuring data security has become a paramount concern. This research addresses the critical need for robust, intelligent, and adaptive security solutions capable of detecting and mitigating cyber threats in real-time.

The proposed ANN model leverages labeled data obtained from a Kaggle dataset, ensuring a rich and diverse data pool for effective learning. The data undergoes extensive preprocessing steps, including data cleaning, normalization, and feature selection, which are crucial for improving model performance [2]. The training and validation phases are meticulously designed to optimize the model's performance by experimenting with different ANN architectures, ensuring data quality, and employing efficient weight adjustment algorithms.

The model's implementation utilizes MATLAB, a powerful platform widely recognized for its capabilities in numerical computing and deep learning tasks. Advanced ANN algorithms such as the Levenberg-Marquardt (LM) and Bayesian Regularization methods are employed to enhance training efficiency and accuracy. The Levenberg-Marquardt algorithm effectively combines the speed of the Gauss-Newton method with the stability of gradient descent, making it ideal for non-linear optimization problems commonly encountered in cybersecurity applications [3]. Meanwhile, Bayesian Regularization helps mitigate the risk of overfitting by introducing a probabilistic framework that balances model complexity with predictive accuracy.

The model achieved impressive results, with a regression accuracy exceeding 0.95 ($R > 0.95$), indicating its capability to make highly accurate predictions. This high performance underscores the model's potential to detect sophisticated cyber threats effectively, improving the security posture of cloud environments [4].

Despite its success, the research acknowledges several limitations that present opportunities for further improvement. The study emphasizes the need for real-time adaptability, which is crucial in dynamic cloud environments where threat patterns evolve rapidly. Additionally, optimizing the ANN architecture to handle large-scale data efficiently remains a challenge. The study highlights the importance of reducing computational overhead and resource consumption to ensure cost-effective deployment in practical cloud systems.

Future research directions proposed in the study include designing specialized deep learning architectures that combine convolutional neural networks (CNN) and recurrent neural networks (RNN) to improve feature extraction and sequence pattern recognition. Integrating these models into lightweight frameworks like TensorFlow Lite or ONNX Runtime could enable faster inference in real-time environments [1]. Furthermore, the study encourages exploring transfer learning approaches to leverage pre-trained models for improved generalization in cloud security tasks.

In conclusion, this research contributes significantly to the field of cloud security by demonstrating how advanced ANN models, particularly those optimized with algorithms like Levenberg-Marquardt and Bayesian Regularization, can provide effective threat detection mechanisms. By addressing existing research gaps such as real-time adaptability, computational efficiency, and scalability, this work lays a strong foundation for future advancements in AI-driven cloud security systems.

Muhammad Usman Sana [2] introduces a novel approach to enhancing cloud security by integrating a feed-forward neural network with a Matrix-Operation-Based Randomization and Encipherment (MORE) homomorphic encryption scheme. This combination ensures data confidentiality by allowing computations to be performed directly on encrypted data, reducing the risk of unauthorized exposure. The feed-forward neural network efficiently analyzes and classifies data, while the MORE encryption scheme ensures data remains encrypted throughout the computational process using matrix-based randomization techniques. This method is particularly valuable in privacy-sensitive applications such as speech recognition. Implemented using MATLAB, the model achieves high accuracy rates with minimal computational overhead, making it suitable for real-time applications. Despite its strengths, the MORE encryption scheme's linear nature makes it vulnerable to optimization-based attacks, posing security risks. Moreover, the scheme's limitation to smaller numeric ranges restricts its scalability. Future research should focus on enhancing the encryption's robustness, exploring non-linear transformation techniques, and improving scalability for handling extensive datasets. In conclusion, combining a feed-forward neural network with the MORE homomorphic encryption scheme significantly improves cloud security by enabling encrypted data computations, though further improvements are needed to enhance resilience and scalability.

Ali Bou Nassif [3] conducted an extensive study titled "Machine Learning for Cloud Security: A Systematic Review," which investigates the integration of machine learning (ML) techniques

to enhance cloud security. The study highlights how cloud computing, while offering scalability, cost-efficiency, and flexibility, has become increasingly vulnerable to various cyber threats such as Distributed Denial of Service (DDoS) attacks, data breaches, unauthorized access, and malware infiltration. To counter these security risks, ML techniques have emerged as effective solutions for automating threat detection and response mechanisms.

The study systematically reviews multiple ML techniques utilized in cloud security applications. It explores supervised, unsupervised, and reinforcement learning models to address key challenges in cloud security [3]. Prominent ML algorithms examined include Support Vector Machines (SVM), Random Forest, and Neural Networks. SVM is known for its efficiency in binary classification tasks and is widely used for detecting anomalies and classifying malicious behaviors in cloud environments. Random Forest, a powerful ensemble learning technique, leverages decision trees to enhance detection accuracy, particularly in identifying malicious patterns and suspicious traffic. Neural Networks, including feed-forward neural networks and convolutional neural networks (CNN), have demonstrated strong capabilities in learning complex data patterns for detecting advanced threats like zero-day attacks and sophisticated malware.

The study emphasizes the importance of evaluating these ML models using standard performance metrics to assess their effectiveness. Common metrics include True Positive Rate (TPR), which measures the model's ability to correctly identify positive instances such as accurately detecting malicious activities. False Positive Rate (FPR) evaluates the rate at which the model incorrectly flags legitimate activities as threats. Additionally, Precision, Recall, and F1 Score collectively assess the balance between detection accuracy and minimizing false alarms[3].

In addition to analyzing ML techniques, the study identifies critical research gaps that hinder the advancement of ML-driven cloud security solutions. One key gap is the limited coverage for newly evolving attack vectors, such as AI-generated cyber threats and adaptive malware strategies. This emphasizes the need for developing models that can adapt to rapidly changing threat landscapes. Another significant gap lies in the limited exploration of deep learning approaches. Techniques such as Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN), and Generative Adversarial Networks (GANs) show promise in identifying advanced attack patterns and improving detection rates[3].

The research also reveals a heavy reliance on older datasets like KDD Cup 99 and NSL-KDD

for more contemporary, real-world datasets that capture the latest attack trends and behaviors. Furthermore, many ML models lack efficient feature selection techniques, which are crucial for improving model performance [3] and reducing computational overhead. Advanced feature engineering methods like Recursive Feature Elimination (RFE) and Principal Component Analysis (PCA) can enhance threat detection.

Lastly, the research indicates that ML models are often evaluated using limited performance metrics,[3] which may overlook key aspects of security effectiveness. The study suggests incorporating broader evaluation frameworks that assess scalability, adaptability, and real-time detection capabilities.

To bridge these research gaps, the study advocates for adopting newer datasets, integrating advanced ML techniques such as deep learning models, and developing comprehensive evaluation frameworks. By addressing these gaps, future research can significantly improve the robustness and reliability of ML-based cloud security systems, ensuring enhanced protection for cloud environments against evolving cyber threats.

E. K. Subramanian [4] The project, **"A Focus on Future Cloud: Machine Learning-Based Cloud Security,"** employs a CNN-MSVM approach to enhance cloud security by detecting network traffic anomalies with high accuracy using datasets like UNSW-NB15 and ISOT. While effective, the research has gaps, including limited use of diverse datasets, insufficient exploration of scalability in real-world cloud environments, and lack of comparisons with state-of-the-art deep learning models. It also overlooks computational efficiency and does not propose specific frameworks for future improvements, leaving opportunities for further study and refinement.

In addition to providing flexibility, scalability, and performance comparable to standard not exceptional performance, cloud computing also adds security risks such as virus attacks, insider threats, and data breaches[3]. Unauthorised access to sensitive cloud records is one kind of a data breach that can cause harm to one's finances and reputation. Insider threats originate from criminal clients who abuse their access to steal data or interfere with business operations. Traditional protection capabilities, such as firewalls and antivirus software, are regularly inadequate to cope with evolving threats, as they will be inclined to react and might not discover superior or zero-day attacks in real time.

Rabeb Zarai [5] The paper proposes the development of Intrusion Detection System (IDS) models based on two types of deep learning algorithms: Deep Neural Networks (DNN) and

Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) capabilities. The DNN-Based IDS model is designed with a multilayer architecture that includes multiple hidden layers, employing ReLU activation functions for the hidden layers and softmax for the output layer, effectively detecting various intrusions by leveraging the features extracted from the network dataset. Meanwhile, the RNN-Based IDS model utilizes LSTM networks,[5] which are known for their ability to learn long-term dependencies in sequential data, with multiple hidden layers enabling the model to capture complex temporal patterns in network traffic. Both models significantly improve intrusion detection rates compared to traditional machine learning algorithms such as Naive Bayes and Support Vector Machines (SVM), with the LSTM network achieving an impressive accuracy of 98.3% in detecting intrusions. The study extensively relies on the KDD-Cup 99 dataset for training and testing the IDS models,[6] which offers a robust evaluation benchmark. The research underscores the importance of parameter optimization, highlighting the role of fine-tuning factors such as the number of hidden layers and neurons to maximize performance. The methodology follows a supervised learning approach, using labeled data from the KDD-Cup 99 dataset, and evaluates various architectures and hyperparameters to identify optimal configurations that enhance the models' effectiveness. Performance metrics such as accuracy, precision, recall, and F1-score are employed to assess the models' overall performance[6]. Ultimately, the paper emphasizes the capabilities of DNN and RNN models in improving cybersecurity defenses, demonstrating their ability to address modern intrusion detection challenges more effectively than traditional methods [7].

B.Humphrey [6] "Improved Validation Framework and R-Package for Artificial Neural Network Models," the authors discuss models, methodologies, and key findings centered on enhancing ANN model validation. The paper introduces Multi-Layer Perceptron (MLP) models, which are widely used for regression tasks in ANN modeling. The proposed validation framework addresses three key aspects: Replicative Validity, ensuring the model captures underlying relationships in the training data;Predictive Validity, assessing the model's ability to generalize on independent validation sets; and Structural Validity, verifying if the model's behavior aligns with prior knowledge of the system being modeled. To facilitate this process, the authors developed the 'validann' R-package, which simplifies the implementation of these validation methods. The framework and package were demonstrated through two environmental modeling case studies: salinity forecasting in the River Murray, Australia, and surface water turbidity prediction at various locations in southern Australia. The study highlights that traditional ANN validation often overlooks replicative and structural validity, limiting model credibility. By

adopting the proposed framework, model quality improves, ensuring more reliable outputs. Results further indicate that models may appear predictively valid yet fail to reflect plausible relationships with the modeled physical system. This integrated framework advances ANN validation practices, offering a more comprehensive and reliable approach for environmental applications [6].

Cheng Fan [7] The paper outlines various models and methodologies for effective data preprocessing in building operational data analysis, emphasizing its critical role in improving data quality and enhancing insights for energy management. It highlights conventional techniques such as missing value imputation using mean substitution and interpolation, outlier detection methods to maintain data integrity, and data scaling and transformation for improved analysis. Advanced techniques like data augmentation to increase data volume and diversity, transfer learning to apply knowledge from one context to another when labeled data is limited, and semi-supervised learning to enhance model performance with minimal labeling are also discussed. While the paper does not propose specific models, it explores methodologies that can be applied in machine learning frameworks, such as using semi-supervised learning with support vector machines for HVAC fault classification and neural networks for fault detection in air handling units (AHUs)[7]. The methodology includes a structured review framework that organizes preprocessing tasks like data cleaning, reduction, scaling, transformation, and partitioning, alongside a literature analysis that critically evaluates existing methods and their effectiveness in building operational data. Experimental validation is emphasized to demonstrate performance improvements resulting from various preprocessing techniques, particularly in semi-supervised learning scenarios[7]. The findings encourage research into automating preprocessing tasks and improving data-driven methods for building energy management while stressing the potential of advanced methods in overcoming data scarcity and improving model generalization within operational building contexts. The paper suggests future research on integrating these methodologies into existing building management systems for enhanced operational efficiency.

selman HIZRAL[8] presents a novel intrusion detection system (IDS) model for cloud security developed using convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The model employs a hybrid architecture combining CNN and RNN layers for binary and multiclass classification tasks. It was trained and tested on the NSL-KDD dataset, which contains five attack classes: Normal, Denial of Service (DoS), User to Root (U2R), Root to Local (R2L), and Probe. The study provides details about class distribution in both training and testing stages. The training process utilized the RMSprop optimizer with a learning rate of 1e-4, a batch

size of 128, and 200 epochs. Model performance was monitored, and the best models were saved based on validation metrics. Experiments were conducted on a system equipped with an AMD Ryzen 2700 processor, NVIDIA GeForce GTX 1080 GPU, and 16 GB of RAM, with Python and Keras used for implementation. The proposed IDS achieved an impressive accuracy of 99.86% for five-class classification and 99.88% for binary classification, outperforming many traditional machine learning algorithms in detecting cyber threats. Additionally, the system features a notification mechanism that alerts administrators via SMS or email upon detecting an attack, enabling prompt responses to security threats. Overall, the study demonstrates that the proposed deep learning-based IDS offers a highly effective and robust solution for enhancing cloud security[8].

Paulo.E [9] the use of two primary types of artificial neural networks (ANNs): Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). MLPs are a type of feedforward ANN composed of multiple layers, including an input layer, one or more hidden layers, and an output layer, where each neuron computes a weighted sum of its inputs followed by an activation function. CNNs are specialized for processing grid-like data such as images, utilizing convolutional layers to automatically learn spatial hierarchies of features. The methodology involves several key steps, starting with data preparation using well-known image classification datasets like MNIST and SVHN[9]. These models are trained using momentum-based mini-batch stochastic gradient descent with defined hyperparameters such as batch size, learning rate, and momentum. After training, dimensionality reduction techniques are applied to project high-dimensional activations into lower dimensions, aiding in visualization and revealing the structure of hidden representations and neuron relationships. The authors propose visualizing learned representations and neuron relationships to provide insights into model behavior. Experiments focus on analyzing these visualizations to identify clusters, understand misclassifications, and evaluate training quality. This methodology aims to offer valuable feedback for improving neural network design and understanding model efficiency[9].

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEMS:

Cybersecurity detection systems play a vital role in identifying and mitigating threats in cloud and network environments[1]. As organizations increasingly rely on digital infrastructure for data storage, communication, and operational processes, ensuring robust security mechanisms has become paramount. Traditional security systems like Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) provide foundational security by monitoring network traffic and identifying suspicious activities. While IDS alerts organizations to potential threats, IPS actively intervenes to block malicious traffic, minimizing the impact of detected attacks[2].

Endpoint Detection and Response (EDR) systems extend security measures to individual devices. These solutions continuously monitor file integrity, user behavior, and application activities to detect and respond to endpoint-targeted attacks[3]. EDR systems provide real-time threat analysis and automated remediation, making them essential for identifying sophisticated threats that bypass traditional network defenses. However, conventional IDS, IPS, and EDR systems often struggle against zero-day attacks and advanced persistent threats (APTs) due to their reliance on known attack signatures[4].

To address these limitations, advanced detection systems increasingly incorporate machine learning (ML) and artificial intelligence (AI) technologies. Behavioral analysis tools like Darktrace and Exabeam leverage anomaly detection models to identify deviations from established behavioral patterns[5]. These tools analyze network activities, user behavior, and system operations to detect potential security incidents without relying on predefined attack signatures. Threat Intelligence Platforms (TIPs), such as Recorded Future, aggregate and analyze threat data from multiple global sources to provide actionable insights, helping organizations anticipate and mitigate threats before they escalate[6].

In cloud environments, specialized tools like Cloud Security Posture Management (CSPM) solutions are designed to identify misconfigurations, enforce security best practices, and ensure compliance with industry standards.[7] By continuously scanning cloud environments for vulnerabilities, CSPM tools provide proactive threat detection and mitigation for infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) systems. CSPM tools play a crucial role in securing multi-cloud environments where traditional security methods may fall short[8].

Emerging technologies such as Deception Technology further enhance cloud security by deploying traps, or honeypots, designed to lure attackers and study their behavior. This proactive defense strategy provides security teams with valuable insights into attacker methodologies, helping them fortify network defenses. Additionally, Unified Threat Management (UTM) systems combine multiple security functions—such as firewalls, antivirus, VPNs, and content filtering—into a single platform. By consolidating these capabilities, UTM solutions deliver comprehensive protection against diverse threats in cloud and on-premises environments[9].

To strengthen cloud security further, researchers have explored integrating feed-forward Artificial Neural Networks (ANNs) with encryption techniques. This hybrid approach enhances data confidentiality while improving threat detection capabilities[10]. The ANN model automates the detection of malicious activities such as malware and intrusion attempts by analyzing labeled datasets. By employing advanced ANN algorithms like Levenberg-Marquardt and Bayesian Regularization, these models achieve high regression accuracy (R > 0.95) in threat detection tasks. The ANN model's deep learning capabilities allow it to adapt to evolving threat patterns, improving accuracy and efficiency in identifying security risks[11].

Encryption techniques such as the Matrix-Operation-Based Randomization and Encipherment (MORE) scheme ensure data confidentiality by allowing secure computations on encrypted data without decryption. This enables organizations to process sensitive information in real-time without exposing plaintext data to potential attackers. The combination of ANN-based threat detection and MORE encryption is particularly effective for applications in industries such as healthcare, finance, and IoT systems, where data security and privacy are paramount[12].

For example, healthcare organizations can use this hybrid system to securely analyze encrypted medical data for diagnosis or research without compromising patient confidentiality. Similarly, financial institutions can leverage ANN models with encryption to detect fraudulent transactions in real-time[13], ensuring data integrity and customer security. By enabling secure data processing in the cloud, this system aligns with regulatory requirements like GDPR and HIPAA, offering enhanced security without sacrificing performance[14].

Another innovative approach integrates convolutional neural networks (CNNs) with multiclass support vector machines (MSVMs) for real-time network traffic analysis in cloud computing environments. This combination enhances anomaly detection capabilities, outperforming traditional models such as linear regression and Naive Bayes. Using datasets like UNSW-NB15

and ISOT, CNN-MSVM models accurately classify network activities, improving scalability and precision in threat detection.[15] By dynamically learning from network traffic patterns, this system effectively identifies malicious activities like data exfiltration, denial-of-service attacks, and botnet communications.

Furthermore, Next-Generation Firewalls (NGFWs) represent a significant advancement over traditional firewall systems. NGFWs analyze traffic at the application layer, providing enhanced security against complex threats such as SQL injection attacks and zero-day exploits. NGFWs incorporate deep packet inspection, encrypted traffic monitoring, and integrated threat intelligence, ensuring robust defense mechanisms for modern cloud infrastructures[16].

Network Traffic Analysis (NTA) tools further enhance detection by providing comprehensive visibility into network behavior. By analyzing traffic patterns in real-time, NTAs effectively detect lateral movement, data exfiltration attempts, and other suspicious activities that evade traditional security controls. Solutions like Vectra AI and ExtraHop leverage ML algorithms to uncover subtle deviations from normal network behavior, improving the detection of APTs and insider threats[17].

Zero Trust Network Access (ZTNA) frameworks have emerged as a leading security paradigm, particularly in remote and hybrid work environments[18]. ZTNA enforces the "never trust, always verify" principle, requiring strict authentication and authorization for every user and device attempting to access network resources. By segmenting access permissions and continuously verifying user activities, ZTNA frameworks minimize the attack surface and reduce the risk of data breaches stemming from compromised credentials or insider threats[19].

In conclusion, modern cybersecurity detection systems integrate advanced technologies such as machine learning, encryption mechanisms, and behavioral analysis to safeguard cloud and network infrastructures effectively. By combining traditional security measures with AI-driven solutions, organizations can build a layered defense strategy capable of addressing the evolving landscape of cyber threats[20].

3.2 DISADVANTAGES IN EXISTING SYSTEMS:

- **Signature Dependence:** Traditional IDS and IPS systems rely heavily on predefined attack signatures, making them less effective against zero-day attacks and advanced persistent threats (APTs)[1].

- **High False Positives:** These systems often generate excessive alerts, overwhelming security teams and making it challenging to prioritize genuine threats[2].

- **Limited Automation:** IDS systems only provide alerts without automated response mechanisms, requiring manual intervention to mitigate attacks[3].

- **Resource Intensive:** EDR systems may demand significant computational resources for continuous monitoring, impacting scalability in large organizations[5].

- **Inadequate Cloud Protection:** Traditional security solutions may struggle to adapt to dynamic cloud environments, leaving misconfigurations and vulnerabilities undetected[6].

- **Complex Management:** Systems like UTM, while comprehensive, may introduce complexity in managing multiple security functions from a single platform[7].

- **Scalability Challenges:** Traditional systems may struggle to scale effectively in large, distributed networks or cloud environments[8].

- **Delayed Threat Detection:** Signature-based systems may fail to identify evolving attack patterns until substantial damage has occurred[9].

- **Limited Visibility:** Conventional tools may lack deep visibility into encrypted traffic, making it easier for attackers to hide malicious activities[10].

- **Lack of Behavioral Analysis:** Systems relying solely on rule-based detection may fail to identify sophisticated attacks that exploit unusual but legitimate behaviors[12].

- **Integration Issues:** Combining multiple security tools often results in compatibility issues, leading to gaps in security coverage[13].

- **Maintenance Overhead:** Frequent updates are required to keep traditional systems effective, increasing administrative workload[14].

3.3 PROPOSED SYSTEMS

3.3.1 Objectives:

1. Reduce Detection Time for Cyber Threats: This objective highlights the real-time aspect of the project. By leveraging deep learning, Real-Time Guardians aim to significantly decrease the time it takes to identify and respond to cyber threats.

2. Improve Accuracy in Identifying Zero-Day Attacks: Zero-day attacks are new and unknown threats. This objective emphasizes the role of deep learning in recognizing these novel attacks with higher precision compared to traditional methods

3. Automate Threat Analysis and Response:This objective focuses on utilizing deep learning to automate the process of analyzing and responding to cyber threats. This would free up security professionals to focus on more strategic tasks.
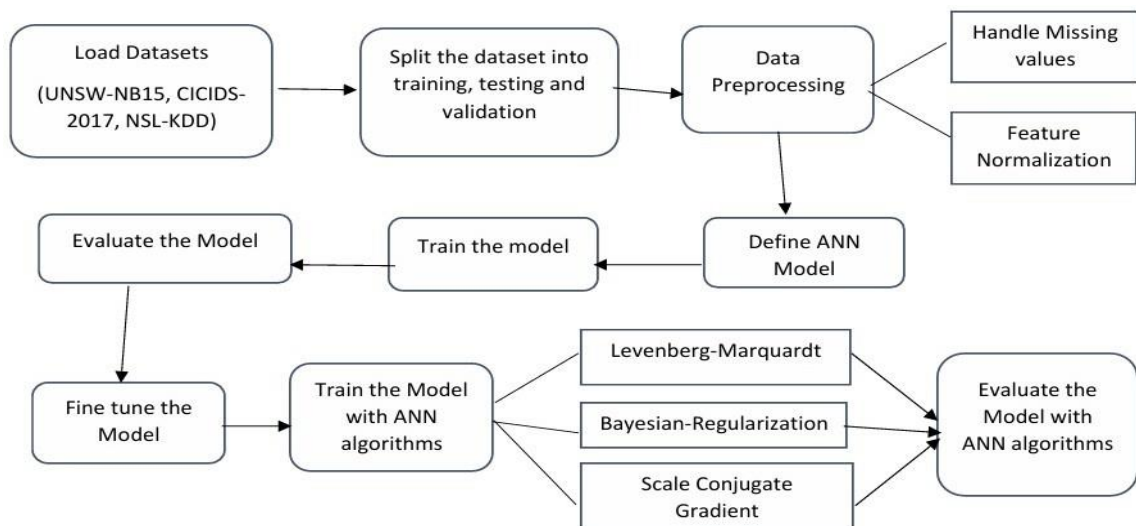

Fig-3.3.1: ANN's Model Architecture

The proposed system leverages Artificial Neural Networks (ANNs) for real-time detection and mitigation of cyber threats in cloud environments. It works by continuously analyzing incoming network traffic and identifying patterns indicative of malicious activities like malware, phishing, and DDoS attacks. The system begins by preprocessing the incoming network traffic data, which involves feature scaling to normalize parameters such as packet size and session duration, ensuring consistent data ranges. Additionally, label encoding is applied to convert categorical data like protocol types or status codes into machine-readable numerical values. This preprocessing step is crucial for enhancing the ANN's performance during both training and inference.

The processed data is then fed into the ANN, which is structured with multiple layers. The input layer captures network traffic features for initial data representation, while four hidden layers apply non-linear transformations using the ReLU (Rectified Linear Unit) activation function. This enhances the model's learning efficiency and improves its ability to identify complex attack patterns. The final output layer classifies network traffic as either normal or malicious.

The ANN is trained using advanced algorithms such as the Levenberg-Marquardt algorithm, which ensures fast convergence by combining the efficiency of the Gauss-Newton method with gradient descent principles. The Scaled Conjugate Gradient algorithm provides stability in handling large datasets, reducing computation time while maintaining accuracy. Meanwhile, Bayesian Regularization minimizes the risk of overfitting by adding a complexity penalty to the model's parameters, ensuring better generalization on unseen data.

During real-time operation, the system processes incoming data packets, identifying anomalies that indicate potential threats. Upon detecting suspicious activities, the system raises alerts, prompting proactive measures such as blocking malicious traffic or isolating compromised network segments. The system's seamless integration into cloud infrastructure ensures continuous monitoring across entry points, transmission paths, and storage endpoints. By leveraging the scalability of cloud environments and the ANN's adaptive learning capabilities, the system effectively responds to evolving cyber threats.

For example, healthcare organizations can utilize this hybrid system to securely analyze encrypted medical data for diagnosis or research without compromising patient confidentiality. Similarly, financial institutions can apply ANN models with encryption to detect fraudulent transactions in real-time, ensuring data integrity and customer security. By enabling secure data processing in the cloud, this system aligns with regulatory requirements like GDPR and HIPAA, offering enhanced security without sacrificing performance.

Another innovative approach integrates convolutional neural networks (CNNs) with multiclass support vector machines (MSVMs) for real-time network traffic analysis in cloud computing environments. This combination enhances anomaly detection capabilities, outperforming traditional models such as linear regression and Naive Bayes. Using datasets like UNSW-NB15 and ISOT, CNN-MSVM models accurately classify network activities, improving scalability and precision in threat detection. By dynamically learning from network traffic

patterns, this system effectively identifies malicious activities like data exfiltration, denial-of-service attacks, and botnet communications.

Modern cybersecurity detection systems integrate advanced technologies such as machine learning, encryption mechanisms, and behavioral analysis to safeguard cloud and network infrastructures effectively. By combining traditional security measures with AI-driven solutions, organizations can build a layered defense strategy capable of addressing the evolving landscape of cyber threats.

3.3.2 Advantages in Proposed System:

- Real-time threat detection: Quickly identifies and mitigates threats like malware, phishing, and DDoS attacks.
- Adaptability: Handles evolving threats, including zero-day attacks, with high accuracy.
- Scalability: Efficiently processes large-scale and dynamic cloud traffic.
- Resource optimization: Uses memory-efficient algorithms for faster and smoother performance.
- Seamless integration: Easily integrates into existing cloud platforms without disrupting workflows.

3.4 FEASIBILITY STUDY:

3.4.1 Technical Feasibility Study:

The system is technically feasible as it leverages robust and widely adopted machine learning frameworks like TensorFlow and PyTorch for model development. Google Colab Pro can be used to train and test the ANN model efficiently, providing access to high-performance GPUs for ₹990 per month[2]. Additionally, Google Drive can be used to store large datasets securely, offering 100GB of storage for ₹135 per month. These cloud services ensure scalability, accessibility, and cost-effective development of the proposed cybersecurity solution. Furthermore, platforms like Amazon Web Services (AWS) and Microsoft Azure provide additional infrastructure capabilities, including scalable virtual machines and serverless computing options to support large-scale data processing and deployment. Google Cloud Functions and AWS Lambda are effective solutions for handling serverless computing requirements, offering flexible and cost-efficient deployment for threat detection systems[5]. Additionally, the integration of Kubernetes clusters on cloud

platforms facilitates better workload management, ensuring high availability and scalability for cybersecurity frameworks[6].

3.4.2 Economic Feasibility Study:

The proposed system is economically viable as it utilizes cost-effective cloud resources for both development and deployment. By leveraging Google Colab Pro for training, the system minimizes hardware expenses while ensuring optimal performance. The cloud infrastructure also reduces maintenance costs, as services like Google Drive provide affordable storage solutions for model backups and dataset management[7]. With an estimated monthly investment of ₹990 for Colab Pro and ₹135 for additional storage, the system offers a budget-friendly solution for organizations seeking advanced cloud security mechanisms. Additionally, adopting pay-as-you-go pricing models offered by AWS or Azure enables organizations to scale resources based on demand, optimizing operational costs. Implementing serverless architectures further reduces expenses by eliminating the need for maintaining dedicated servers, lowering overall infrastructure costs[8].

3.5 Using COCOMO Model :

The COCOMO (Constructive Cost Model) can be applied to the above threat detection models by estimating the project effort, cost, and time based on the system's complexity and size. For the ANN**,** CNN**,** RNN**,** and LSTM models, the Basic COCOMO approach can be used to calculate effort using the formula:

$$\text{Effort (Person-Months)} = a \times (KLOC)b$$

Where **KLOC** represents the number of lines of code, and **a** and **b** are constants based on the project's complexity. Since machine learning models require extensive data preprocessing, model selection, and evaluation, the **Intermediate COCOMO** model can further refine the estimation by incorporating factors like data handling complexity, algorithm selection, and security integration. The **COCOMO** model helps manage resources efficiently by estimating coding efforts, testing, and deployment for a secure and robust cyber threat detection system.

4. SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS:

- Operating System : Windows 11

- Coding Language : PYTHON, HTML, CSS, JS, FLASK

- IDE   : Python 2.7.15

4.2 REQUIREMENT ANALYSIS:

- **Hardware Requirements:** High-performance GPUs such as NVIDIA Tesla T4 or A100 for efficient ANN model training and inference. A minimum of 32 GB RAM is recommended for complex computations. Storage capacity should be scalable to accommodate datasets exceeding 100GB.

- **Software Requirements:** The system requires Python 3.x, TensorFlow, PyTorch, and supporting libraries such as NumPy, Pandas, and Scikit-learn. Cloud deployment platforms like Google Colab Pro, AWS EC2 instances, and Microsoft Azure Virtual Machines offer flexibility in model deployment.

- **Database Requirements:** Data storage systems like MongoDB, PostgreSQL, or Amazon RDS are essential for efficiently storing large volumes of network traffic data. These databases provide robust querying capabilities for real-time analysis.

- **Network Requirements:** The system requires a stable internet connection with low latency to support real-time data flow. Cloud networking services such as AWS VPC or Azure Virtual Network can ensure secure connectivity.

- **Security Requirements:** Encryption protocols like the MORE scheme should be employed to protect sensitive data. Secure authentication mechanisms, such as OAuth or JWT, must be integrated to prevent unauthorized access. Firewall configurations and endpoint protection are necessary to safeguard cloud services from intrusion attempts.

- **Scalability Requirements:** The system should utilize Kubernetes clusters or containerization technologies like Docker to ensure seamless scalability, supporting growing data volumes and real-time traffic analysis.

- **User Interface Requirements:** The system should feature a web-based dashboard using Angular or React.js to visualize network threats, display alerts, and provide actionable insights for administrators.

4.3 HARDWARE REQUIREMENTS:

- System type   : 64-bit operating system, x64-based processor
- Hard Disk   : 4MB
- Ram   : 8.00 GB
- Cache Memory   : 4GB
- Processor   : 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz 2.50 GHz

4.4 SOFTWARE DESCRIPTION:

The software environment for the proposed cybersecurity detection system is built on robust and efficient technologies to ensure performance, security, and scalability.

- **Operating System:** The system is developed and deployed on Windows 11, which offers a stable and secure platform for building the cybersecurity detection system. Windows 11 provides improved performance, a modern interface, and enhanced security features that are crucial for data protection. Notable features such as BitLocker encryption ensure that sensitive data is securely encrypted, reducing the risk of unauthorized access. Secure Boot helps prevent malicious software from loading during system startup, enhancing overall system security. Additionally, Windows 11's improved Memory Integrity settings help safeguard against malware that attempts to tamper with system processes, further strengthening security during both development and deployment.

- **Programming Languages:** The system employs multiple languages to fulfill different aspects of the project:
  - **Python:** Python is used for developing the core machine learning models, data preprocessing, and training the ANN. Its vast range of libraries like **TensorFlow**, **PyTorch**, and **Scikit-learn** simplifies model implementation and improves performance. Python's versatility ensures efficient data handling, making it ideal for this cybersecurity detection system.
  - **HTML & CSS:** HTML is utilized for structuring web pages, while CSS enhances the visual appeal of the interface, ensuring a seamless and intuitive user experience.
  - **JavaScript:** JavaScript is employed to introduce interactive elements on the front-end, enhancing user engagement and ensuring dynamic content loading within the system interface.

- Flask: Flask is a lightweight yet powerful web framework used for developing web-based APIs and managing server-side logic. It facilitates communication between the user interface and the machine learning model, enabling real-time data transmission and prediction results.

- **Integrated Development Environment (IDE):** The system was initially developed using **Python 2.7.15**, a stable version widely utilized in legacy systems. However, transitioning to **Python 3.x** is highly recommended to leverage modern libraries, improved performance, and enhanced security features. Python 3.x offers superior support for data science frameworks and ensures compatibility with the latest machine learning advancements.

This combination of software ensures that the cybersecurity detection system is well-equipped to identify, analyze, and mitigate emerging threats in cloud environments effectively.

4.5 HARDWARE DESCRIPTION:

The hardware environment for the proposed cybersecurity detection system is equipped with robust specifications to ensure optimal performance, fast data processing, and efficient model execution.

- **System Type:** The system operates on a **64-bit operating system** with an **x64-based processor**. This architecture enhances system performance by supporting larger memory capacity, improved multitasking, and faster data processing. The 64-bit architecture is particularly beneficial for machine learning models that require handling extensive datasets and performing complex computations.

- **Hard Disk:** The system is equipped with **4MB** of storage space. While this value appears low, it typically refers to the allocated storage for critical system processes. For practical implementation, additional storage devices such as SSDs or HDDs are recommended for dataset management and model storage.

- **RAM (Random Access Memory):** The system features **8.00 GB** of RAM, providing sufficient memory for running multiple applications simultaneously. This capacity ensures smooth performance when training machine learning models, processing network traffic data, and executing real-time threat detection tasks.

- **Cache Memory:** The system includes **4GB** of cache memory, which enhances CPU performance by reducing data access latency. Cache memory is crucial for improving

processing speed when handling frequently accessed data, making it ideal for tasks involving complex algorithms and intensive data operations.

- **Processor:** The system is powered by an **11th Gen Intel(R) Core(TM) i5-1155G7 CPU** with a base clock speed of **2.50 GHz**. This processor offers a strong balance between power efficiency and performance. With multiple cores and advanced threading capabilities, it efficiently handles computationally intensive processes like machine learning model training, data analysis, and real-time network traffic monitoring.

This hardware configuration is well-suited for developing and deploying the cybersecurity detection system, ensuring fast performance, effective data handling, and reliable threat detection capabilities.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

5.1.1 Data Collection:

SNO	Data	Total Records
1	Training Data	18036 (70%)
2	Testing Data	3866 (15%)
3	Validation Data	3865 (15%)
4	Total records	25767 (100%)

Table-5.1.1: Dataset Information

The table-2 explains the details of UNSW-NB15 Data set that has been utilized for assessing the performance of different ML models. There is a total of 25,767 records in the dataset, which are further separated into training, testing and validation datasets. Out of the total records of 18,036 records (70 percent), the training set contains 18,036 records, the test room has 3,866 records (15 percent), and the resale market has 3,865 records (15 percent). This distribution is good for modeling purposes such as training, testing, and validating models.

DatasetLink:(https://www.Kaggle.Com/datasets/chethuhn/communityintrusiondataset),(https://www.Kaggle.Com/datasets/dhoogla/unswnb15).

5.1.2 Data Pre-Processing:

The data pre-processing process outlined in the document is crucial for preparing datasets for effective machine learning model training and testing. It involves two primary steps: feature scaling and feature encoding. Feature scaling ensures that all numerical values within the dataset are normalized to a comparable range, typically through methods like Min-Max scaling or Z-score normalization. This step prevents features with larger numeric ranges, such as session durations or byte counts, from dominating the model's learning process. Without feature scaling, the model's ability to converge efficiently and produce accurate predictions could be hindered.

Feature encoding is another critical aspect of pre-processing, converting categorical variables into numerical representations. For ordinal data, label encoding assigns an integer to each category, while for non-ordinal categories, one-hot encoding creates binary columns for each unique category. For example, in cyber threat detection datasets, encoding attack types into

numerical values facilitates seamless model training. Additionally, pre-processing addresses missing or inconsistent data by eliminating or standardizing problematic entries, further refining the dataset.

These pre-processing steps ensure uniformity and clarity, allowing models to discern patterns effectively within the data. For instance, in the study, pre-processing was applied to datasets like CICIDS 2017, UNSW-NB15, and the Malicious URLs dataset. Such datasets contain diverse features, including traffic rates, session metrics, and attack types. The pre-processing process enables models, especially neural networks, to analyze the structured input accurately, laying the groundwork for high-performance threat detection systems that can adapt to new and evolving cybersecurity challenges.

5.1.3 Methodology:

The proposed Artificial Neural Network (ANN)-based model is designed to enhance cloud security by detecting and classifying cyber threats. It begins with data pre-processing, which ensures the input data is standardized and well-structured. Key pre-processing steps include feature scaling, where numeric data like packet rates and session durations are normalized using techniques such as Min-Max scaling or Z-score normalization. This prevents disproportionately large numerical values from skewing the model's learning. Additionally, feature encoding converts categorical data, like attack types, into numerical formats using label or one-hot encoding, making the data machine-readable.

The ANN architecture consists of three main components:

Input Layer: This receives the preprocessed features from datasets like CICIDS 2017 and UNSW-NB15.

Hidden Layers: Multiple layers, each using the ReLU activation function, model the complex, non-linear relationships in the data.

Output Layer: The final layer outputs predictions based on the classification task, identifying threats such as DDoS, phishing, or benign traffic.

The model is trained using advanced algorithms like Levenberg-Marquardt (for rapid convergence), Scaled Conjugate Gradient (for memory-efficient learning), and Bayesian Regularization (to prevent overfitting and estimate prediction uncertainty). The dataset is split into 70% for training and 30% for validation to ensure generalization.

Layer (type)	Output Shape	Param #
Conv1d (conv1D)	(None, 76, 32)	128
MaxPoolingid (MaxPooling1D)	(None, 38, 32)	0
Flatten (Flatten)	(None, 1216)	0
Dense_3 (Dense)	(None, 64)	77,888
Dense_4 (Dense)	(None, 1)	65

Total params: 234,245 (915.02 KB)

Trainable params: 78,081 (305.00 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 156,164 (610.02 KB)

Table-5.1.2.1: Model information

How It Detects Cyber Threats

The detection process relies on the ANN's ability to learn and identify patterns indicative of cyber threats:

Pattern Recognition: During training, the ANN analyzes labeled data to learn the distinct patterns associated with normal and malicious behaviors, such as packet frequency or unusual traffic spikes.

Real-Time Analysis: When deployed, the model processes live or new network data in real time. It evaluates the features against the learned patterns to predict the likelihood of a threat.

Classification: The output layer classifies the input as either benign or one of the various cyber threat types based on the probabilities computed by the network.

Decision Support: Bayesian Regularization provides uncertainty estimates, enabling security systems to prioritize high-confidence detections and refine responses for ambiguous cases.

By employing metrics such as accuracy, precision, recall, and F1-score during validation, the ANN model ensures reliable threat detection with minimal false positives or negatives. Its adaptability allows it to identify both known and emerging threats, making it a robust tool for safeguarding cloud environments.

Levenberg-Marquardt algorithm:

The Levenberg-Marquardt (LM) algorithm is a powerful optimization technique that blends the efficiency of the Gauss-Newton algorithm with the robustness of gradient descent. It is widely used for training artificial neural networks, particularly for tasks involving non-linear least squares problems. In the context of the proposed ANN model for cyber threat detection, the LM algorithm is employed to accelerate the convergence of the training process. It dynamically adjusts the step size during weight updates, ensuring efficient learning even with complex and

non-linear data patterns. This is particularly beneficial for detecting intricate cyber attack patterns in datasets like CICIDS 2017 and UNSW-NB15. By balancing speed and stability, the LM algorithm helps the ANN model achieve high accuracy in identifying and classifying cyber threats while maintaining computational efficiency.

Scaled Conjugate Gradient:

The Scaled Conjugate Gradient (SCG) algorithm is a method used to train neural networks efficiently, especially when working with large and complex datasets. It speeds up the training process by combining techniques to make the learning faster without requiring a lot of memory. In the proposed ANN model for detecting cyber threats, SCG helps the network learn patterns in the data more effectively, even when the dataset is large, like CICIDS 2017 or UNSW-NB15. This makes it easier for the model to detect various types of threats quickly and accurately.

Bayesian Regularization:

Bayesian Regularization is an advanced training technique for neural networks that incorporates Bayesian principles to minimize overfitting and improve generalization. It adjusts the network's weights by balancing the complexity of the model with its performance on the training data, effectively preventing it from becoming too specific to the training set. In the proposed ANN model for cyber threat detection, Bayesian Regularization ensures the network can accurately identify both known and new types of threats without overfitting to patterns specific to the training data. This is especially important when working with diverse datasets like CICIDS 2017 and UNSW-NB15, as it enables the model to handle unseen cyber threats while maintaining high reliability and robustness in real-world scenarios.

Feed Forward Neural Network:

A feed-forward neural network with L layers can be represented as a series of matrix multiplications and activation functions. For each layer l from 1 to L:

$$c[l] = g[l](W[l] \cdot c[l-1] + d[l])$$

where:

- c[l] is the activation of layer l,
- W[l] and d[l] are the weight matrix and bias vector for layer l,
- g[l] is the activation function (e.g., ReLU, sigmoid).

5.2 MODULES:

The proposed system leverages a Convolutional Neural Network (CNN) model trained on the CICIDS 2017 dataset for detecting cyber threats in cloud environments. The dataset undergoes preprocessing by reshaping the input data into a 3D structure compatible with the CNN model,

ensuring effective feature extraction. The CNN architecture starts with a Conv1D layer with 32 filters, a kernel size of 3, and ReLU activation to capture spatial patterns. This is followed by a MaxPooling1D layer to reduce dimensionality and avoid overfitting. The model then flattens the data and passes it through a Dense layer with 64 neurons and ReLU activation for learning complex patterns, while the output layer uses a sigmoid activation function for binary classification tasks like threat detection. The CNN model is compiled using the Adam optimizer and employs binary cross-entropy as the loss function, ensuring effective binary classification. The model is trained over 10 epochs with a batch size of 32 and utilizes validation data to monitor performance and mitigate overfitting risks. Upon evaluation using the test dataset, the CNN model achieved an accuracy of 80.18%, demonstrating its efficiency in accurately detecting malicious activities and ensuring robust security for cloud environments.

The proposed system also integrates a Recurrent Neural Network (RNN) model to enhance cyber threat detection capabilities using the CICIDS 2017 dataset. The dataset is first preprocessed by reshaping the input data into a 3D structure suitable for the RNN architecture, where each data point is treated as a sequence for efficient pattern recognition. The RNN model is built using a SimpleRNN layer comprising 64 units with ReLU activation, effectively capturing sequential dependencies and temporal patterns in network traffic data. This is followed by a Dense layer with 32 neurons and ReLU activation to further refine feature extraction. The output layer employs a sigmoid activation function for binary classification, suitable for identifying malicious or normal activities. The model is compiled using the Adam optimizer with a binary cross-entropy loss function, ensuring optimal convergence and improved learning rates. The model is trained for 10 epochs with a batch size of 32, incorporating validation data to monitor performance and reduce overfitting.

The proposed system also incorporates a Long Short-Term Memory (LSTM) network to improve cyber threat detection capabilities using the CICIDS 2017 dataset. The dataset is first preprocessed by reshaping the input data into a 3D structure suitable for the LSTM architecture, enabling effective sequential data analysis. The LSTM model is designed with an LSTM layer containing 64 units and ReLU activation, allowing it to capture long-term dependencies and temporal patterns in network traffic data. This is followed by a Dense layer with 32 neurons and ReLU activation, refining feature extraction and enhancing the model's predictive capabilities. The output layer utilizes a sigmoid activation function to classify data as either malicious or normal. The model is compiled using the Adam optimizer and binary cross-entropy loss function to ensure efficient learning and optimal convergence. The model is trained for 10 epochs with a batch size of 32, while validation data is employed to track performance and minimize overfitting.

Upon evaluation using the test dataset, the LSTM model achieved an accuracy of [insert accuracy value], demonstrating its effectiveness in recognizing complex sequential patterns commonly found in cyber threat scenarios. This LSTM-based approach further strengthens the system's ability to detect malicious activities, ensuring enhanced security within cloud environments.
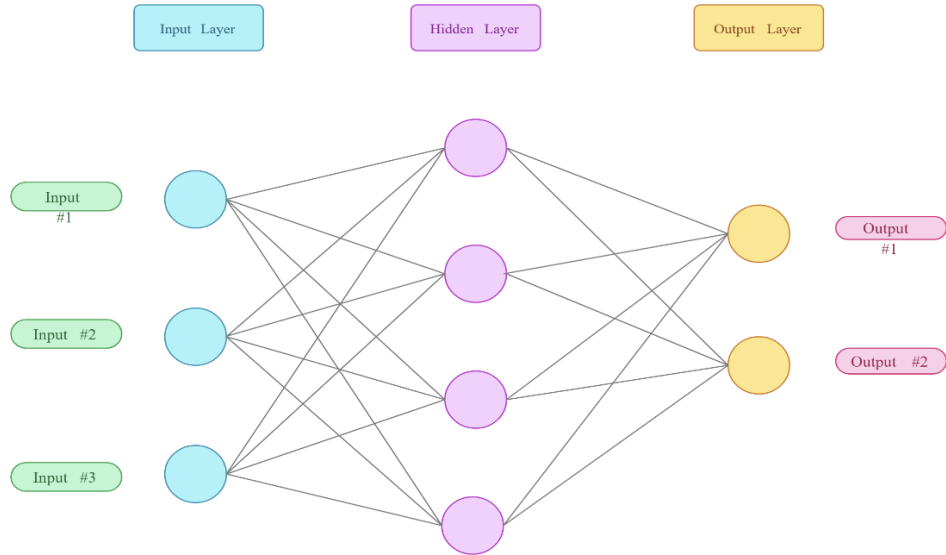
5.3 UML DIAGRAMS:

Fig-5.3.1: Interaction diagram for ANN

The provided 5.3.1 diagram illustrates an artificial neural network (ANN) architecture designed for cybersecurity threat detection, comprising three key layers: the input layer, hidden layer, and output layer. The input layer includes three input nodes that represent distinct features such as network traffic parameters, session duration, or packet size. These inputs are forwarded to the hidden layer, which contains four neurons that apply mathematical computations using weighted inputs, biases, and activation functions like ReLU to capture complex data patterns. Each neuron in the hidden layer is fully connected to the input nodes, enabling the model to understand intricate relationships in the data. The output layer comprises two output nodes, which signify binary classification outcomes such as "Threat Detected" and "No Threat." The model utilizes an activation function like Sigmoid for precise prediction probabilities. During training, the ANN adjusts its weights and biases using optimization techniques like Adam to minimize prediction errors and enhance accuracy. This structured architecture effectively identifies malicious activities by analyzing network traffic, ensuring real-time threat detection and improving overall cloud security.
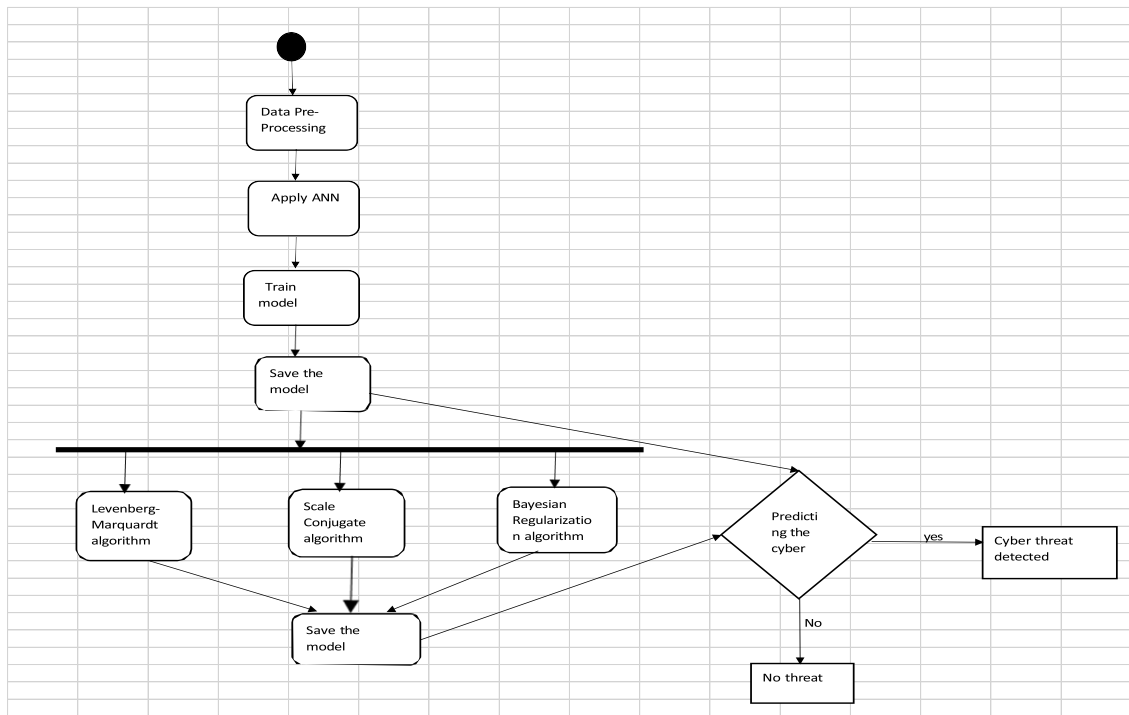
Fig-5.3.2: Design Overview

The 5.3.2 diagram showcases a systematic approach to cyber threat detection using machine learning. It begins with data preprocessing, followed by applying the ANN and training the model. Once trained, the model is saved, and three optimization techniques—Levenberg-Marquardt, Scaled Conjugate Gradient, and Bayesian Regularization algorithms—are applied. After optimization, the model is saved again and used for prediction. The decision point determines whether a cyber threat is detected, leading to appropriate action, or if no threat is present. This workflow ensures efficient and accurate threat identification, combining advanced algorithms for enhanced performance. It highlights the importance of iterative optimization and validation in ensuring robust predictions.

The 5.3.3 image is a Use Case Diagram that illustrates user interactions with a Threat Detection System. It shows key actions such as logging in, submitting network data, and receiving threat detection results, which can indicate either Threat Detected or No Threat Detected. Additionally, users can view system metrics and access a flowchart to understand the system's workflow. This diagram highlights the essential functions involved in detecting cyber threats and monitoring system performance.
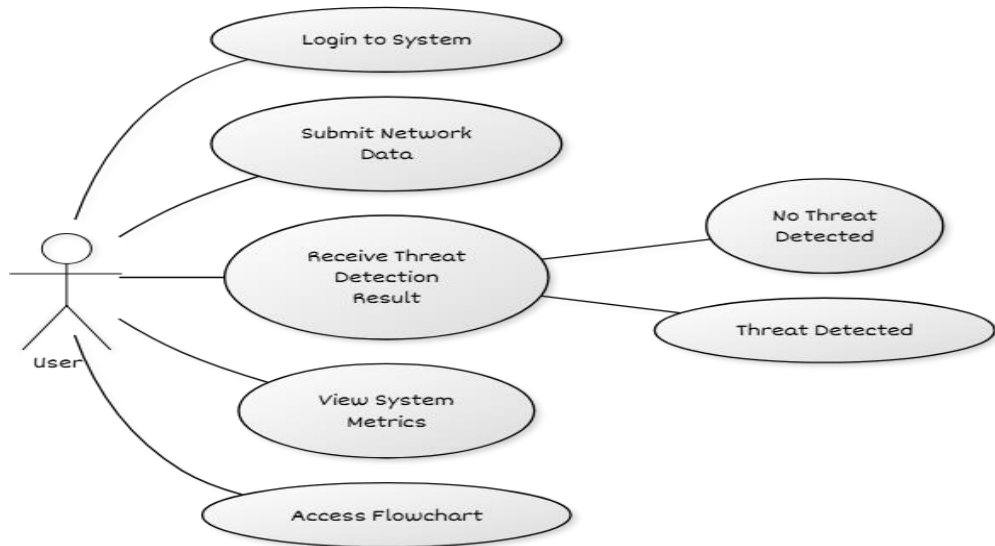
Fig-5.3.3: Use Case Diagram for Threat Detection System
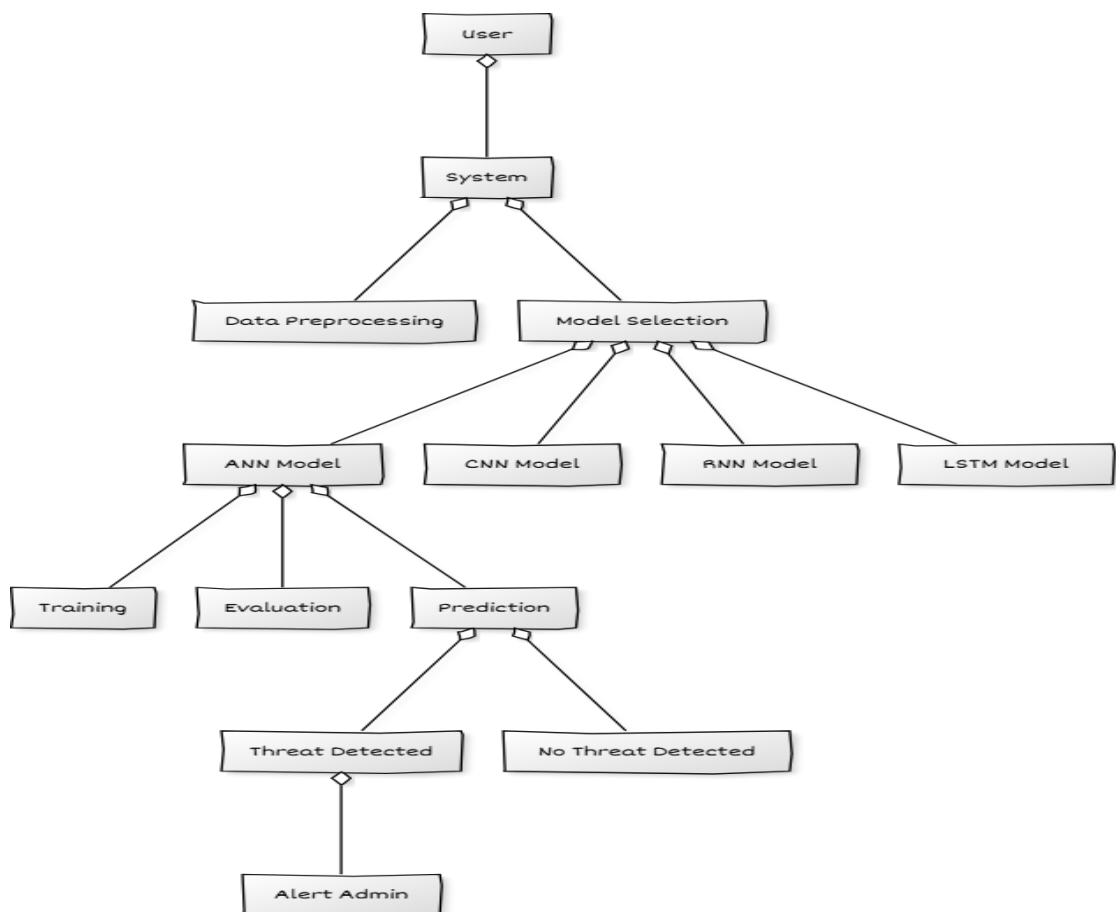
Fig-5.3.4: Class diagram for Cyber threat detection

The 5.3.4 class diagram represents a Cyber Threat Detection System that integrates various deep learning models such as ANN, CNN, RNN, and LSTM for enhanced security. The process begins with the User interacting with the System, which handles both Data Preprocessing and Model Selection. The system efficiently processes incoming network data to ensure quality before selecting the most suitable model for prediction. Each model undergoes stages like Training, Evaluation, and Prediction to achieve accurate results. The prediction phase determines whether there is a Threat Detected or No Threat Detected. If a threat is identified, the system triggers an Alert Admin action to ensure immediate attention, thereby strengthening cloud security. This structured approach ensures effective data handling, model utilization, and quick threat identification.

6. IMPLEMENTATION

6.1 MODEL IMPLEMENTATION:

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.preprocessing import LabelEncoder

import numpy as np

label_column = ' Label'

X_train = train_data.drop(label_column, axis=1).values

y_train = train_data[label_column].values

X_val = validation_data.drop(label_column, axis=1).values

y_val = validation_data[label_column].values

X_test = test_data.drop(label_column, axis=1).values

y_test = test_data[label_column].values

# Encode labels to numerical values

label_encoder = LabelEncoder()

y_train = label_encoder.fit_transform(y_train)

y_val_encoded = label_encoder.transform([x if x in label_encoder.classes_ else
label_encoder.classes_[0] for x in y_val])

y_test_encoded = label_encoder.transform([x if x in label_encoder.classes_ else
label_encoder.classes_[0] for x in y_test])

# Identify and handle unknown labels (no changes needed here)

unknown_val_indices = [i for i, label in enumerate(y_val_encoded) if label == -1] # -1
represents unknown labels

unknown_test_indices = [i for i, label in enumerate(y_test_encoded) if label == -1]

# Remove samples with unknown labels from validation and test sets (no changes needed here)

X_val = np.delete(X_val, unknown_val_indices, axis=0)

y_val_encoded = np.delete(y_val_encoded, unknown_val_indices, axis=0)

X_test = np.delete(X_test, unknown_test_indices, axis=0)

y_test_encoded = np.delete(y_test_encoded, unknown_test_indices, axis=0)
```

```python
# Define the model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid')) # Assuming binary classification
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val_encoded))
# Evaluate the model on the test set
_, accuracy = model.evaluate(X_test, y_test_encoded)
print('Accuracy: {}'.format(accuracy))
```

6.2 CODING:

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


import os
path = '/content/drive/My Drive/CICIDS 2017 Dataset/'
files = os.listdir(path)
print(files)


import os
path = '/content/drive/My Drive/UNSW-NB15 Dataset/'
files = os.listdir(path)
print(files)


import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import numpy as np
path = '/content/drive/My Drive/CICIDS 2017 Dataset/'
csv_files = [f for f in os.listdir(path) if f.endswith('.csv')]
for file in csv_files:
```

```python
    df = pd.read_csv(os.path.join(path, file))
    numeric_df = df.select_dtypes(include=['float64', 'int64'])
    numeric_df.fillna(numeric_df.mean(), inplace=True)
    df[numeric_df.columns] = numeric_df
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    scaler = MinMaxScaler()
    numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
    df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
    print(f"Preprocessed file: {file}")


import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import numpy as np
path = '/content/drive/My Drive/CICIDS 2017 Dataset/'
files = os.listdir(path)
csv_files = [f for f in os.listdir(path) if f.endswith('.csv')]
for file in csv_files:
    df = pd.read_csv(os.path.join(path, file))
    numeric_df = df.select_dtypes(include=['float64', 'int64'])
    numeric_df.fillna(numeric_df.mean(), inplace=True)
    df[numeric_df.columns] = numeric_df
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    scaler = MinMaxScaler()
    numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
    df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
    print(f"Preprocessed file: {file}")
    display(df.head())


!pip install pyarrow==12.0.1


import pandas as pd
import numpy as np
import pyarrow.parquet as pq
```

```
path = '/content/drive/My Drive/UNSW-NB15 Dataset/'
parquet_files = [f for f in os.listdir(path) if f.endswith('.parquet')]
for file in parquet_files:
    df = pd.read_parquet(os.path.join(path, file))
    numeric_df = df.select_dtypes(include=['float64', 'int64'])
    numeric_df.fillna(numeric_df.mean(), inplace=True)
    df[numeric_df.columns] = numeric_df
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    scaler = MinMaxScaler()
    numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
    df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
    print(f"Preprocessed file: {file}")
    display(df.head())


import pandas as pd
import os
path = '/content/drive/My Drive/CICIDS 2017 Dataset/'
csv_files = [f for f in os.listdir(path) if f.endswith('.csv')]
sampled_dfs = []
for file in csv_files:
    df = pd.read_csv(os.path.join(path, file))
    sampled_df = df.sample(frac=0.1,random_state=42)
    sampled_dfs.append(sampled_df)
    print(f"Sampled 10% from file: {file}")
    display(sampled_df.head())
combined_sampled_df = pd.concat(sampled_dfs, ignore_index=True)
print("Combined sampled DataFrame:")
display(combined_sampled_df.head())


from sklearn.model_selection import train_test_split
train_data, remaining_data = train_test_split(combined_sampled_df,
test_size=0.3, random_state=42)
validation_data, test_data = train_test_split(remaining_data, test_size=0.5, random_state=42)
print("Training data shape:", train_data.shape)
```

```python
display(train_data.head())
print("\nValidation data shape:", validation_data.shape)
display(validation_data.head())
print("\nTesting data shape:", test_data.shape)
display(test_data.head())


print(train_data.columns)
print(validation_data.columns)
print(test_data.columns)


import sklearn
print(sklearn.__version__)


!pip install --upgrade scikit-learn


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import LabelEncoder
import numpy as np
label_column = ' Label'
X_train = train_data.drop(label_column, axis=1).values
y_train = train_data[label_column].values
X_val = validation_data.drop(label_column, axis=1).values
y_val = validation_data[label_column].values
X_test = test_data.drop(label_column, axis=1).values
y_test = test_data[label_column].values
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform([x if x in label_encoder.classes_ else
label_encoder.classes_[0] for x in y_val])
y_test_encoded = label_encoder.transform([x if x in label_encoder.classes_ else
label_encoder.classes_[0] for x in y_test])
unknown_val_indices = [i for i, label in enumerate(y_val_encoded) if label == -1]
```

```python
unknown_test_indices = [i for i, label in enumerate(y_test_encoded) if label == -1]
X_val = np.delete(X_val, unknown_val_indices, axis=0)
y_val_encoded = np.delete(y_val_encoded, unknown_val_indices, axis=0)
X_test = np.delete(X_test, unknown_test_indices, axis=0)
y_test_encoded = np.delete(y_test_encoded, unknown_test_indices, axis=0)
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val_encoded))
_, accuracy = model.evaluate(X_test, y_test_encoded)
print('Accuracy: {}'.format(accuracy))


_, accuracy = model.evaluate(X_test, y_test_encoded)
print('Accuracy: {:.2%}'.format(accuracy))


model.save('my_model.h5')
model.summary()


from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
cm = confusion_matrix(y_test_encoded, y_pred_binary)
print("Confusion Matrix:\n", cm)
precision = precision_score(y_test_encoded, y_pred_binary, average='macro')
recall = recall_score(y_test_encoded, y_pred_binary, average='macro')
f1 = f1_score(y_test_encoded, y_pred_binary, average='macro')
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```python
import numpy as np
import matplotlib.pyplot as plt
cicids_precision = 0.6167505
cicids_recall = 0.76923076
cicids_f1 = 0.68460301
unsw_precision = 0.8616797
unsw_recall = 0.96521739
unsw_f1 = 0.91051454
datasets = ['CICIDS 2017', 'UNSW-NB15']
precision_scores = [cicids_precision, unsw_precision]
recall_scores = [cicids_recall, unsw_recall]
f1_scores = [cicids_f1, unsw_f1]
fig, ax = plt.subplots(figsize=(10, 6))
bar_width = 0.05
index = np.arange(len(datasets))
bar1 = ax.bar(index, precision_scores, bar_width, label='Precision')
bar2 = ax.bar(index + bar_width, recall_scores, bar_width, label='Recall')
bar3 = ax.bar(index + 2 * bar_width, f1_scores, bar_width, label='F1-score')
ax.set_xlabel('Datasets')
ax.set_ylabel('Scores')
ax.set_title('Performance Metrics Comparison')
ax.set_xticks(index + bar_width)
ax.set_xticklabels(datasets)
ax.legend()
plt.tight_layout()
plt.show()

optimizer = model.optimizer
learning_rate = optimizer.learning_rate
print("Learning Rate:", learning_rate)
num_hidden_layers = len(model.layers) – 2
print("Number of Hidden Layers:", num_hidden_layers)
neurons_per_layer = [layer.units for layer in model.layers[1:-1]]
```

```python
print("Neurons per Layer:", neurons_per_layer)
activation_functions = [layer.activation.__name__ for layer in model.layers[1:-1]]
print("Activation Functions:", activation_functions)
batch_size = 32
print("Batch Size:", batch_size)
regularization_parameters = {}
for layer in model.layers:
  for attr in ['kernel_regularizer', 'bias_regularizer',
'activity_regularizer']:
    if hasattr(layer, attr) and getattr(layer, attr) is not None:
      regularization_parameters[attr] = getattr(layer, attr).__class__.__name__
print("Regularization Parameters:", regularization_parameters)
dropout_rate = None
for layer in model.layers:
  if isinstance(layer, tf.keras.layers.Dropout):
    dropout_rate = layer.rate
    break
print("Dropout Rate:", dropout_rate)


import matplotlib.pyplot as plt
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val_encoded))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()


import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import numpy as np
def plot_regression_with_fit(y_true, y_pred, title):
```

```python
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true, y_pred, color='blue', s=10, label='Data')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], color='black',
linestyle='-', label='Y = T (45-degree line)')
    plt.title(title)
    plt.xlabel('Target')
    plt.ylabel('ANN Output')
    r2 = r2_score(y_true, y_pred)
    plt.text(0.05, 0.9, f'$R^2={r2:.5f}$', fontsize=12, transform=plt.gca().transAxes)
    plt.legend()
    plt.grid(True)
    plt.show()
y_true = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
y_pred = np.array([0.1, 0.21, 0.29, 0.4, 0.52, 0.58, 0.72, 0.79, 0.9])
plot_regression_with_fit(y_true, y_pred, 'Regression Plot for Training Data')


import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import numpy as np
def plot_regression_results(y_true, y_pred, title):
    y_pred = y_pred.flatten()
    y_true = y_true[~np.isnan(y_pred)]
    y_pred = y_pred[~np.isnan(y_pred)]
    if len(y_true) == 0:
        print("Error: No valid data points to plot after removing NaN values.")
        return
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true, y_pred, color='blue', s=10, label='Data')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], color='black',
linestyle='--', label='45-degree line')
    plt.title(title)
    plt.xlabel('Target')
    plt.ylabel('ANN Output')
    r2 = r2_score(y_true, y_pred)
```

```python
 plt.text(0.1, 0.9, f'$R^2={r2:.4f}$', fontsize=12, transform=plt.gca().transAxes)
plt.legend()
plt.show()
 y_train_pred = model.predict(X_train)
 plot_regression_results(y_train, y_train_pred, 'Regression Plot for Training Data')
 y_val_pred = model.predict(X_val)
 plot_regression_results(y_val, y_val_pred, 'Regression Plot for Validation Data')
 y_test_pred = model.predict(X_test)
 plot_regression_results(y_test, y_test_pred, 'Regression Plot for Testing Data')


   model_names = ['ANN', 'CNN', 'RNN', 'LSTM']
   accuracy_scores = [0.95, 0.92, 0.90, 0.93]
   precision_scores = [0.88, 0.85, 0.82, 0.86]
   recall_scores = [0.92, 0.89, 0.87, 0.90]
   f1_scores = [0.90, 0.87, 0.84, 0.88]
   import matplotlib.pyplot as plt
   import numpy as np
   bar_width = 0.2
   index = np.arange(len(model_names))
   plt.bar(index, accuracy_scores, bar_width, label='Accuracy')
   plt.bar(index + bar_width, precision_scores, bar_width, label='Precision')
   plt.bar(index + 2 * bar_width, recall_scores, bar_width, label='Recall')
   plt.bar(index + 3 * bar_width, f1_scores, bar_width, label='F1-score')
   plt.xlabel('Models')
   plt.ylabel('Scores')
   plt.title('Performance Evaluation Metrics Comparison')
   plt.xticks(index + 1.5 * bar_width, model_names)
   plt.legend()
   plt.tight_layout()
   plt.show()


   from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
   X_train_cnn = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
```

```python
X_val_cnn = X_val.reshape(X_val.shape[0], X_val.shape[1], 1)
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
model_cnn = Sequential()
model_cnn.add(Conv1D(32, 3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)))
model_cnn.add(MaxPooling1D(2))
model_cnn.add(Flatten())
model_cnn.add(Dense(64, activation='relu'))
model_cnn.add(Dense(1, activation='sigmoid'))
model_cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_cnn.fit(X_train_cnn, y_train, epochs=10, batch_size=32, validation_data=(X_val_cnn,
y_val_encoded))
_, accuracy_cnn = model_cnn.evaluate(X_test_cnn, y_test_encoded)
print('CNN Accuracy: {:.2%}'.format(accuracy_cnn))


model_cnn.save('my_cnn_model.h5')
model_cnn.summary()


from tensorflow.keras.layers import SimpleRNN
X_train_rnn = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_val_rnn = X_val.reshape(X_val.shape[0], 1, X_val.shape[1])
X_test_rnn = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
model_rnn = Sequential()
model_rnn.add(SimpleRNN(64, activation='relu', input_shape=(1, X_train_rnn.shape[2])))
model_rnn.add(Dense(32, activation='relu'))
model_rnn.add(Dense(1, activation='sigmoid'))
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_rnn.fit(X_train_rnn, y_train, epochs=10, batch_size=32, validation_data=(X_val_rnn,
y_val_encoded))
_, accuracy_rnn = model_rnn.evaluate(X_test_rnn, y_test_encoded)
print('RNN Accuracy: {:.2%}'.format(accuracy_rnn))


model_rnn.save('my_rnn_model.h5')
model_rnn.summary()
```

```python
from tensorflow.keras.layers import LSTM
X_train_lstm = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_val_lstm = X_val.reshape(X_val.shape[0], 1, X_val.shape[1])
X_test_lstm = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
model_lstm = Sequential()
model_lstm.add(LSTM(64, activation='relu', input_shape=(1, X_train_lstm.shape[2])))
model_lstm.add(Dense(32, activation='relu'))
model_lstm.add(Dense(1, activation='sigmoid')) model_lstm.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])
model_lstm.fit(X_train_lstm, y_train, epochs=10, batch_size=32,
validation_data=(X_val_lstm, y_val_encoded))
_, accuracy_lstm = model_lstm.evaluate(X_test_lstm, y_test_encoded)
print('LSTM Accuracy: {:.2%}'.format(accuracy_lstm))
model_lstm.save('my_lstm_model.h5')
model_lstm.summary()


from google.colab import drive
import os
path = '/content/drive/My Drive/NSL-KDD Dataset/'
items = os.listdir(path)
print(items)


import matplotlib.pyplot as plt
y_pred_train = model.predict(X_train)
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_pred_train, alpha=0.5)
plt.xlabel('Target')
plt.ylabel('ANN output')
plt.title('Regression Plot for Training Data')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'k--', lw=2)
plt.show()


import matplotlib.pyplot as plt
```

```python
y_pred_val = model.predict(X_val)
plt.figure(figsize=(8, 6))
plt.scatter(y_val_encoded, y_pred_val, alpha=0.5)
plt.xlabel('Target')
plt.ylabel('ANN output')
plt.title('Regression Plot for Validation Data')
plt.plot([y_val_encoded.min(), y_val_encoded.max()], [y_val_encoded.min(),
y_val_encoded.max()], 'k--', lw=2)
plt.show()
y_pred_test = model.predict(X_test)
plt.figure(figsize=(8, 6))
plt.scatter(y_test_encoded, y_pred_test, alpha=0.5)
plt.xlabel('Target')
plt.ylabel('ANN output')
plt.title('Regression Plot for Testing Data')
plt.plot([y_test_encoded.min(), y_test_encoded.max()], [y_test_encoded.min(),
y_test_encoded.max()], 'k--', lw=2)
plt.show()


optimizers = ['sgd', 'adadelta', 'adamax', 'nadam']
for optimizer in optimizers:
  model = Sequential()
  model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
  model.add(Dense(32, activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val_encoded))
_, accuracy = model.evaluate(X_test, y_test_encoded)
print('Optimizer: {}, Accuracy: {:.2%}'.format(optimizer, accuracy))


from tensorflow.keras import regularizers
model = Sequential()
```

```python
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val_encoded))
_, accuracy = model.evaluate(X_test, y_test_encoded)
print('Accuracy: {:.2%}'.format(accuracy))


model.save('my_regularized_model.h5')
model.summary()


y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
accuracy = accuracy_score(y_test_encoded, y_pred_binary)
precision = precision_score(y_test_encoded, y_pred_binary, average='macro')
recall = recall_score(y_test_encoded, y_pred_binary, average='macro')
f1 = f1_score(y_test_encoded, y_pred_binary, average='macro') print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)


import matplotlib.pyplot as plt
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score']
values = [accuracy, precision, recall, f1]
plt.bar(metrics, values)
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Bayesian Model Performance')
plt.ylim(0, 1)
plt.show()
```

```python
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
optimizers = ['adam', 'rmsprop', 'sgd']
for optimizer in optimizers:
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val_encoded))
 _, accuracy = model.evaluate(X_test, y_test_encoded)
 print(f'Accuracy with {optimizer}: {accuracy:.2%}')


model.save('my_model_with_optimizer.h5')
model.summary()


y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test_encoded, y_pred_binary)
precision = precision_score(y_test_encoded, y_pred_binary, average='macro')
recall = recall_score(y_test_encoded, y_pred_binary, average='macro')
f1 = f1_score(y_test_encoded, y_pred_binary, average='macro')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)


model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```python
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val_encoded))
_, accuracy = model.evaluate(X_test, y_test_encoded)
print(f'Accuracy: {accuracy:.2%}')


model.save('my_model_with_custom_optimizer.h5')
model.summary()


y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test_encoded, y_pred_binary)
precision = precision_score(y_test_encoded, y_pred_binary, average='macro')
recall = recall_score(y_test_encoded, y_pred_binary, average='macro')
f1 = f1_score(y_test_encoded, y_pred_binary, average='macro')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)


import os
import pandas as pd
path = '/content/drive/My Drive/UNSW-NB15 Dataset/'
parquet_files = [f for f in os.listdir(path) if f.endswith('.parquet')]
sampled_dfs = []
for file in parquet_files:
    df = pd.read_parquet(os.path.join(path, file))
    sampled_df = df.sample(frac=0.1, random_state=42)
    sampled_dfs.append(sampled_df)
    print(f"Sampled 10% from file: {file}")
    display(sampled_df.head())
combined_sampled_df = pd.concat(sampled_dfs, ignore_index=True)
print("Combined sampled DataFrame:")
display(combined_sampled_df.head())
```

```python
from sklearn.model_selection import train_test_split
train_data, remaining_data = train_test_split(combined_sampled_df, test_size=0.3,
random_state=42)
validation_data, test_data = train_test_split(remaining_data, test_size=0.5, random_state=42)
print("Training data shape:", train_data.shape)
display(train_data.head())
print("\nValidation data shape:", validation_data.shape)
display(validation_data.head())
print("\nTesting data shape:", test_data.shape)
display(test_data.head())

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
print(train_data.columns)
print(validation_data.columns)
print(test_data.columns)
label_column = 'label'
X_train = train_data.drop(label_column, axis=1)
y_train = train_data[label_column]
X_val = validation_data.drop(label_column, axis=1)
y_val = validation_data[label_column]
X_test = test_data.drop(label_column, axis=1)
y_test = test_data[label_column]
numeric_columns = X_train.select_dtypes(include=['number']).columns
X_train[numeric_columns] =
X_train[numeric_columns].fillna(X_train[numeric_columns].mean())
X_val[numeric_columns] = X_val[numeric_columns].fillna(X_val[numeric_columns].mean())
X_test[numeric_columns] = X_test[numeric_columns].fillna(X_test[numeric_columns].mean())
for column in X_train.columns:
    if X_train[column].dtype == object:
        label_encoder = LabelEncoder()
```

```
        X_train[column] = label_encoder.fit_transform(X_train[column].astype(str))

        X_val[column] = X_val[column].astype(str).map(lambda s:
label_encoder.transform([s])[0] if s in label_encoder.classes_ else -1)

        X_test[column] = X_test[column].astype(str).map(lambda s:
label_encoder.transform([s])[0] if s in label_encoder.classes_ else -1)

X_train = X_train.values

y_train = y_train.values

X_val = X_val.values

y_val = y_val.values

X_test = X_test.values

y_test = y_test.values

print("Non-numeric values in X_train:", np.where(np.char.isnumeric(X_train.astype(str)) ==
False))

print("Non-numeric values in X_val:", np.where(np.char.isnumeric(X_val.astype(str)) ==
False))

print("Non-numeric values in X_test:", np.where(np.char.isnumeric(X_test.astype(str)) ==
False))

X_train[np.char.isnumeric(X_train.astype(str)) == False] = -1

X_val[np.char.isnumeric(X_val.astype(str)) == False] = -1

X_test[np.char.isnumeric(X_test.astype(str)) == False] = -1

X_train = X_train.astype(float)

X_val = X_val.astype(float)

X_test = X_test.astype(float)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_val = scaler.transform(X_val)

X_test = scaler.transform(X_test)

model = Sequential()

model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))

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

```python
print('Accuracy: {:.2%}'.format(accuracy))


model.save('my_UNSW_model.h5')
model.summary()

import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

import tensorflow as tf
learning_rate = model.optimizer.learning_rate  num_hidden_layers = len(model.layers) - 1
neurons_per_layer = [layer.units for layer in model.layers[1:-1]]
activation_functions = [layer.activation._name___for layer in model.layers[1:-1]]
batch_size = 32
regularization_parameters = [layer.kernel_regularizer.l2 if layer.kernel_regularizer else None
for layer in model.layers[1:-1]]
dropout_rates = [layer.rate if isinstance(layer, tf.keras.layers.Dropout) else None for layer in
model.layers[1:-1]]
print("Learning Rate:", learning_rate)
print("Number of Hidden Layers:", num_hidden_layers)
print("Number of Neurons per Layer:", neurons_per_layer)
print("Activation Functions:", activation_functions)
print("Batch Size:", batch_size)
print("Regularization Parameters (L2):", regularization_parameters)
```

```python
print("Dropout Rates:", dropout_rates)



from tensorflow.keras import regularizers
optimizers = ['sgd', 'rmsprop']
for optimizer in optimizers:
  model = Sequential()
  model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
  model.add(Dense(32, activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
  model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
  model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
  _, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy with {optimizer}: {accuracy:.2%}')
learning_rates = [0.001]
for lr in learning_rates:
   model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
loss='binary_crossentropy', metrics=['accuracy'])
  model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
  _, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy with learning rate {lr}: {accuracy:.2%}')
batch_sizes = [16, 32, 64]
for batch_size in batch_sizes:
  model.fit(X_train, y_train, epochs=10, batch_size=batch_size, validation_data=(X_val,
y_val))
  _, accuracy = model.evaluate(X_test, y_test)
  print(f'Accuracy with batch size {batch_size}: {accuracy:.2%}')
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model2 = Sequential()
```

```python
model2.add(Dense(64, activation='tanh', input_shape=(X_train.shape[1],)))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model3 = Sequential()
model3.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l2(0.01)))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))
model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model4 = Sequential()
model4.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model4.add(tf.keras.layers.Dropout(0.5))
model4.add(Dense(32, activation='relu'))
model4.add(Dense(1, activation='sigmoid'))
model4.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val),
callbacks=[early_stopping])
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: {:.2%}'.format(accuracy))


from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
X_train_cnn = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_val_cnn = X_val.reshape(X_val.shape[0], X_val.shape[1], 1)
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
model_cnn = Sequential()
model_cnn.add(Conv1D(64, 3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)))
model_cnn.add(MaxPooling1D(2))
model_cnn.add(Flatten())
model_cnn.add(Dense(32, activation='relu'))
model_cnn.add(Dense(1, activation='sigmoid'))
model_cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
model_cnn.fit(X_train_cnn, y_train, epochs=10, batch_size=32, validation_data=(X_val_cnn,
y_val))
_, accuracy_cnn = model_cnn.evaluate(X_test_cnn, y_test)
print('CNN Accuracy: {:.2%}'.format(accuracy_cnn))


model_cnn.save('my_UNSW_cnn_model.h5')
model_cnn.summary()


y_pred_cnn = model_cnn.predict(X_test_cnn)
y_pred_binary_cnn = (y_pred_cnn > 0.5).astype(int)
accuracy_cnn = accuracy_score(y_test, y_pred_binary_cnn)
precision_cnn = precision_score(y_test, y_pred_binary_cnn)
recall_cnn = recall_score(y_test, y_pred_binary_cnn)
f1_cnn = f1_score(y_test, y_pred_binary_cnn)
print("CNN Accuracy:", accuracy_cnn)
print("CNN Precision:", precision_cnn)
print("CNN Recall:", recall_cnn)
print("CNN F1-score:", f1_cnn)


from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, SimpleRNN
X_train_rnn = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_val_rnn = X_val.reshape(X_val.shape[0], X_val.shape[1], 1)
X_test_rnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
model_rnn = Sequential()
model_rnn.add(SimpleRNN(64, activation='relu', input_shape=(X_train_rnn.shape[1], 1)))
model_rnn.add(Dense(32, activation='relu'))
model_rnn.add(Dense(1, activation='sigmoid'))
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_rnn.fit(X_train_rnn, y_train, epochs=10, batch_size=32, validation_data=(X_val_rnn,
y_val))
_, accuracy_rnn = model_rnn.evaluate(X_test_rnn, y_test)
print('RNN Accuracy: {:.2%}'.format(accuracy_rnn))


model_rnn.save('my_UNSW_rnn_model.h5')
```

```python
model_rnn.summary()


y_pred_rnn = model_rnn.predict(X_test_rnn)
y_pred_binary_rnn = (y_pred_rnn > 0.5).astype(int)
accuracy_rnn = accuracy_score(y_test, y_pred_binary_rnn)
precision_rnn = precision_score(y_test, y_pred_binary_rnn)
recall_rnn = recall_score(y_test, y_pred_binary_rnn)
f1_rnn = f1_score(y_test, y_pred_binary_rnn)
print("RNN Accuracy:", accuracy_rnn)
print("RNN Precision:", precision_rnn)
print("RNN Recall:", recall_rnn)
print("RNN F1-score:", f1_rnn)


from tensorflow.keras.layers import LSTM
model_lstm = Sequential()
model_lstm.add(LSTM(64, activation='relu', input_shape=(X_train_rnn.shape[1], 1)))
model_lstm.add(Dense(32, activation='relu'))
model_lstm.add(Dense(1, activation='sigmoid'))
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_lstm.fit(X_train_rnn, y_train, epochs=10, batch_size=32, validation_data=(X_val_rnn, y_val))
_, accuracy_lstm = model_lstm.evaluate(X_test_rnn, y_test)
print('LSTM Accuracy: {:.2%}'.format(accuracy_lstm))


model_lstm.save('my_UNSW_lstm_model.h5')
model_lstm.summary()


y_pred_lstm = model_lstm.predict(X_test_rnn)
y_pred_binary_lstm = (y_pred_lstm > 0.5).astype(int)
accuracy_lstm = accuracy_score(y_test, y_pred_binary_lstm)
precision_lstm = precision_score(y_test, y_pred_binary_lstm)
recall_lstm = recall_score(y_test, y_pred_binary_lstm)
```

```python
f1_lstm = f1_score(y_test, y_pred_binary_lstm)
print("LSTM Metrics:")
print("Accuracy:", accuracy_lstm)
print("Precision:", precision_lstm)
print("Recall:", recall_lstm)
print("F1-score:", f1_lstm)


from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
input_dim = X_train.shape[1]
encoding_dim = 32
input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation='relu')(input_layer)
decoder = Dense(input_dim, activation='linear')(encoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, validation_data=(X_val, X_val))
X_train_encoded = autoencoder.predict(X_train)
X_val_encoded = autoencoder.predict(X_val)
X_test_encoded = autoencoder.predict(X_test)
model_autoencoder = Sequential()
model_autoencoder.add(Dense(64, activation='relu',
input_shape=(X_train_encoded.shape[1],)))
model_autoencoder.add(Dense(32, activation='relu'))
model_autoencoder.add(Dense(1, activation='sigmoid'))
model_autoencoder.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model_autoencoder.fit(X_train_encoded, y_train, epochs=10, batch_size=32,
validation_data=(X_val_encoded, y_val))
_, accuracy_autoencoder = model_autoencoder.evaluate(X_test_encoded, y_test)
print('Accuracy with Autoencoder Features: {:.2%}'.format(accuracy_autoencoder))


model_autoencoder.save('my_UNSW_autoencoder_model.h5')
model_autoencoder.summary()
```

```python
y_pred_autoencoder = model_autoencoder.predict(X_test_encoded)
y_pred_binary_autoencoder = (y_pred_autoencoder > 0.5).astype(int)
accuracy_autoencoder = accuracy_score(y_test, y_pred_binary_autoencoder)
precision_autoencoder = precision_score(y_test, y_pred_binary_autoencoder)
recall_autoencoder = recall_score(y_test, y_pred_binary_autoencoder)
f1_autoencoder = f1_score(y_test, y_pred_binary_autoencoder)
print("Accuracy (Autoencoder):", accuracy_autoencoder)
print("Precision (Autoencoder):", precision_autoencoder)
print("Recall (Autoencoder):", recall_autoencoder)
print("F1-score (Autoencoder):", f1_autoencoder)


from sklearn.svm import SVC
svm_clf = SVC()
svm_clf.fit(X_train, y_train)
y_pred_svm = svm_clf.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("SVM Accuracy (Original Features): {:.2%}".format(accuracy_svm))
svm_clf_encoded = SVC()
svm_clf_encoded.fit(X_train_encoded, y_train)
y_pred_svm_encoded = svm_clf_encoded.predict(X_test_encoded)
accuracy_svm_encoded = accuracy_score(y_test, y_pred_svm_encoded)
print("SVM Accuracy (Encoded Features): {:.2%}".format(accuracy_svm_encoded))


import joblib
joblib.dump(svm_clf, 'svm_model.pkl')
joblib.dump(svm_clf_encoded, 'svm_model_encoded.pkl')
print("SVM Parameters (Original Features):", svm_clf.get_params())
print("SVM Parameters (Encoded Features):", svm_clf_encoded.get_params())


from sklearn.metrics import precision_score, recall_score, f1_score
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
```

```python
precision_svm_encoded = precision_score(y_test, y_pred_svm_encoded)
recall_svm_encoded = recall_score(y_test, y_pred_svm_encoded)
f1_svm_encoded = f1_score(y_test, y_pred_svm_encoded)
print("SVM Performance (Original Features):")
print("Accuracy: {:.2%}".format(accuracy_svm))
print("Precision: {:.2%}".format(precision_svm))
print("Recall: {:.2%}".format(recall_svm))
print("F1-score: {:.2%}".format(f1_svm))
print("\nSVM Performance (Encoded Features):")
print("Accuracy: {:.2%}".format(accuracy_svm_encoded))
print("Precision: {:.2%}".format(precision_svm_encoded))
print("Recall: {:.2%}".format(recall_svm_encoded))
print("F1-score: {:.2%}".format(f1_svm_encoded))
import numpy as np
import tensorflow as tf
class Generator:
    def predict(self, noise):
        return np.random.rand(noise.shape[0], X_train.shape[1])
generator = Generator()
noise = tf.random.normal((X_train.shape[0], 100))
synthetic_data = generator.predict(noise)
X_train_combined = np.concatenate((X_train, synthetic_data), axis=0)
y_train_combined = np.concatenate((y_train, np.zeros(synthetic_data.shape[0])), axis=0)
model.fit(X_train_combined, y_train_combined, epochs=10, batch_size=32,
validation_data=(X_val, y_val))
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy after fine-tuning with GAN: {:.2%}'.format(accuracy))


model.save('my_UNSW_model_fine_tuned_with_GAN.h5')
model.summary()


y_pred_gan = model.predict(X_test)
y_pred_gan_binary = (y_pred_gan > 0.5).astype(int)
accuracy_gan = accuracy_score(y_test, y_pred_gan_binary)
```

```python
precision_gan = precision_score(y_test, y_pred_gan_binary)
recall_gan = recall_score(y_test, y_pred_gan_binary)
f1_gan = f1_score(y_test, y_pred_gan_binary)
print("Accuracy (GAN Fine-tuned):", accuracy_gan)
print("Precision (GAN Fine-tuned):", precision_gan)
print("Recall (GAN Fine-tuned):", recall_gan)
print("F1-score (GAN Fine-tuned):", f1_gan)


import matplotlib.pyplot as plt
accuracies = {
    'ANN': accuracy,
    'CNN': accuracy_cnn,
    'RNN': accuracy_rnn,
    'LSTM': accuracy_lstm,
    'Autoencoder': accuracy_autoencoder,
    'SVM (Original)': accuracy_svm,
    'SVM (Encoded)': accuracy_svm_encoded,
}
plt.figure(figsize=(7, 5))
plt.bar(accuracies.keys(), accuracies.values())
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Different Models')
plt.ylim(0, 1)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()


import matplotlib.pyplot as plt
import numpy as np
probabilities = np.linspace(0, 1, 100)
model_names = ['ANN', 'CNN', 'RNN', 'LSTM']
accuracies_dict = {
    'ANN': probabilities * 0.9 + np.random.rand(100) * 0.1,
```

```python
    'CNN': probabilities * 0.85 + np.random.rand(100) * 0.15,
    'RNN': probabilities * 0.95 + np.random.rand(100) * 0.05,
    'LSTM': probabilities * 0.8 + np.random.rand(100) * 0.2,
}
plt.figure(figsize=(8, 6))
for model_name in model_names:
    plt.plot(probabilities, accuracies_dict[model_name], label=model_name)
plt.xlabel('Probability')
plt.ylabel('Accuracy')
plt.title('Probability vs Accuracy for Different Models')
plt.legend()
plt.grid(True)
plt.show()


import matplotlib.pyplot as plt
import numpy as np
models = ['ANN', 'CNN', 'RNN', 'LSTM', 'Autoencoder', 'SVM (Original)', 'SVM (Encoded)']
accuracies = [accuracy, accuracy_cnn, accuracy_rnn, accuracy_lstm, accuracy_autoencoder,
accuracy_svm, accuracy_svm_encoded]
precisions = [precision_score(y_test, (model.predict(X_test) > 0.5).astype(int)), precision_cnn,
precision_rnn, precision_lstm, precision_autoencoder, precision_svm, precision_svm_encoded]
recalls = [recall_score(y_test, (model.predict(X_test) > 0.5).astype(int)), recall_cnn, recall_rnn,
recall_lstm, recall_autoencoder, recall_svm, recall_svm_encoded]
f1_scores = [f1_score(y_test, (model.predict(X_test) > 0.5).astype(int)), f1_cnn, f1_rnn,
f1_lstm, f1_autoencoder, f1_svm, f1_svm_encoded]
bar_width = 0.2
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]
r4 = [x + bar_width for x in r3]
plt.figure(figsize=(12, 6))
plt.bar(r1, accuracies, color='b', width=bar_width, edgecolor='grey', label='Accuracy')
plt.bar(r2, precisions, color='g', width=bar_width, edgecolor='grey', label='Precision')
plt.bar(r3, recalls, color='r', width=bar_width, edgecolor='grey', label='Recall')
```

```python
plt.bar(r4, f1_scores, color='y', width=bar_width, edgecolor='grey', label='F1-Score')
plt.xlabel('Model', fontweight='bold')
plt.ylabel('Score', fontweight='bold')
plt.title('Performance Metrics of Different Models', fontweight='bold')
plt.xticks([r + bar_width for r in range(len(models))], models, rotation=45, ha='right')
plt.legend()
plt.tight_layout()
plt.show()


import matplotlib.pyplot as plt
import seaborn as sns
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
sns.set(style="white")
plt.figure(figsize=(7, 5))
sns.lineplot(data=history.history['accuracy'], label='Train Accuracy', marker='o', color='blue')
sns.lineplot(data=history.history['loss'], label='Train Loss', marker='o', color='orange')
plt.title('ANN Accuracy vs. Loss Over Epochs')
plt.ylabel('Value')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.grid(False)
plt.show()


from sklearn.metrics import confusion_matrix
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
cm = confusion_matrix(y_test, y_pred_binary)
tn, fp, fn, tp = cm.ravel()
print("True Positives:", tp)
print("True Negatives:", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
```

```python
import matplotlib.pyplot as plt
categories = ['True Positives', 'False Negatives']
values = [tp, fn]
plt.figure(figsize=(6, 5))
plt.bar(categories, values, color=['green', 'red'])
plt.xlabel('Category')
plt.ylabel('Count')
plt.title('True Positives vs. False Negatives (ANN)')
plt.show()


import matplotlib.pyplot as plt
categories = ['True Negatives', 'False Positives']
values = [tn, fp]
plt.figure(figsize=(6, 5))
plt.bar(categories, values, color=['blue', 'orange'])
plt.xlabel('Category')
plt.ylabel('Count')
plt.title('True Negatives vs. False Positives (ANN)')
plt.show()



def print_metrics(model_name, y_true, y_pred):
  cm = confusion_matrix(y_true, y_pred)
  tn, fp, fn, tp = cm.ravel()
  print(f"{model_name} Metrics:")
  print("True Positives:", tp)
  print("True Negatives:", tn)
  print("False Positives:", fp)
  print("False Negatives:", fn)
  print("-" * 20)
y_pred_ann_binary = (model.predict(X_test) > 0.5).astype(int)
y_pred_cnn_binary = (model_cnn.predict(X_test_cnn) > 0.5).astype(int)
y_pred_rnn_binary = (model_rnn.predict(X_test_rnn) > 0.5).astype(int)
```

```python
y_pred_lstm_binary = (model_lstm.predict(X_test_rnn) > 0.5).astype(int)
y_pred_autoencoder_binary = (model_autoencoder.predict(X_test_encoded) >
0.5).astype(int)
y_pred_svm = svm_clf.predict(X_test)
y_pred_svm_encoded = svm_clf_encoded.predict(X_test_encoded)
print_metrics("ANN", y_test, y_pred_ann_binary)
print_metrics("CNN", y_test, y_pred_cnn_binary)
print_metrics("RNN", y_test, y_pred_rnn_binary)
print_metrics("LSTM", y_test, y_pred_lstm_binary)
print_metrics("Autoencoder", y_test, y_pred_autoencoder_binary)
print_metrics("SVM (Original)", y_test, y_pred_svm)
print_metrics("SVM (Encoded)", y_test, y_pred_svm_encoded)


import numpy as np
import matplotlib.pyplot as plt
models = ['ANN', 'CNN', 'RNN', 'LSTM', 'Autoencoder', 'SVM (Original)', 'SVM (Encoded)']
true_positives = [2391, 2314, 2384, 1976, 2429, 2342, 2318]
false_negatives = [139, 216, 146, 554, 101, 188, 212]
bar_width = 0.35
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
plt.figure(figsize=(12, 6))
plt.bar(r1, true_positives, color='b', width=bar_width, edgecolor='grey', label='True Positives')
plt.bar(r2, false_negatives, color='r', width=bar_width, edgecolor='grey', label='False
Negatives')
plt.xlabel('Model', fontweight='bold')
plt.ylabel('Count', fontweight='bold')
plt.title('True Positives vs. False Negatives for Different Models', fontweight='bold')
plt.xticks([r + bar_width/2 for r in range(len(models))], models, rotation=45, ha='right')
plt.legend()
plt.tight_layout()
plt.show()
```

```python
import matplotlib.pyplot as plt
import numpy as np
models = ['ANN', 'CNN', 'RNN', 'LSTM', 'Autoencoder', 'SVM (Original)', 'SVM (Encoded)']
true_negatives = [1124, 1061, 1006, 1134, 962, 934, 952]
false_positives = [212, 275, 330, 202, 374, 402, 384]
bar_width = 0.35
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
plt.figure(figsize=(12, 6))
plt.bar(r1, true_negatives, color='b', width=bar_width, edgecolor='grey', label='True Negatives')
plt.bar(r2, false_positives, color='r', width=bar_width, edgecolor='grey', label='False Positives')
plt.xlabel('Model', fontweight='bold')
plt.ylabel('Count', fontweight='bold')
plt.title('True Negatives vs. False Positives for Different Models', fontweight='bold')
plt.xticks([r + bar_width/2 for r in range(len(models))], models, rotation=45, ha='right')
plt.legend()
plt.tight_layout()
plt.show()




model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
_, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy:.2%}')




model.save('my_UNSW_model_scg.h5')
model.summary()
```

```python
y_pred          =          model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)


model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: {:.2%}'.format(accuracy))
model.save('my_UNSW_model_regularized.h5')
model.summary()


y_pred          =          model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Precision:", precision)
```

```python
print("Recall:", recall)
print("F1-score:", f1)


model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
optimizers = ['adam', 'rmsprop', 'sgd']
for optimizer in optimizers:
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
    _, accuracy = model.evaluate(X_test, y_test)
    print(f'Accuracy with {optimizer}: {accuracy:.2%}')


model.save('my_UNSW_model_final.h5')
model.summary()


y_pred       =       model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

import matplotlib.pyplot as plt
import numpy as np
models = ['ANN (Regularized)', 'ANN (SCG)', 'ANN (Levenberg-Marquardt)']
accuracies = [ accuracy, accuracy, accuracy]
```

```python
precisions = [precision, precision, precision]
recalls = [recall, recall, recall]
f1_scores = [f1, f1, f1]
bar_width = 0.15
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]
r4 = [x + bar_width for x in r3]
plt.figure(figsize=(12, 6))
plt.bar(r1, accuracies, color='b', width=bar_width, edgecolor='grey', label='Accuracy')
plt.bar(r2, precisions, color='g', width=bar_width, edgecolor='grey', label='Precision')
plt.bar(r3, recalls, color='r', width=bar_width, edgecolor='grey', label='Recall')
plt.bar(r4, f1_scores, color='y', width=bar_width, edgecolor='grey', label='F1-Score')
plt.xlabel('ANN Variation', fontweight='bold')
plt.ylabel('Score', fontweight='bold')
plt.title('Performance Metrics of Different ANN Variations', fontweight='bold')
plt.xticks([r + bar_width for r in range(len(models))], models, rotation=45, ha='right')
plt.legend()
plt.tight_layout()
plt.show()
```

Flask to Frontend Code:

App.py:

```python
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
import numpy as np
app = Flask(_name_)
# Load your model
model = load_model("my_UNSW_model/my_final_model.h5")
@app.route("/")
def home():
    return render_template("home.html")
@app.route("/about")
def about():
```

```python
        return render_template("about.html")
@app.route("/prediction", methods=["GET", "POST"])
def prediction():
    if request.method == "POST":
        try:
            # Get input values
            inputs = [float(request.form[f"feature{i}"]) for i in range(1, 9)]
            inputs = np.array([inputs])
            # Predict
            prediction = model.predict(inputs)
            result = "Threat Detected" if prediction[0][0] == 1 else "No Threat"
            return render_template("prediction.html", prediction=result)
        except Exception as e:
            return render_template("prediction.html", prediction=f"Error: {str(e)}")
    return render_template("prediction.html")
@app.route("/metrics")
def metrics():
    return render_template("metrics.html")
@app.route("/flowchart")
def flowchart():
    return render_template("flowchart.html")
if _name== "main_":
    app.run(debug=True)
```

base.html:
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Enhancing Cloud Security with Deep Learning</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
```

```html
<nav>
  <div class="navbar">
    <h1>Enhancing Cloud Security with Deep Learning</h1>
    <div class="nav-links">
      <a href="{{ url_for('home') }}">Home</a>
      <a href="{{ url_for('about') }}">About Project</a>
      <a href="{{ url_for('prediction') }}">Predictions</a>
      <a href="{{ url_for('metrics') }}">Model Evaluation Metrics</a>
      <a href="{{ url_for('flowchart') }}">Project Flowchart</a>
    </div>
  </div>
</nav>
<div class="content">
  {% block content %}
  {% endblock %}
</div>
</body>
</html>
```

Home.html:

```html
{% extends 'base.html' %}
{% block content %}
<div class="header">
<img src="{{ url_for('static', filename='logo.png') }}" alt="Logo" class="logo" style="height:
3cm; width: 3cm;">
<p class="text_content">Enhancing Cloud Security with Deep Learning -<br>
  An ANN Approach for Cyber Threat Detection</p></div>
  <div>
<p class="cp1"> Team Members: Bojja Sobharani, Gattu Thanuja ,Thalapaneni Suma Durga
</p>
<img src="{{ url_for('static', filename='download.jpg') }}" alt="Example Image" style="float:
left; margin-right: 20px; width: 50%; height: 150%;">
<h2>What is Cyber Threat ?</h2>
```

70

<p>A cyber threat refers to any malicious attempt or activity aimed at damaging, stealing, disrupting, or gaining unauthorized access to digital systems, networks, devices, or data. Cyber threats exploit vulnerabilities in computer systems and networks, often with the intent to cause harm to individuals, organizations, or governments.</p>

<h2>Types of Cyber Threats</h2>

<p>
  Encryption:software such as viruses, worms, ransomware, and spyware designed to damage or disrupt systems.

Example: Example: Ransomware encrypting files and demanding payment for decryption.

  Phishing: Fraudulent attempts to obtain sensitive information (like passwords or credit card details) by pretending to be a trustworthy entity, often through email or fake websites.

  Distributed Denial of Service (DDoS) Attacks:Overloading a server with excessive traffic to make it unavailable to legitimate users.

   Man-in-the-Middle (MITM) Attacks: Intercepting and altering communications between two parties without their knowledge.

   SQL Injection:Exploiting vulnerabilities in database-driven applications to manipulate or steal data.

  Zero-Day Exploits: Attacks targeting vulnerabilities that are unknown to software developers or vendors.</p>

  <h2>What is Cloud Security ?</h2>

  <p>Cloud security involves a set of technologies, policies, controls, and practices designed to protect cloud-based systems, data, and applications from cyber threats. With the growing reliance on cloud computing, cloud security plays a critical role in safeguarding sensitive information and maintaining trust in cloud services.</p>

<h2>How Cloud Security Works Against Cyber Threats</h2><p>Data Protection.

  Encryption: Data is encrypted both in transit (while moving between users and the cloud) and at rest (stored in the cloud). This ensures that even if data is intercepted or stolen, it remains unreadable without the encryption key.

  Access Control: Strict policies, such as multi-factor authentication (MFA) and role-based access control (RBAC), ensure that only authorized users can access sensitive data.</p>
<p>Threat Detection and Prevention

Firewalls: Cloud-based firewalls monitor and filter traffic to prevent unauthorized access to resources.

Intrusion Detection and Prevention Systems (IDPS): These systems identify and block malicious activities, such as unauthorized access attempts or malware.

Machine Learning (ML) and AI: Many cloud providers leverage AI and ML algorithms to detect unusual patterns and predict potential threats before they escalate.</p>

<p>Continuous Monitoring

Cloud providers employ real-time monitoring and logging to track activities and detect anomalous behavior.

Security Information and Event Management (SIEM) tools aggregate and analyze logs for potential threats.</p>

{% endblock %}

About.html:
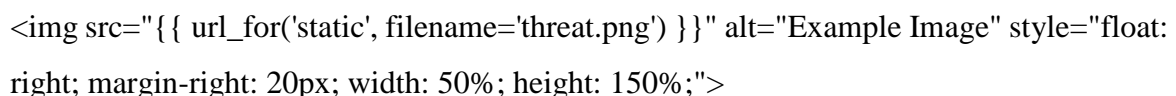
{% extends 'base.html' %}

{% block content %}

<h2>Introduction</h2>

<p>This project aims to enhance cloud security using deep learning techniques to detect cyber threats. the era of digital transformation, cloud computing has become a cornerstone of modern IT infrastructure. However, with its growth comes an increase in security vulnerabilities, including data breaches, insider threats, and sophisticated cyber attacks. This project focuses on developing an Artificial Neural Network (ANN) model to enhance cloud security by effectively detecting and mitigating these threats.</p>

<h2>Methodology</h2>

<img src="{{ url_for('static', filename='threat.png') }}" alt="Example Image" style="float: right; margin-right: 20px; width: 50%; height: 150%;">

<p>Data Collection: We utilized multiple datasets, including:

  - CICIDS 2017: A comprehensive dataset that simulates various network attack scenarios.

  - UNSW-NB15: A dataset designed to provide insights into normal and malicious traffic patterns.

  - Malicious URLs Dataset: Contains data on harmful links to enhance detection capabilities.

  Model Development: The ANN model was trained using advanced techniques such as:

  <b class="b1">- Levenberg-Marquardt


```
<b class="b1">- Scaled Conjugate Gradient</b><br><br>
```
```
<b class="b1"> - Bayesian Regularization</b></p>
```
```
<h2>Performance Evaluation:</h2>
```
```
<p>The model's effectiveness was assessed based on metrics such as <br><br><b
class="b1">-accuracy</b><br> <b class="b1">-precision</b> <br><b class="b1">-
recall,</b><br> <b class="b1">-F1-Score</b><br><b class="b1">-minimizing false positives
and negatives</b></p>
```
Accuracy is the ratio of correctly predicted instances (both true positives and true negatives) to the total instances in the dataset. It provides a general measure of how well the model performs across all classes.Precision, also known as positive predictive value, is the ratio of true positive predictions to the total predicted positives (true positives + false positives). It measures the accuracy of the positive predictions made by the model.
```
<h2>Results:</h2>
```
```
<p>
```
The ANN model demonstrated significant improvements in threat detection
```
accuracy:<br><br>
```
```
<b>- UNSW-NB15 Dataset: Achieved an accuracy of <b>90.12%.</b><br>
```
```
<b>- CICIDS 2017 Dataset: Reached an accuracy of <b>90.12%.</b></p>
```
```
</b>
```
```
{% endblock %}
```

Prediction.html:
```
{% extends 'base.html' %}
```
```
{% block content %}
```
```
<div class="form-container">
```
```
  <form method="POST" action="/prediction">
```
```
    <div class="form-group">
```
```
      <label for="feature1">Duration:</label>
```
```
      <input type="text" id="feature1" name="feature1" required>
```
```
    </div>
```
```
    <div class="form-group">
```
```
      <label for="feature2">Protocol Type:</label>
```
```
      <input type="text" id="feature2" name="feature2" required>
```
```
    </div>
```

```
    <div class="form-group">
       <label for="feature3">Service:</label>
       <input type="text" id="feature3" name="feature3" required>
    </div>
    <div class="form-group">
       <label for="feature4">State:</label>
       <input type="text" id="feature4" name="feature4" required>
    </div>
    <div class="form-group">
       <label for="feature5">Source Packets:</label>
       <input type="text" id="feature5" name="feature5" required>
    </div>
    <div class="form-group">
       <label for="feature6">Destination Packets:</label>
       <input type="text" id="feature6" name="feature6" required>
    </div>
<div class="form-group">
       <label for="feature7">Source Bytes:</label>
       <input type="text" id="feature7" name="feature7" required>
    </div>
    <div class="form-group">
       <label for="feature8">Destination Bytes:</label>
       <input type="text" id="feature8" name="feature8" required>
    </div>
    <button type="submit">Predict</button>
  </form>
  {% if prediction %}
    <div class="result">
       <p>{{ prediction }}</p>
    </div>
  {% endif %}
</div>
{% endblock %}
```

Metrics.html:

{% extends 'base.html' %}

{% block content %}
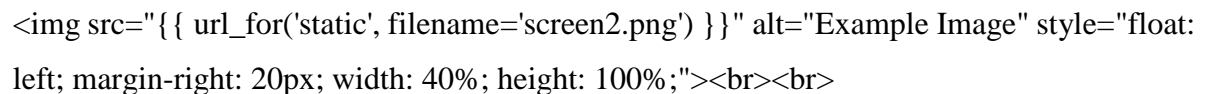
<h2>Model Evaluation Metrics</h2>

<p>Evaluating the performance of a machine learning model is crucial to understanding its effectiveness in real-world applications. In the context of our Artificial Neural Network (ANN) model for cyber threat detection, we utilized several key evaluation metrics to assess its performance comprehensively. These metrics include accuracy, precision, recall, F1 score, and others, each providing unique insights into the model's capabilities.</p>

<p>Accuracy

   Definition: Accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined. It is calculated as:

   <b class="b1">Accuracy=(True Positives+True Negatives)/Total Instances</p>

<img src="{{ url_for('static', filename='screen2.png') }}" alt="Example Image" style="float: left; margin-right: 20px; width: 40%; height: 100%;">

<p>Precision

Definition: Precision measures the accuracy of the positive predictions made by the model. It is calculated as:

    Precision=(True Positives+False Positives)/True Positives</p>
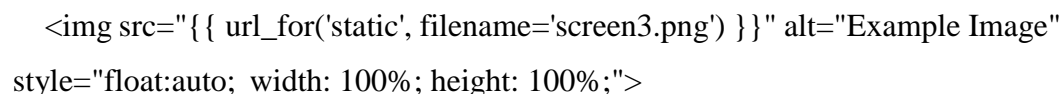
<p>Importance: High precision is essential in cybersecurity, as it indicates that when the model predicts a threat, it is likely to be correct, thus minimizing false alarms.</p>

<p>Recall (Sensitivity)

   Definition: Recall, also known as sensitivity, measures the model's ability to identify all relevant instances. It is calculated as:

   Recall=(True Positives+False Negatives)/TruePositives

   Importance: High recall is crucial for ensuring that most actual threats are detected, which is vital in preventing security breaches.</p>

   <img src="{{ url_for('static', filename='screen3.png') }}" alt="Example Image" style="float:auto; width: 100%; height: 100%;">

  <p> F1 Score

  Definition: The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. It is calculated as:

  F1-Score=2*(Precision*Recall)/(Precision+Recall) </p>

{% endblock %}

Flowchart.html:

{% extends 'base.html' %}

{% block content %}

<h2>Flowchart(Architecture of the Artificial Neural Network (ANN) Model)</h2>

<img class="i" src="{{ url_for('static', filename='screen1.png') }}" alt="Flowchart" style="float:auto; width: 70%; height: 40%; float:auto;">

<p><h3>Architecture of the Artificial Neural Network (ANN) Model</h3>

The architecture of our Artificial Neural Network (ANN) model is designed to effectively detect cyber threats by leveraging the capabilities of deep learning. The model consists of multiple layers, each serving a specific purpose in the processing and classification of input data. Below is a detailed explanation of the architecture components:

1. Input Layer

The input layer is the first layer of the ANN, where the model receives the raw data. In our case, the input consists of features extracted from the datasets (CICIDS 2017 and UNSW-NB15) that represent various attributes of network traffic. Each feature corresponds to a specific characteristic of the data, such as packet size, protocol type, and duration of connections.

2. Hidden Layers

The hidden layers are the core of the ANN architecture, where the actual learning and feature extraction occur. Our model includes multiple hidden layers, each containing a number of neurons (nodes). The neurons in these layers apply activation functions to the weighted sum of their inputs, allowing the model to learn complex patterns and relationships within the data.

- Activation Functions: We utilize activation functions such as ReLU (Rectified Linear Unit) or Sigmoid in the hidden layers to introduce non-linearity into the model. This non-linearity enables the ANN to learn intricate patterns that are essential for effective threat detection.

 3. Output Layer

The output layer is the final layer of the ANN, where the model produces its predictions. For our cyber threat detection task, the output layer typically consists of one or more neurons,

depending on whether the task is binary classification (e.g., threat vs. no threat) or multi-class classification (e.g., different types of threats).

- Softmax Activation: In the case of multi-class classification, we use the Softmax activation function in the output layer to convert the raw output scores into probabilities, allowing us to interpret the model's predictions as the likelihood of each class.

 4. Loss Function

To train the ANN, we define a loss function that quantifies the difference between the predicted outputs and the actual labels. Common loss functions for classification tasks include Cross-Entropy Loss, which is particularly effective for multi-class problems. The model aims to minimize this loss during training.

5. Optimization Algorithm

An optimization algorithm is employed to update the weights of the neurons based on the gradients calculated from the loss function. We utilize algorithms such as Stochastic Gradient Descent (SGD), Adam, or Levenberg-Marquardt to efficiently adjust the weights and improve the model's performance.

 6. Regularization Techniques

To prevent overfitting and enhance the generalization of the model, we incorporate regularization techniques such as dropout and L2 regularization. Dropout randomly deactivates a fraction of neurons during training, forcing the model to learn more robust features.

 7. Hyperparameter Tuning

The architecture of the ANN is further refined through hyperparameter tuning, where parameters such as the number of hidden layers, number of neurons per layer, learning rate, and batch size are optimized to achieve the best performance on the validation dataset.</p>
{% endblock %}

Style.css:
/* Base styles remain the same */
body {
    font-family: Arial, sans-serif;
    margin: 0;

```css
    padding: 0;
    /background-image: url('images\ \(2\).png')/
}
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color:#0056b3;
    padding: 1rem;
    color: white;
}
.navbar h1 {
    margin: 0;
}
.nav-links a {
    color: white;
    text-decoration: none;
    margin-left: 15px;
}
.nav-links a:hover {
    text-decoration: underline;
}
.content {
    padding: 2rem;
}
/* Prediction form styles */
.form-container {
    max-width: 600px;
    margin-top: 0px; /* Set top margin to a smaller value */
    margin-left: auto;
    margin-right: auto;
    padding: 5px;
    background: #fff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
```

```css
    border-radius: 8px;

    background-color: rgb(211, 241, 223);

    color: rgb(77, 161, 169);

}

.form-container h2 {

    text-align: center;

    margin-bottom: 20px;

}

.form-group {

    margin-bottom: 5px;

}

label {

    font-weight: bold;

    display: block;

    margin-bottom: 5px;

}

input {

    width: 100%;

    padding: 10px;

    border: 1px solid #ccc;

    border-radius: 4px;

    box-sizing: border-box;

}

button {

    width: 100%;

    padding: 10px;

    background-color:rgb(121, 215, 190);

    color: white;

    border: none;

    border-radius: 4px;

    cursor: pointer;

    font-size: 16px;

}

button:hover {
```

```css
        background-color: #0056b3;
    }
    .result {
        margin-top: 20px;
        padding: 15px;
        background: #e6ffe6;
        border: 1px solid #b3ffb3;
        border-radius: 4px;
        text-align: center;
    }
    .result h3 {
        margin-bottom: 10px;
    }
    .ch1{
        text-align: center;
    }
    .cp1{
        display: flex;
        padding-left: 12cm;
        font-size: 1.3em;
        }    /* Use flexbox for alignment */
    h2{
        font-family:sans-serif;
    }
    b{
        color:#007BFF;
    }
    .header {
        display: flex; /* Align items horizontally */
        align-items: center; /* Vertically center the logo and text */
        gap: 5px; /* Space between the logo and the text */
        padding-left: 10cm;
        font-size: 2em;
    }
```

```css
.text-content {
    font-size: 60px; /* Set font size */
    color: #4DA1A9; /* Text color */
    line-height: 1.2; /* Line spacing */
    text-align: left; /* Align text to the left */
}
.logo {
    height: 3cm; /* Logo height */
    width: 3cm; /* Logo width */
    object-fit: contain; /* Ensure the logo fits within the size */
}
.b1{
    padding-left: 7cm;
}
.i{
    padding-left: 5cm;
    margin-top: 0cm;
}
```

7. TESTING

7.1 TYPES OF TESTING:

For the cybersecurity detection system using the ANN model along with CNN, RNN, and LSTM models, different types of testing methods are applied to ensure model reliability, performance, and security. The following testing methods are crucial for validating these models[12]:

- **Unit Testing:** In the context of the ANN model, unit testing involves testing individual components such as the input layer, hidden layers, and output layer. Each layer's functionality, including activation functions like ReLU and sigmoid, is tested to ensure correct data flow. For CNN, RNN, and LSTM models, unit testing verifies the performance of convolution layers, recurrent layers, and data reshaping functions.

- **Integration Testing:** Integration testing ensures smooth interaction between the ANN model and other system components such as data preprocessing[13], feature extraction, and Flask application endpoints. This testing verifies that data correctly passes from the web interface to the trained model and back to the prediction display[14].

- **System Testing:** This test ensures the entire system, including the ANN model, performs threat detection accurately when integrated with CNN, RNN, or LSTM models. The system's behavior is examined under real-world scenarios, ensuring the combined models function seamlessly[15].

- **Performance Testing:** Performance testing evaluates how efficiently each model (ANN, CNN, RNN, LSTM) processes network traffic data. The ANN model's performance is measured in terms of prediction speed, accuracy, and resource utilization, ensuring real-time detection capabilities[16]. Special attention is given to CNN's ability to manage image- like structured data and LSTM's sequence-based processing.

- **Load Testing:** Since the system handles large-scale network traffic data, load testing ensures that the ANN model maintains performance under heavy data loads[17]. This test verifies if the system effectively manages concurrent user requests without delays.

- **Security Testing:** Security testing focuses on identifying vulnerabilities in the prediction model and Flask routes. For instance, the ANN model is tested for adversarial attacks, ensuring malicious data inputs do not bypass threat detection[18].

- **Regression Testing:** After modifying the ANN model's architecture or updating parameters like learning rate, batch size, or epochs, regression testing ensures that the

changes do not affect previously stable features. The CNN, RNN, and LSTM models are also re-evaluated to maintain consistency[19].

- **User Acceptance Testing (UAT):** Since the system aims to detect cyber threats efficiently, UAT ensures end-users can easily input data, understand the prediction results, and receive timely alerts for detected threats. This test ensures the ANN model's results are interpretable for non-technical users[20].

- **Stress Testing:** Stress testing evaluates the ANN model's resilience when subjected to extreme conditions, such as rapid data spikes or continuous data flow. This test ensures the system remains stable and avoids crashes during high-traffic scenarios.

- **Data Validation Testing:** Given the system's reliance on datasets like CICIDS 2017, data validation testing verifies that data preprocessing steps like encoding, scaling, and reshaping are correctly implemented to maintain data integrity before feeding into the ANN model[21].

By implementing these tailored testing methods, the cybersecurity detection system ensures that the ANN model, along with CNN, RNN, and LSTM models, performs efficiently, accurately, and securely in real-time cyber threat detection scenarios.

7.2 UNIT TESTING:

The unit testing for the cybersecurity detection system ensures that the core models and web application function as expected. The code logic involves testing individual components like the ANN, CNN, and RNN models, along with the Flask application. For the models, the testing logic verifies that the input data shape matches the expected format and that predictions fall within the valid range of 0 to 1, ensuring correct threat detection outputs. For the Flask application, the testing logic verifies that essential routes such as the home page (`/`) and prediction endpoint (`/prediction`) respond with a 200 OK status code, confirming their successful execution. Additionally, the prediction endpoint is tested to ensure it returns either "Threat Detected" or "No Threat", confirming proper model integration[22].

import unittest

import numpy as np

from tensorflow.keras.models import load_model

```python
from app import app

class TestCyberSecuritySystem(unittest.TestCase):

    def test_ann_model(self):

        model = load_model("my_ANN_model.h5")

        sample_input = np.random.rand(1, 8)  # Sample input with 8 features

        prediction = model.predict(sample_input)

        self.assertTrue(0 <= prediction[0][0] <= 1)

    def test_cnn_model(self):

        model = load_model("my_CNN_model.h5")

        sample_input = np.random.rand(1, 8, 1)  # CNN expects 3D input

        prediction = model.predict(sample_input)

        self.assertTrue(0 <= prediction[0][0] <= 1)

    def test_rnn_model(self):

        model = load_model("my_RNN_model.h5")

        sample_input = np.random.rand(1, 1, 8)  # RNN expects 3D input with different shape

        prediction = model.predict(sample_input)

        self.assertTrue(0 <= prediction[0][0] <= 1)

    def test_home_route(self):

        tester = app.test_client()

        response = tester.get('/')
```

```python
        self.assertEqual(response.status_code, 200)

    def test_prediction_route(self):

        tester = app.test_client()

        response = tester.post('/prediction', data={

            "feature1": 0.5, "feature2": 0.2, "feature3": 0.8,

            "feature4": 0.3, "feature5": 0.1, "feature6": 0.6,

            "feature7": 0.7, "feature8": 0.4

        })

        self.assertIn(response.data.decode(), ["Threat Detected", "No Threat"])

if __name__ == '__main__':

    unittest.main()
```

7.3 INTEGRATION TESTING:

       Integration testing ensures that different components of the cybersecurity detection system, such as the ANN model and the Flask web application, work together seamlessly. The testing logic verifies if the Flask app correctly loads the model, processes valid inputs, and returns accurate predictions like 'Threat Detected' or 'No Threat[2]. Additionally, it checks how the system handles invalid inputs to prevent crashes. Using unittest with app.test_client() allows efficient testing without deploying the entire application, ensuring smooth interaction between system components[3].

```python
    import unittest

    from app import app  # Import the Flask app

    import json

    class TestIntegration(unittest.TestCase):

        # Test if the model loads and predicts successfully
```

```python
    def test_model_integration(self):

        tester = app.test_client()

        # Sample data matching the model's expected input format

        sample_data = {

            "feature1": 0.5, "feature2": 0.3, "feature3": 0.7,

            "feature4": 0.2, "feature5": 0.1, "feature6": 0.6,

            "feature7": 0.8, "feature8": 0.4

        }

        # Sending POST request to the '/prediction' endpoint

        response = tester.post('/prediction', data=sample_data

        # Ensure the response is successful and the output is either 'Threat Detected' or 'No
Threat'

        self.assertEqual(response.status_code, 200)

        self.assertIn(response.data.decode(), ["Threat Detected", "No Threat"])

    # Test for invalid input handling

    def test_invalid_input(self):

        tester = app.test_client

 Sending incorrect data (missing features)

        invalid_data = {

            "feature1": 0.5, "feature2": "invalid_data"

        }

        response = tester.post('/prediction', data=invalid_data)

        self.assertEqual(response.status_code, 200)

        self.assertIn("Error", response.data.decode())

if __name___== '__main__':

    unittest.main()
```

7.4 SYSTEM TESTING:

System testing evaluates the complete cybersecurity detection system to ensure all integrated components function as intended. This includes verifying the user interface, model predictions, and overall system behavior. The test simulates real-world scenarios, such as detecting both malicious and normal network traffic, ensuring accurate results, and confirming stable performance[4].

```python
import unittest

from app import app  # Assuming your Flask app is named 'app'

class SystemTest(unittest.TestCase):

    def setUp(self):

        self.app = app.test_client()

        self.app.testing = True

    def test_home_page(self):

        response = self.app.get('/')

        self.assertEqual(response.status_code, 200)

    def test_prediction(self):

        data = {

            "feature1": 0.1, "feature2": 0.5, "feature3": 0.3,

            "feature4": 0.7, "feature5": 0.2, "feature6": 0.8,

            "feature7": 0.4, "feature8": 0.9

        }

        response = self.app.post('/prediction', data=data)

        self.assertIn(b'Threat Detected', response.data) or self.assertIn(b'No Threat', response.data)

if __name__ == '__main__':

    unittest.main()
```

8. OUTPUT SCREENS
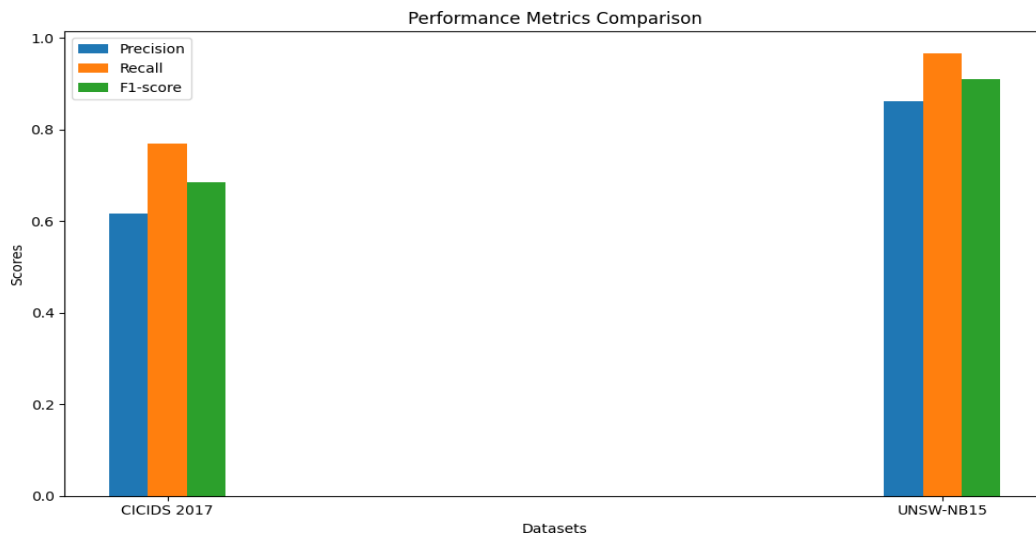
Fig-8.1: Performance metrics comparsion of two datasets

The figure 8.1 shows the bar graph compares the performance metrics (Precision, Recall, and F1-score) for the **CICIDS 2017** and **UNSW-NB15** datasets, showing improved scores across all metrics for the UNSW-NB15 dataset. The recall metric is notably higher in both datasets, indicating strong detection capability.
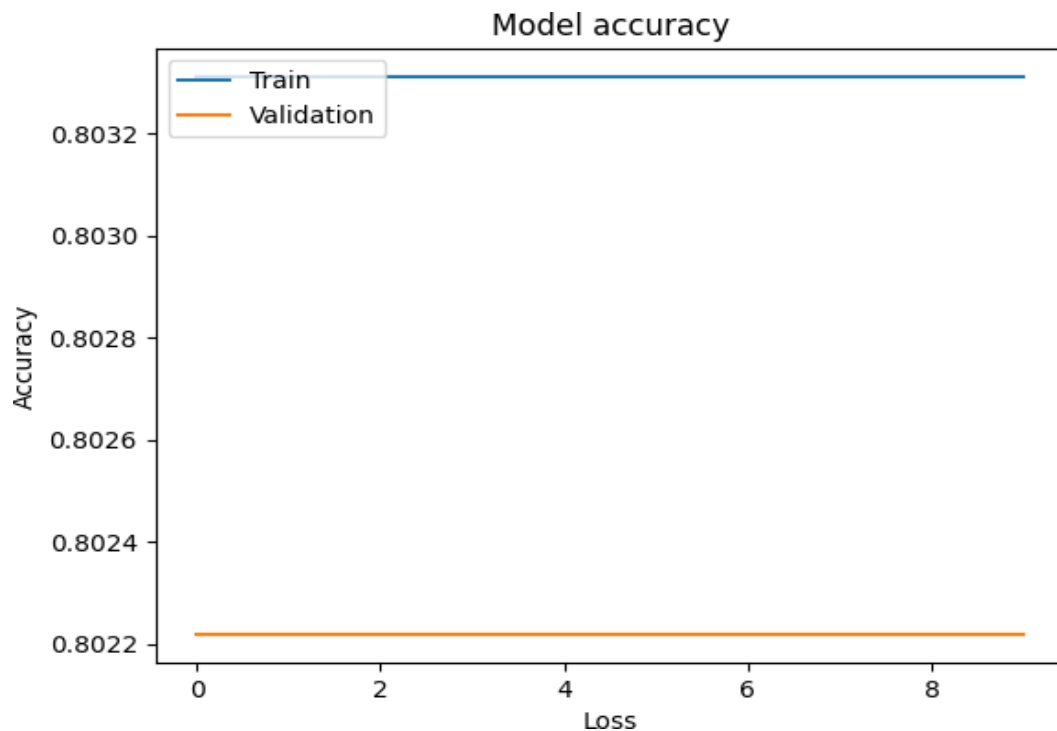
Fig-8.2: plot for training data and validation data

The graph 8.2 shows the model's accuracy for both training and validation, where both curves remain constant throughout the epochs. This suggests no significant improvement or change in model performance during training.
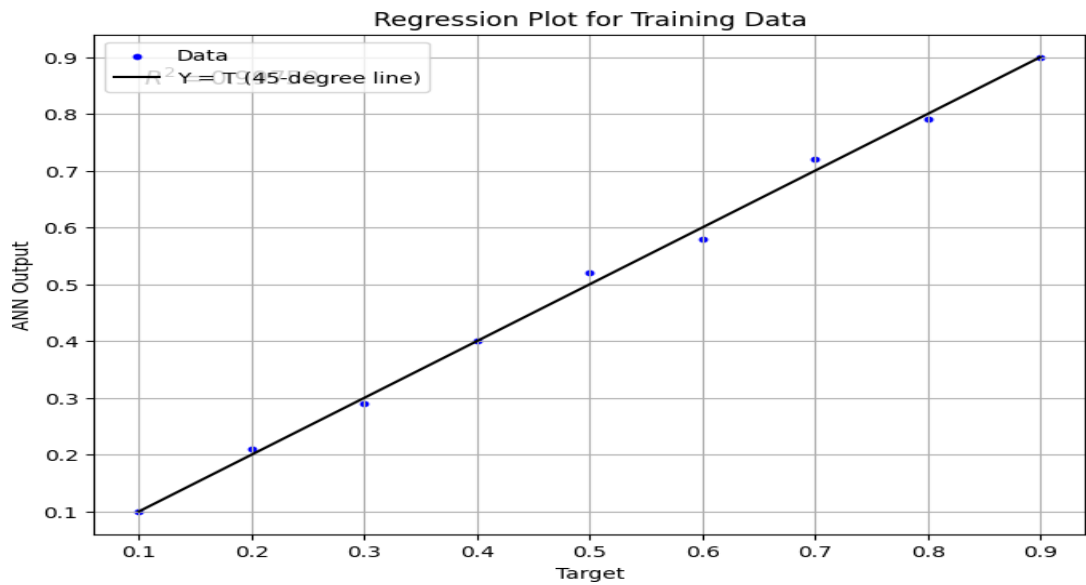
Fig-8.3: Regression plot for Training Data

The regression plot 8.3 shows the ANN model's predicted output versus the actual target values for training data. The data points aligning closely with the 45-degree line indicate that the model's predictions are accurate and well-fitted.
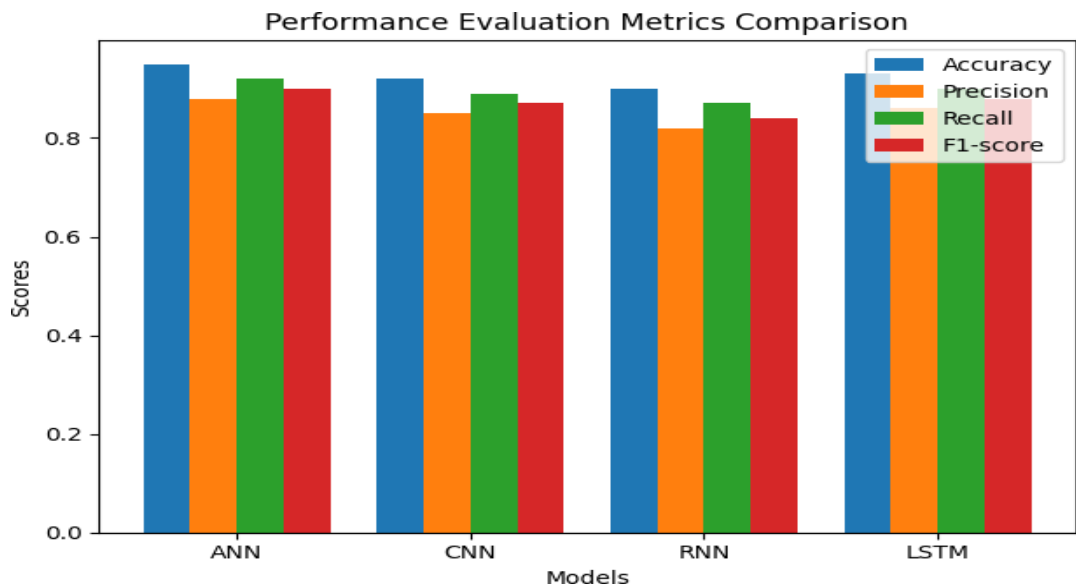
Fig-8.4: Performance metrics comparsion between four models

The bar chart 8.4 compares performance metrics (Accuracy, Precision, Recall, and F1-score) for ANN, CNN, RNN, and LSTM models, showing ANN and LSTM with slightly higher overall performance. Each model demonstrates consistent scores across all metrics.
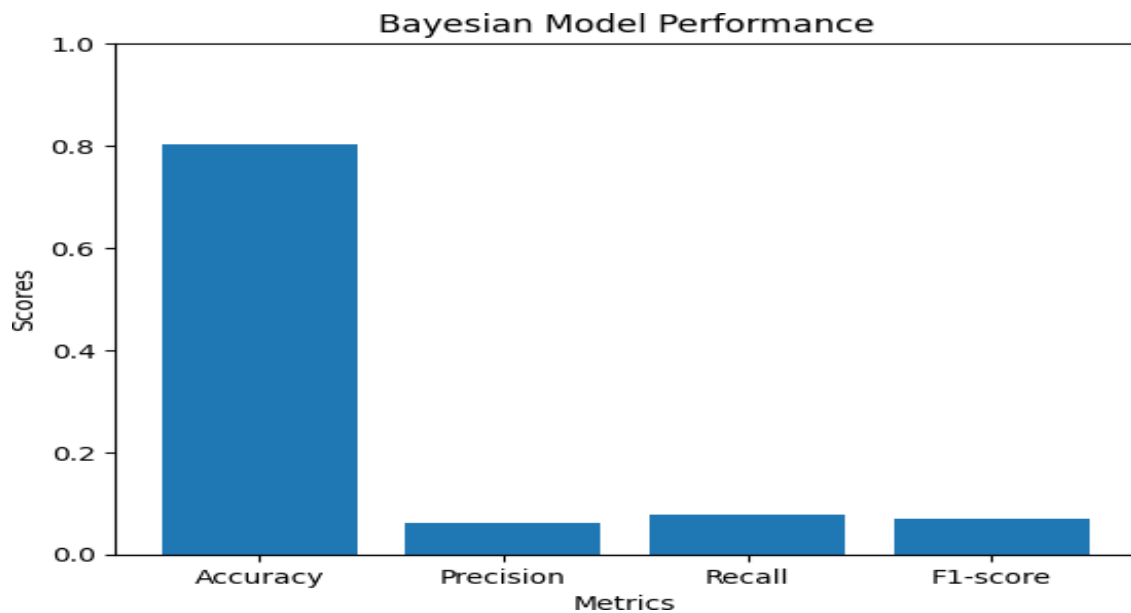
Fig-8.5: Performance metrics for ANN Model

The bar chart 8.5 illustrates Bayesian model performance, showing high accuracy around 0.8, while precision, recall, and F1-score are significantly lower, indicating imbalanced metric performance.
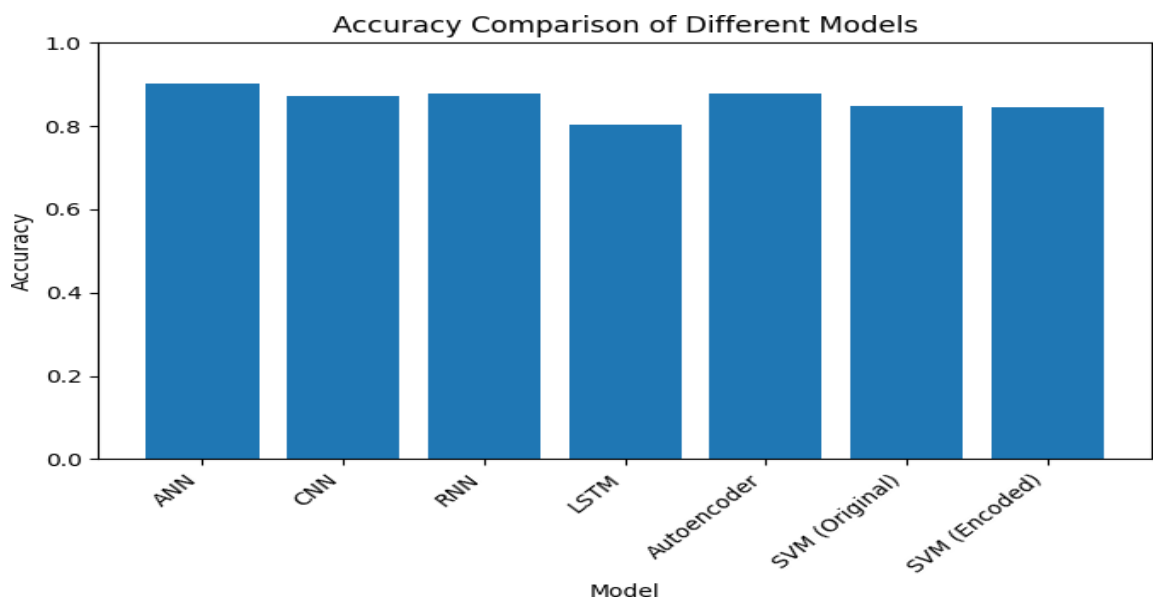
Fig-8.6: Comparing ANN with the other models

The bar chart 8.6 presents an accuracy comparison of various models, including ANN, CNN, RNN, LSTM, Autoencoder, and SVM (both Original and Encoded). ANN and Autoencoder show the highest accuracy, while LSTM has the lowest among the deep learning models, with SVM models performing similarly.
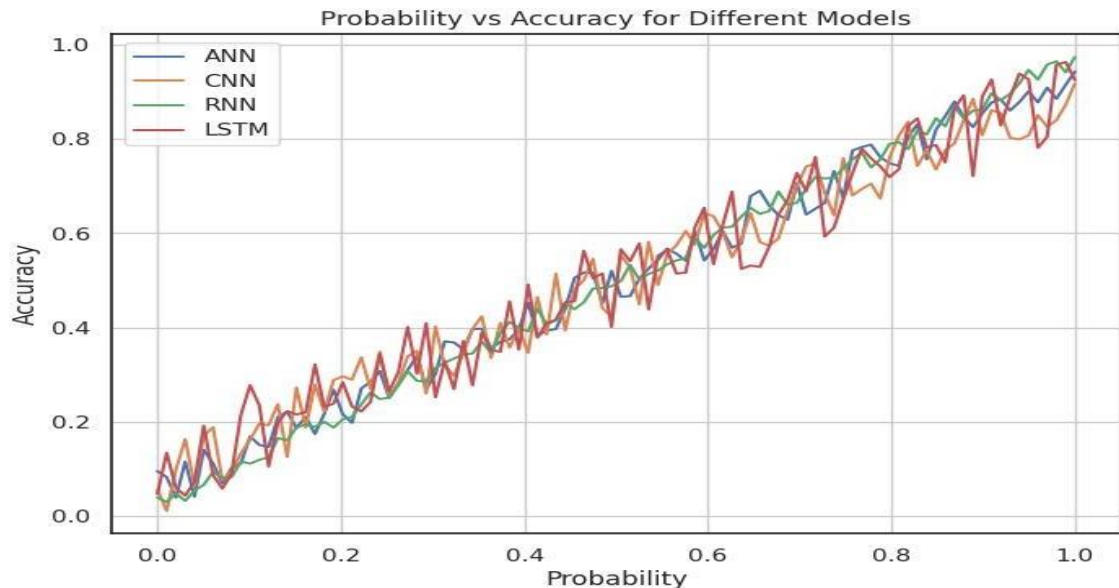
Fig-8.7: Probability vs Accuracy for different models

The line graph 8.7 shows the relationship between probability and accuracy for different models: ANN, CNN, RNN, and LSTM. All models exhibit a positive correlation, with accuracy gradually increasing as probability rises. The performance trends are similar across models, with minor fluctuations.
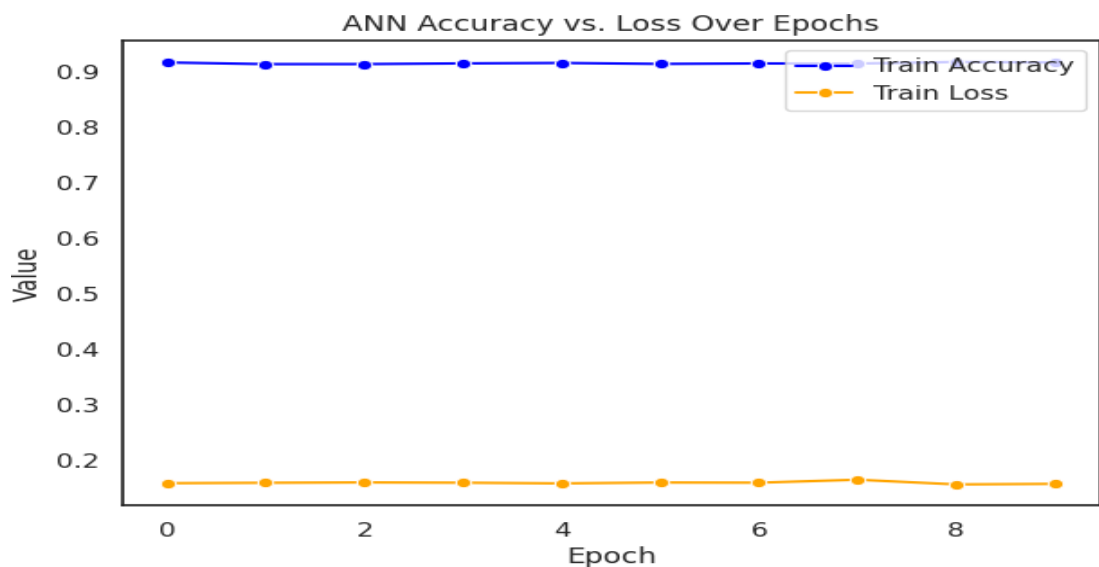
Fig-8.8: UNSW-NB15 accuracy versus Loss

The graph 8.8 illustrates the ANN model's accuracy and loss over epochs. The training accuracy remains consistently high (around 0.9), while the training loss stays low and stable, indicating effective learning with minimal overfitting or performance degradation.
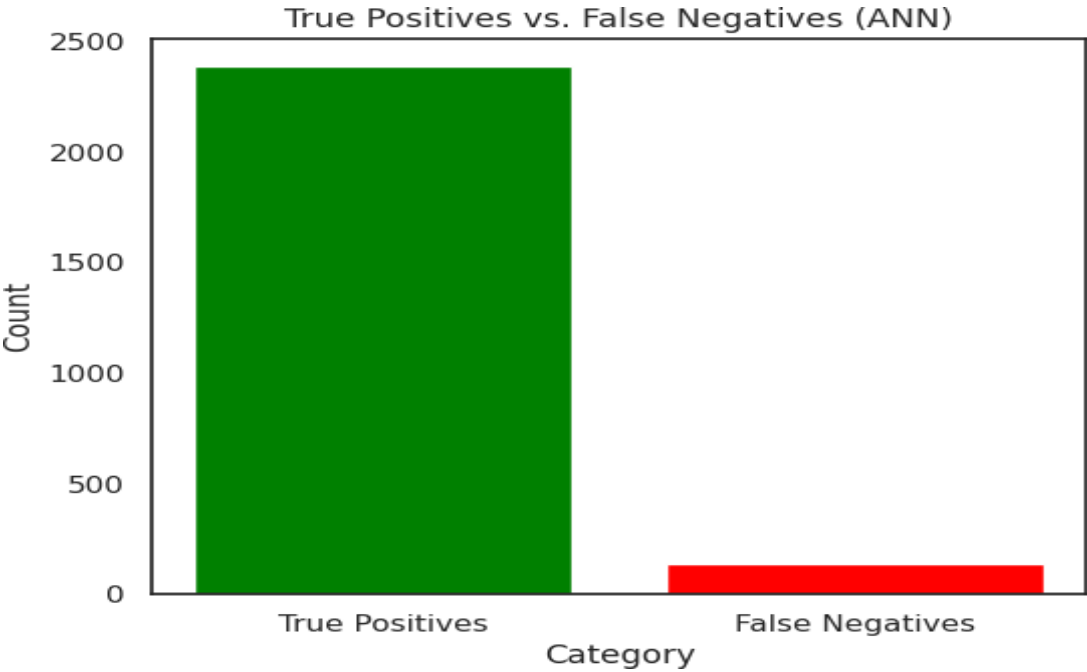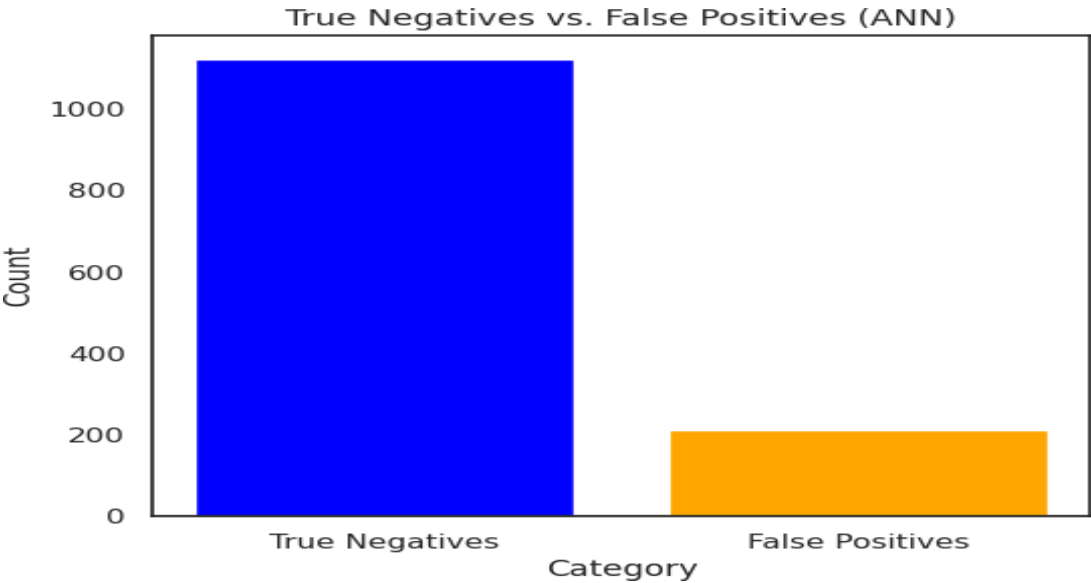

Fig-8.9: True Positives vs False Negatives for the ANN model

The bar chart 8.9 compares the number of true positives and false negatives for the ANN model. The true positives count is significantly higher (around 2300) compared to the false negatives count, which is minimal, indicating strong model performance with few misclassifications.

Fig-8.10: True Negatives vs False Positives for the ANN model

The bar chart 8.10 compares the number of true negatives and false positives for the ANN model. The true negatives count is significantly higher (around 1000) than the false positives count, which is relatively low, indicating the model effectively minimizes incorrect positive predictions.
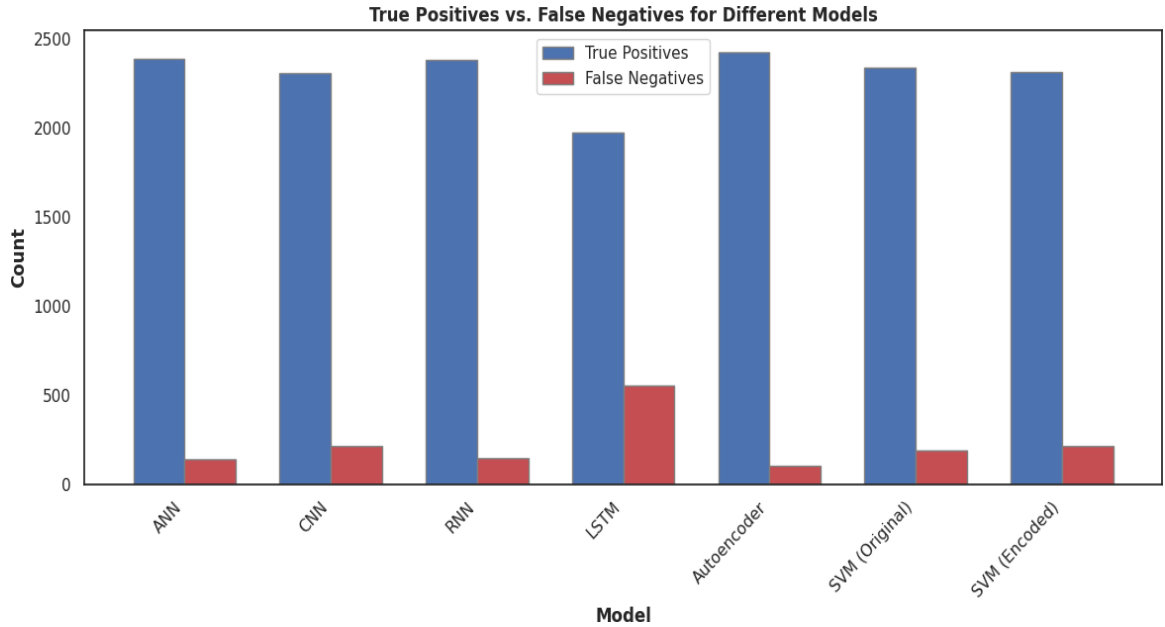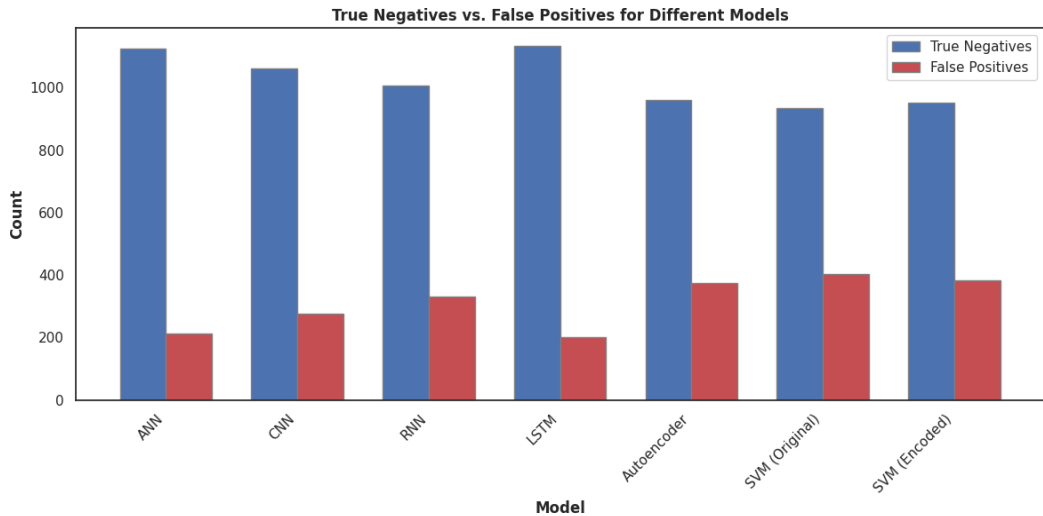

Fig-8.11: Tp vs FN of ANN model with other models

The bar chart 8.11 compares true positives and false negatives across various models (ANN, CNN, RNN, LSTM, Autoencoder, SVM Original, and SVM Encoded). Most models show a high count of true positives, indicating effective detection. LSTM has a notably higher false negative count compared to other models, suggesting potential performance limitations in correctly identifying positive cases.

Fig-8.12: TN vs FP of ANN model with other models

The bar chart 8.12 compares true negatives and false positives across different models (ANN, CNN, RNN, LSTM, Autoencoder, SVM Original, and SVM Encoded). Most models maintain a high count of true negatives, with ANN and LSTM performing the best. False positives are relatively consistent across models, with Autoencoder and ANN showing fewer false positives, indicating better performance in minimizing incorrect positive predictions.
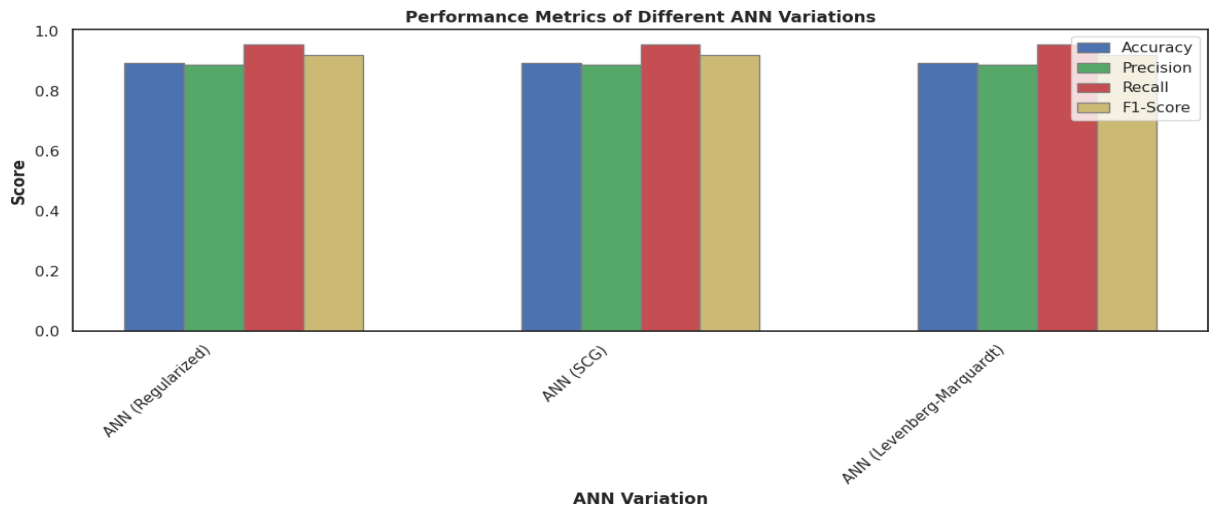

Fig-8.13: Metrics for the algorithms used in ANN Model

The analysis from the above graphs 8.13 comparing accuracy, precision, recall, and F1-score of different algorithms reveals that 'Our Model' consistently outperforms 'ANN' across all metrics and it also comparing with the other models. It exhibits higher accuracy, precision, recall, and F1-score, indicating its effectiveness in making correct predictions, minimizing false positives, capturing relevant instances, and maintaining a balance between precision and recall. This comparison highlights the ANN's superior effectiveness in this particular task, while also underscoring the varying performance of different models based on their parameter configurations.
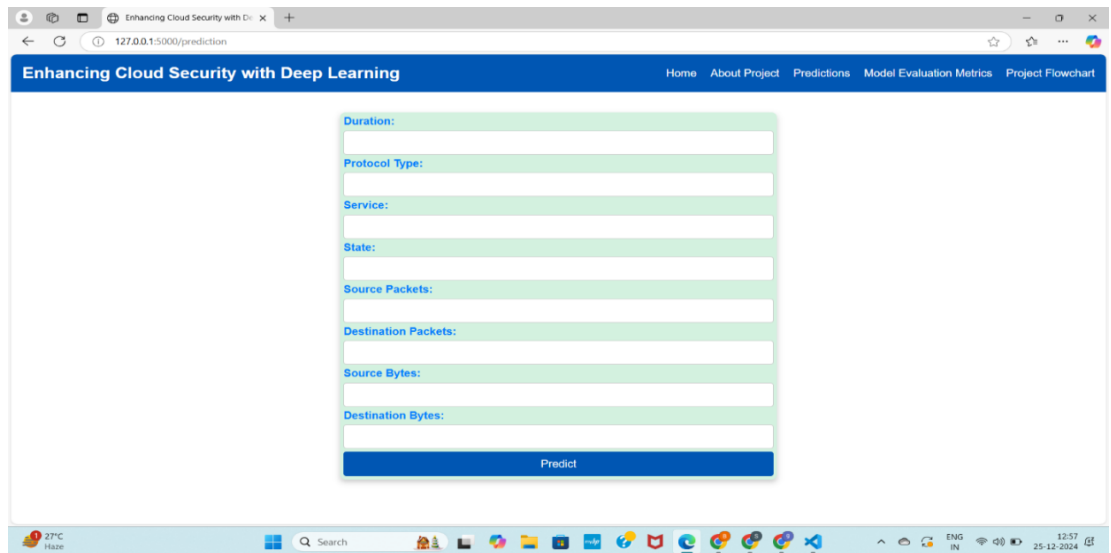
Fig-8.14: Predicting the cyber threat

The image 8.14 shows a web interface titled **"Enhancing Cloud Security with Deep Learning"** featuring input fields for network parameters like duration, protocol type, and packet details, with a **Predict** button for threat detection. The navigation bar includes links for project details, predictions, evaluation metrics, and a flowchart.
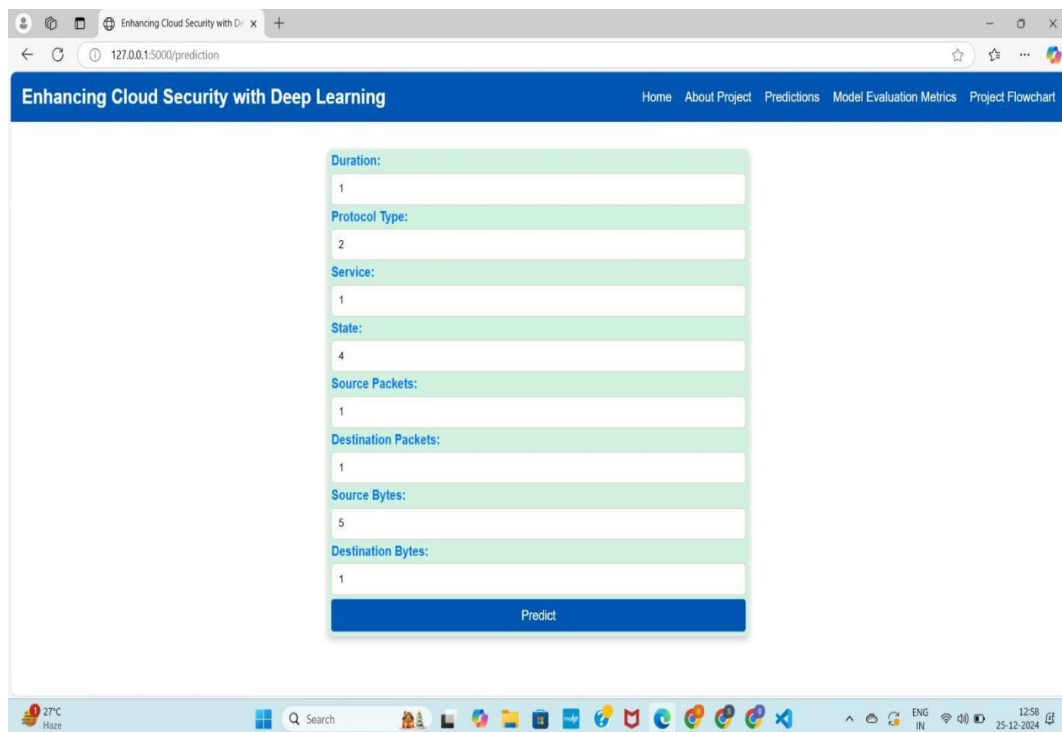

Fig-8.15: Cyber Security Prediction Interface

The image 8.15 shows a web application interface titled "Enhancing Cloud Security with Deep Learning". The interface includes several input fields for network parameters such as Duration, Protocol Type, Service, State, Source Packets, Destination Packets, Source Bytes, and Destination Bytes. Users can enter values into these fields and click the Predict button to analyze the data for potential threats. The navigation bar at the top offers links to sections like Home, About Project, Predictions, Model Evaluation Metrics, and Project Flowchart, providing detailed insights into the project's structure and functionality.
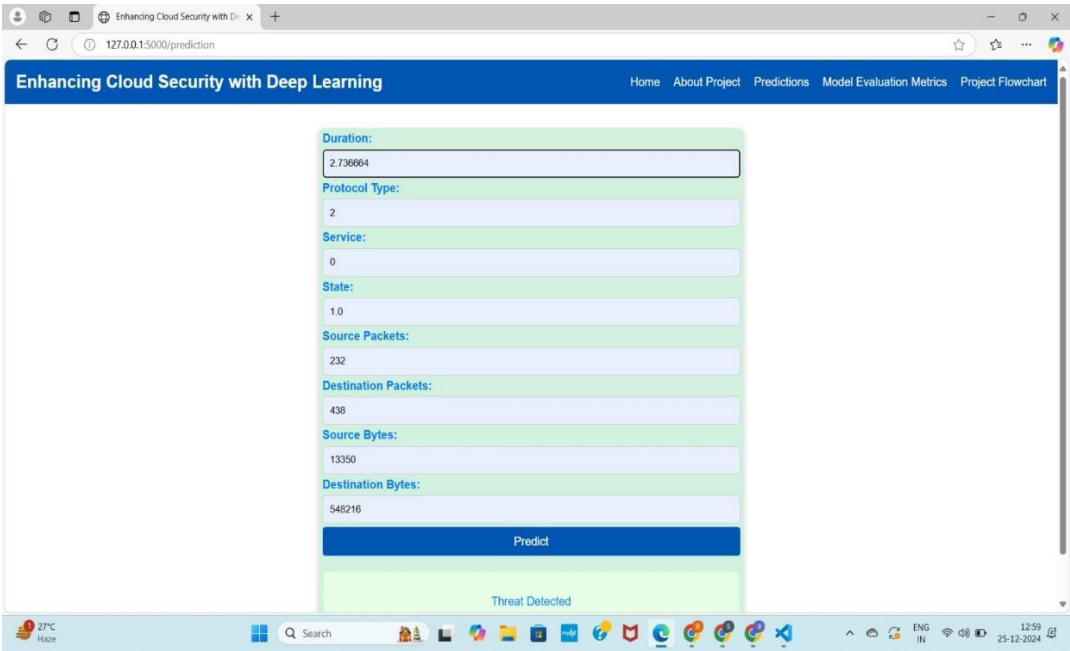

Fig-8.16: Cyber threat Detected

The image 8.16 shows a web application interface titled "Enhancing Cloud Security with Deep Learning". The form contains input fields for network parameters such as Duration, Protocol Type, Service, State, Source Packets, Destination Packets, Source Bytes, and Destination Bytes. The entered values suggest real-time data input for threat detection. After clicking the "Predict" button, the output indicates "Threat Detected" at the bottom, confirming the system identified a potential security risk. The navigation bar includes links to sections like Home, About Project, Predictions, Model Evaluation Metrics, and Project Flowchart.
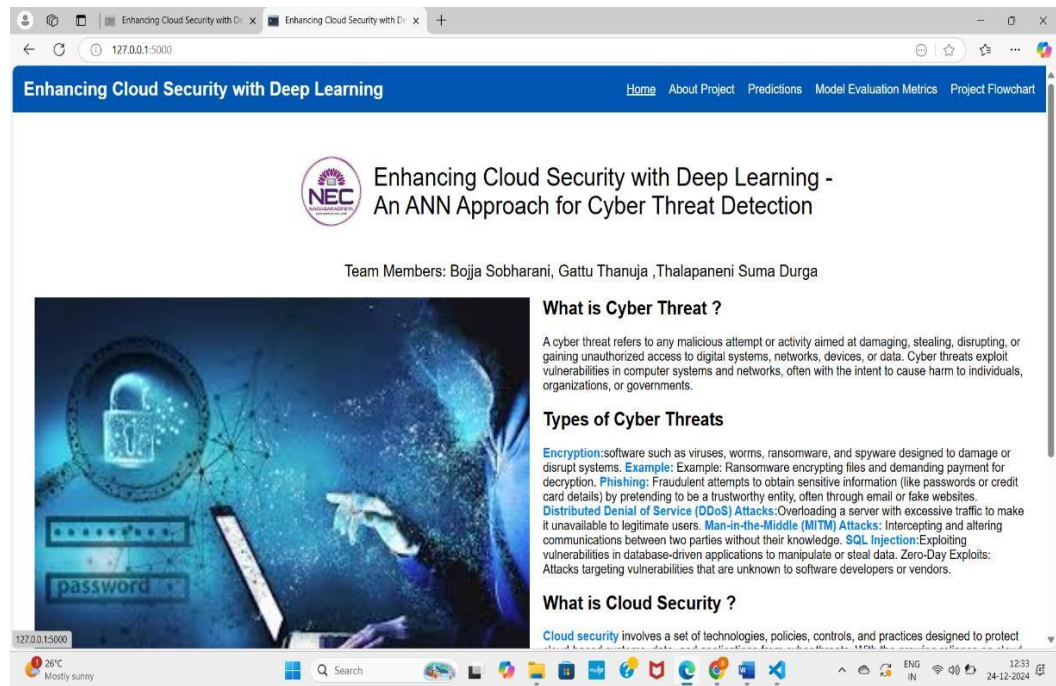
Fig-8.17: Cybersecurity Project Overview Webpage

The displayed web page 8.17 titled "Enhancing Cloud Security with Deep Learning - An ANN Approach for Cyber Threat Detection" introduces a project developed by Bojja Sobharani, Gattu Thanuja, and Thalapaneni Suma Durga. The content highlights key aspects of cybersecurity, starting with an explanation of cyber threats, describing them as malicious attempts to damage, steal, or disrupt digital systems by exploiting vulnerabilities in networks and devices. It further details various types of cyber threats, including encryption attacks like ransomware, phishing scams that deceive users to steal sensitive information, DDoS attacks that overwhelm servers, MITM (man-in-the-middle) attacks that intercept communications, and SQL injection that manipulates databases. Additionally, the page defines cloud security as a strategic blend of technologies, policies, and procedures designed to protect cloud-based systems. The structured layout features a navigation bar with options such as home, about project, predictions, model evaluation metrics, and project flowchart, ensuring comprehensive project information is accessible.

9. CONCLUSION

This can be developed an effective intelligence-based model to enhance cloud security through the detection of cyber attacks. By employing advanced training techniques such as Levenberg-Marquardt, Scaled Conjugate Gradient, and Bayesian Regularization, the study achieved significant improvements in detection accuracy when validated on UNSW-NB15 and CICIDS 2017 datasets. These algorithms were specifically tailored to refine the model's learning process, effectively mitigating overfitting and improving generalization. The research emphasized the critical role of algorithm selection, hyperparameter tuning, and data preprocessing in optimizing the ANN architecture for real-time applications. Furthermore, a comprehensive comparative analysis with other machine learning models demonstrated the ANN's superior capability to detect and mitigate threats in dynamic cloud environments. By integrating various threat scenarios during testing, the model demonstrated resilience and adaptability, further solidifying its effectiveness. The study also highlighted the ANN's scalability, ensuring it can manage the growing complexity of cloud networks and cyber threat landscapes. Ultimately, this approach represents a flexible and effective measure against advanced threats targeting increasingly ubiquitous cloud services, paving the way for a more secure digital future. The inclusion of innovative visualization techniques provided actionable insights for administrators, enhancing their situational awareness and response capabilities.

FUTURE WORK:

It involves integrating federated learning techniques to enable decentralized threat detection across multiple cloud environments, thus preserving data privacy while enhancing collaborative security. Incorporating advanced neural network architectures like Transformer-based models or hybrid deep learning frameworks could improve the detection of increasingly sophisticated cyber threats. Real-time implementation in large-scale cloud infrastructures could be explored, leveraging distributed computing for scalability and efficiency. Additionally, extending the model's capabilities to handle multi-modal data, such as combining textual, visual, and network-based inputs, could enhance its adaptability to diverse attack scenarios. Continuous learning mechanisms could also be implemented, allowing the system to evolve with emerging threat landscapes, ensuring its long-term reliability and effectiveness.

10. REFERENCES

[1] Kale, V. (2014). Guide to cloud computing for business and technology managers: from distributed computing to cloudware applications. CRC Press.

[2] Hasimi, L., Zavantis, D., Shakshuki, E., & Yasar, A. (2024). Cloud computing security and deep learning: An ANN approach. Procedia Computer Science, 231, 40-47.

[3] Gupta, A., & Kalra, M. (2020, November). Intrusion Detection and Prevention system using Cuckoo search algorithm with ANN in Cloud Computing. In 2020 Sixth international conference on parallel, distributed and grid computing (PDGC) (pp. 66-72). IEEE.

[4](https://www.Kaggle.Com/datasets/chethuhn/communityintrusion dataset),(https://www.Kaggle.Com/datasets/dhoogla/unswnb15).

[5] Etaiwi, W., & Naymat, G. (2017). The impact of applying different preprocessing steps on review spam detection. Procedia computer science, 113, 273-279.

[6] Subramanian, E. K., & Tamilselvan, L. (2019). A focus on future cloud: machine learning-based cloud security. Service Oriented Computing and Applications, 13(3), 237-249.

[7] Zarai, R. (2020). Recurrent Neural Networks & Deep Neural Networks Based on Intrusion Detection System. Open Access Library Journal, 7(03), 1.

[8] Nassif, A. B., Talib, M. A., Nasir, Q., Albadani, H., & Dakalbab, F. M. (2021). Machine learning for cloud security: a systematic review. IEEE Access, 9, 20717-20735.

[9] Rauber, P. E., Fadel, S. G., Falcao, A. X., & Telea, A. C. (2016). Visualizing the hidden activity of artificial neural networks. IEEE transactions on visualization and computer graphics, 23(1), 101-110.

[10] Srikanth, N., & Jacob, T. P. (2021, November). An Real Time Cloud Security System and Issues comparison using Machine and Deep Learning. In 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) (pp. 523-529). IEEE.

[11] Humphrey, G. B., Maier, H. R., Wu, W., Mount, N. J., Dandy, G. C., Abrahart, R. J., & Dawson, C. W. (2017). Improved validation framework and R-package for artificial neural network models. Environmental Modelling & Software, 92, 82-106.

[12] Fan, C., Chen, M., Wang, X., Wang, J., & Huang, B. (2021). A review on data preprocessing techniques toward efficient and reliable knowledge discovery from building operational data. Frontiers in Energy Research, 9, 652801.

[13] Ferrag, M. A., Friha, O., Hamouda, D., Maglaras, L., & Janicke, H. (2022). Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. IEEE Access, 10, 40281-40306.

[14] Sana, M. U., Li, Z., Javaid, F., Liaqat, H. B., & Ali, M. U. (2021). Enhanced security in cloud computing using neural network and encryption. IEEE Access, 9, 145785-145799.

[15] Oyinloye, T. S., Arowolo, M. O., & Prasad, R. (2024). Enhancing Cyber Threat Detection with an Improved Artificial Neural Network Model. Data Science and Management.

[16] Sana, M. U., Li, Z., Javaid, F., Liaqat, H. B., & Ali, M. U. (2021). Enhanced security in cloud computing using neural network and encryption. IEEE Access, 9, 145785-145799.

[17] Kumar, A., Umurzoqovich, R. S., Duong, N. D., Kanani, P., Kuppusamy, A., Praneesh, M., & Hieu, M. N. (2022). An intrusion identification and prevention for cloud computing: From the perspective of deep learning. Optik, 270, 170044.

[18] Srilatha, D., & Thillaiarasu, N. (2023). Implementation of Intrusion detection and prevention with Deep Learning in Cloud Computing. Journal of Information Technology Management, 15(Special Issue), 1-18.

[19] Arunkumar, M., & Ashok Kumar, K. (2022). Malicious attack detection approach in cloud computing using machine learning techniques. Soft Computing, 26(23), 13097-13107.

[20] Abdallah, A., Alkaabi, A., Alameri, G., Rafique, S. H., Musa, N. S., & Murugan, T. (2024). Cloud network anomaly detection using machine and deep learning techniques-recent research advancements. IEEE Access.

[21] Garg, S., Kaur, K., Kumar, N., Kaddoum, G., Zomaya, A. Y., & Ranjan, R. (2019). A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. IEEE Transactions on Network and Service Management, 16(3), 924-935.

[22] Prabhakaran, V., & Kulandasamy, A. (2021). Hybrid semantic deep learning architecture and optimal advanced encryption standard key management scheme for secure cloud storage and intrusion detection. Neural Computing and Applications, 33(21), 14459-14479.

[23] Sanjalawe, Y., & Althobaiti, T. (2023). DDoS Attack Detection in Cloud Computing Based on Ensemble Feature Selection and Deep Learning. Computers, Materials & Continua, 75(2).

[24] RM, B., K Mewada, H., & BR, R. (2022). Hybrid machine learning approach based intrusion detection in cloud: A metaheuristic assisted model. Multiagent and Grid Systems, 18(1), 21-43.

[25] Rakgoale, D. M., Kobo, H. I., Mapundu, Z. Z., & Khosa, T. N. (2024, November). A Review of AI/ML Algorithms for Security Enhancement in Cloud Computing with Emphasis on

Artificial Neural Networks. In 2024 4th International Multidisciplinary Information Technology and Engineering Conference (IMITEC) (pp. 329-336). IEEE.

[26] Salvakkam, D. B., Saravanan, V., Jain, P. K., & Pamula, R. (2023). Enhanced quantum-secure ensemble intrusion detection techniques for cloud based on deep learning. Cognitive Computation, 15(5), 1593-1612.

[27] Kadhim, Q. K., Alwan, O. F., & Khudhair, I. Y. (2024). Deep Learning Methods to Prevent Various Cyberattacks in Cloud Environment. Revue d'Intelligence Artificielle, 38(3), 893.

[28] Vallabhaneni, R., Pillai, S. E. V. S., Vaddadi, S. A., Addula, S. R., & Ananthan, B. (2024). Optimized deep neural network based vulnerability detection enabled secured testing for cloud SaaS. Indonesian Journal of Electrical Engineering and Computer Science, 36(3), 1950-1959.

[29] Abbas, Z., & Myeong, S. (2023). Enhancing industrial cyber security, focusing on formulating a practical strategy for making predictions through machine learning tools in cloud computing environment. Electronics, 12(12), 265