**Network Intrusion Detection System Using Machine Learning and Deep Learning for IoT Networks**

A Project Presented to

The Faculty of the Department of

Aviation and Technology

San Jose State University

**In Partial Fulfilment**

**of the Requirements for the degree**

**Master of Science**

**In**

**Engineering Technology**

**By**

**Shailja Roy**

**May 2025**

Copyright © 2025

Shailja Roy

ALL RIGHTS RESERVED

**SAN JOSE STATE UNIVERSITY**

**The Undersigned Committee Approves**

Network Intrusion Detection System Using Machine Learning and Deep Learning for IoT Networks

Of

Shailja Roy

APPROVED FOR THE DEPARTMENT OF AVIATION AND TECHNOLOGY

Dr Armin Moghadam, Committee Chair Date

Dr Riti Gour, Committee Member Date

Prof Bill Saichek, Committee Member Date

**ABSTRACT**

Network Intrusion Detection System Using Machine Learning and Deep Learning for IoT Networks

By Shailja Roy

The rapid expansion of the Internet of Things (IoT) has revolutionized connectivity and automation across various domains but also introduces substantial cybersecurity risks. Traditional security measures struggle to counter sophisticated cyberattacks in IoT environments, necessitating advanced defence mechanisms.

This study presents a robust Network Intrusion Detection System (NIDS) for IoT networks using machine learning (ML) and deep learning (DL) techniques. Leveraging the UNSW-NB15 dataset, which contains both normal and malicious network traffic—two supervised ML classifiers, Decision Tree (DT) and Random Forest (RF), along with three sequential neural network models, were implemented for binary classification of network behaviour as either "Normal" or "Attack."

The Decision Tree model achieved 92% accuracy on the training set and 91% on the test set, while the Random Forest model attained 94% training accuracy and 91% test accuracy. Among the three neural network models, the best-performing model—featuring six hidden layers with 32 neurons per layer—achieved high accuracy and recall on both the training and test sets.

To enable real-time intrusion detection, a Flask API was developed, allowing for network traffic classification through API requests. This work underscores the importance of ML and DL-based NIDS in addressing evolving cybersecurity threats and ensuring the security of IoT ecosystems.

**Acknowledgements**

I would like to extend my deepest gratitude to my committee chair, **Dr. Armin Moghadam**, for his invaluable guidance in the selection and development of this project. His unwavering support, insightful expertise, and mentorship were instrumental to the successful completion of this work. I am equally grateful to my committee members, **Dr. Riti Gour** and **Prof. Bill Saichek**, for their generous contributions of time, thoughtful feedback, and constructive suggestions, which significantly enhanced the quality of this project.

Finally, I would like to express my heartfelt appreciation to my family for their constant encouragement, patience, and steadfast support throughout my academic journey. Their belief in my abilities has been a source of strength and motivation during every step of this process.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| TABLE OF CONTENTS | Page Number |
| ABSTRACT | 4 |
| ACKNOWLEDGEMENTS | 5 |
| TABLE OF CONTENTS | 6 |
| NOMENCLATURE | 7 |
| LIST OF TABLES | 8 |
| LIST OF FIGURES | 9 |
| 1.0 INTRODUCTION | 10 |
| 1.1         LITERATURE REVIEW | 15 |
| 1.2         OBJECTIVE | 21 |
| 2.0   METHODOLOGY |  |
| 2.1   UNSW-NB15 DATASET | 22 |
| 2.2   DATA PREPARATION | 25 |
| 2.3 DATA VISUALIZATION | 27 |
| 2.4   OVERVIEW OF ML MODELS | 29 |
| 2.5 OVERVIEW OF DL MODELS | 33 |
| 2.6 OVERVIEW OF FLASK API DEVELOPMENT | 35 |
| 3.0   RESULTS AND DISCUSSION | 37 |
| 4.0 CONCLUSION | 40 |
| REFERENCES | 42 |
| Appendix A: Dataset Features and Descriptions | 45 |
| Appendix B: Model Training and Hyperparameters | 47 |
| Appendix C: Flask API Deployment Instructions | 49 |

**NOMENCLATURE**

|  |  |
| --- | --- |
| IoT | Internet of Things |
| NIDS | Network Intrusion Detection System |
| ML | Machine Learning |
| DL | Deep Learning |
| DT | Decision tree |
| RF | Random Forest |
| CNN | Convolutional Neural Network |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| Table 1 - | Value count for service column | 25 |
| Table 2 - | UNSW-NB15 Training Set Results | 37 |
| Table 3 - | UNSW-NB15 validation Set Results | 38 |
| Table 4 - | UNSW-NB15 Test Set Results | 38 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Figure 1 | Applications of the Internet of Things (IoT) | 11 |
| Figure 2 | The Scenario of the IDS applied in the IoT network | 14 |
| Figure 3 | IXIA Traffic Generator Overview | 22 |
| Figure 4 | Pie chart distribution for normal and abnormal labels | 27 |
| Figure 5 | Class distribution graph for the train dataset | 27 |
| Figure 6 | Pie chart distribution of multi-class labels | 28 |
| Figure 7 | API development environment | 36 |
| Figure 8 | Sample API requests and responses | 39 |

1. **INTRODUCTION**

The rapid proliferation of the Internet of Things (IoT) has transformed the digital landscape, enabling unprecedented levels of connectivity and automation across various sectors. The Internet of Things refers to a network of physical objects that are connected using the Internet protocols in the TCP/IP stack and can communicate with each other. These objects, also known as “smart devices”, are equipped with sensors, software, and other technologies that allow them to collect and exchange data. Thanks to the advent of inexpensive computer chips and high bandwidth telecommunication, we now have billions of devices connected to the Internet of Things. This means everyday devices like toothbrushes, vacuums, cars, and even industrial machines—such as factory robots, smart appliances, or medical equipment—can use sensors to collect telemetry and respond intelligently to users.

The Internet of Things integrates everyday “things” with the Internet using TCP/IP protocols. Computer Engineers have been adding sensors, actuators, controllers and processors to everyday objects since the 1990s.

A typical IoT system works through the collection and exchange of real-time telemetry. An IoT system has three components:

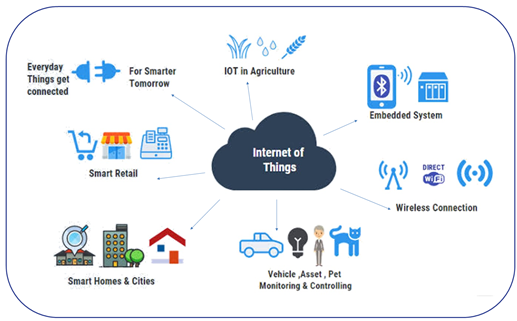
### **Smart Devices:**

Smart devices are physical objects embedded with sensors, processing capabilities, and networking features. These devices are capable of collecting telemetry from the environment (e.g., temperature, motion, light), user interactions (e.g., pressing a button, voice commands), or usage patterns (e.g., exercise data, appliance usage). Examples include:

* **Smart Thermostats**: Collect temperature data and allow users to control them remotely via apps.
* **Smart Speakers**: Equipped with microphones and speech recognition, these devices can collect audio data to provide voice-based commands and responses.
* **Wearables**: Devices like fitness trackers that collect physiological data, such as heart rate or steps taken, to monitor health.

These smart devices communicate with the IoT platform to send telemetry data and receive commands or updates, allowing remote control and automation.

Figure 1. Applications of the Internet of Things (IoT)



### **IoT Application:**

An IoT application is essentially the backend infrastructure that collects, processes, and acts upon the telemetry data from smart devices. This may involve:

* **Telemetry Data Processing**: The application often uses machine learning (ML) or artificial intelligence (AI) algorithms to process large amounts of data. This can include predictive analytics (e.g., predicting a machine failure based on usage patterns), anomaly detection (e.g., detecting abnormal energy consumption), or optimizing system behavior (e.g., adjusting heating or cooling systems based on ccupancy).
* **Decision-Making**: Once processed, the application makes decisions that can be sent back to actuators (devices that perform actions based on the data, such as motors or heating elements) or to cloud-based data stores for further analysis.

For example, in a smart home, the IoT application might control lighting, HVAC systems, or security cameras based on sensor data, time of day, and user preferences.

### **Graphical User Interface (GUI):**

The GUI serves as the interface through which users interact with their IoT devices and systems. This can be through mobile applications or web platforms, offering functionalities such as:

* **Device Control**: Users can turn devices on or off, change settings, or schedule tasks.
* **Telemetry Visualization**: Users can view real-time and historical data collected from devices in the form of graphs, charts, and reports.
* **Notifications and Alerts**: The GUI may send notifications about important events or issues, such as low battery, device malfunction, or threshold breaches (e.g., temperature exceeding a set limit).

For example, a mobile app might allow users to control smart lighting, check the status of security cameras, or receive an alert when their refrigerator is left open.

### **Technologies Used in IoT Systems**:

1. **Edge Computing**: Edge computing refers to processing data closer to where it is generated—on the device or at a nearby edge server—rather than sending all data to a central cloud server. This helps to:
   * **Reduce Latency**: By processing data at the edge, response times are quicker, which is critical in real-time applications such as autonomous vehicles or industrial automation.
   * **Bandwidth Efficiency**: Only important or processed data may need to be sent to the cloud, reducing the strain on network bandwidth.
   * **Local Decision-Making**: Devices can take action without waiting for cloud-based decisions, increasing system responsiveness.

Edge computing is particularly important for time-sensitive IoT applications, such as healthcare monitoring systems or smart manufacturing

1. **Fog Computing**: Fog computing is a decentralized computing infrastructure that extends cloud computing capabilities closer to the data source, but not directly on the device like edge computing. It acts as a middle layer between edge devices and the cloud, enabling processing, storage, and analysis to occur at local or regional network nodes (like gateways or routers).

Key benefits of fog computing in IoT include:

* **Distributed Processing**: Fog nodes can process and filter data before it reaches the cloud, reducing latency and improving real-time responsiveness.
* **Enhanced Security**: Sensitive data can be processed locally rather than being transmitted to the cloud, minimizing exposure to external threats.
* **Scalability**: It supports large-scale, distributed IoT systems by offloading processing tasks from both the edge and the cloud.
* **Real-Time Analytics**: Fog nodes can perform real-time analytics and send only summarized or critical data to the cloud, optimizing network usage.

An example use case is a smart city traffic system: Fog nodes installed at traffic lights or intersections can process video feeds and sensor data locally to make immediate traffic flow decisions, while sending aggregated data to the cloud for long-term planning and analysis.

1. **Cloud Computing**: Cloud computing is used for scalable, centralized storage and processing power. It provides remote storage solutions for large volumes of telemetry data and enables centralized device management. Key advantages of cloud computing in IoT include:
   * **Data Storage**: Cloud platforms, like AWS IoT, Microsoft Azure, and Google Cloud IoT, provide scalable storage options for storing large amounts of telemetry data generated by devices.
   * **Remote Management**: IoT devices can be managed and updated remotely through cloud-based platforms. This includes tasks such as firmware updates, device configuration, and monitoring.
   * **Analytics**: Cloud computing can offer advanced analytics tools and machine learning models to derive insights from the telemetry data.

An example is an IoT system for monitoring agricultural sensors: Data collected from soil moisture sensors can be stored and analyzed in the cloud, and the system can send irrigation commands to the devices accordingly.

1. **Networking Protocols**: IoT systems rely on various messaging and communication protocols to transmit data between devices and platforms, including:
   * **MQTT**: A lightweight messaging protocol used in IoT for low-bandwidth communication.
   * **HTTP/HTTPS**: Often used for web-based communication, especially when interacting with APIs or web interfaces.
   * **LoRaWAN**: A low-power, wide-area networking protocol for long-range communications, ideal for applications like smart agriculture or smart cities.
   * **Zigbee/Z-Wave**: Short-range communication protocols commonly used in home automation.

Together, these technologies allow IoT systems to function seamlessly by enabling real-time communication, efficient data processing, and large-scale data management.

**Machine Learning and Deep Learning in IoT Security**

Machine Learning (ML) refers to the software and algorithms used to process telemetry to make real-time decisions. These machine-learning algorithms can be deployed in the cloud or at the edge. Deep Learning (DL), a subset of ML, leverages neural networks with multiple layers to model complex patterns in data, offering even greater capabilities for tasks such as image recognition, natural language processing, and anomaly detection in network traffic.

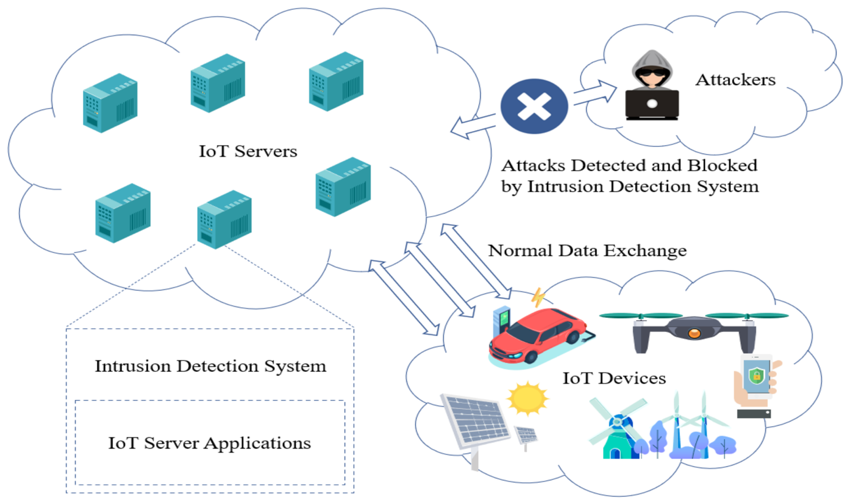
Used in industrial, commercial, and residential environments, IoT devices are integral to modern life, driving innovation and enhancing efficiency. By 2025, IoT devices are estimated to generate over 79.4 zettabytes of data annually, highlighting the critical need for efficient data management and security. According to studies, 57% of IoT devices are vulnerable to medium or high-severity attacks, emphasising the critical need for robust security solutions like intrusion detection systems. One of the greatest threats to IoT security is the lack of encryption on regular transmissions. Many IoT devices don't encrypt the data they send, which means if someone penetrates the network, they can intercept credentials and other important information transmitted to and from the device.

Network Intrusion Detection Systems (NIDS) attempt to detect cyber-attacks, malware, denial of service (DoS) attacks, or port scans on a computer network or a computer itself. NIDS monitor network traffic and detect malicious activity by identifying suspicious patterns in incoming packets. Any malicious activity or violation is typically reported to an administrator or collected centrally using a security information and event management (SIEM) system. Traditional security measures, often designed for conventional IT networks, are inadequate for the unique demands of IoT ecosystems.

The necessity for robust security in IoT networks has led to the development of Network Intrusion Detection Systems (NIDS) that leverage Machine Learning (ML) and Deep Learning (DL) techniques. Unlike traditional NIDS that rely on predefined rules and signatures, ML and DL-based systems can dynamically adapt to new threats, detecting anomalies and potential intrusions with higher accuracy. These systems can analyse vast amounts of network traffic data in real time, identifying patterns and behaviours that may indicate malicious activity. Deep Learning, in particular, excels at processing high-dimensional data and uncovering intricate relationships within the data, making it highly effective for identifying sophisticated and previously unseen threats.

This research presents the design and implementation of a Machine Learning (ML) and Deep Learning (DL)-based Network Intrusion Detection System (NIDS) specifically tailored for Internet of Things (IoT) networks. The proposed system addresses the unique security challenges inherent in IoT environments, with the objective of strengthening network protection and preserving the integrity of connected devices. The incorporation of advanced Deep Learning techniques enhances the system's capability to identify and respond to emerging threats with high accuracy and adaptability, contributing to a more resilient and robust security architecture for IoT ecosystems.

Figure 2. The Scenario of the IDS applied in the IoT network.



* 1. **LITERATURE REVIEW**

The paper [1] presents a machine learning (ML)-based approach for network intrusion detection (NID) in Internet of Things (IoT) systems, addressing the growing security risks associated with the proliferation of IoT devices. Using a publicly available NID dataset, the study evaluates four ML models: Naive Bayes (NB), Decision Tree (DT), K-Nearest Neighbor (KNN), and Logistic Regression (LR). The methodology involves comprehensive data preprocessing steps, including cleaning, integration, transformation, feature selection, and data splitting, followed by performance evaluation using metrics such as accuracy, precision, recall, and F1-score. Among the tested models, DT demonstrated the highest accuracy of 99.47%, outperforming KNN (99.16%), LR (95.55%), and NB (90.68%). The results highlight the potential of ML techniques in effectively identifying network threats, providing valuable insights for enhancing IoT security.

The paper [2] investigates Intrusion Detection Systems (IDS) tailored for IoT networks to address growing security threats due to IoT's limited computational capabilities. It proposes using Long Short-Term Memory (LSTM), a deep learning technique, and K-Nearest Neighbor (KNN), a machine learning method, for detecting cyber-attacks such as DDoS, DoS, and service scans using the Bot-IoT dataset. The dataset is preprocessed, features are selected using the Gain Ratio method, and performance is evaluated based on detection time, geometric mean, Kappa statistic, and accuracy. Experimental results indicate that LSTM outperforms KNN in terms of accuracy, geometric mean, and detection time, making it more effective and suitable for IoT's resource-constrained environments.

This paper [3] proposes a Big Data-based Distributed Denial of Service (DDoS) Network Intrusion Detection System (NIDS) to effectively handle DDoS attacks in distributed networks. The system employs a modular design, utilizing a collector module for data acquisition and preprocessing and a detector module for classification and threat detection. It integrates Random Forest (RF) for classification and Principal Component Analysis (PCA) for feature selection, supported by the Hadoop File System (HDFS) for scalable storage and Spark for efficient data processing. Evaluated using the NSL-KDD benchmark dataset, the model achieved a high accuracy of 99.89%, outperforming traditional machine learning methods like Decision Tree, Support Vector Machine (SVM), and Logistic Regression. The results demonstrate the system’s superior performance in terms of precision, recall, and F1-score, validating its efficacy for intrusion detection in distributed environments.

The paper [4] explores the critical issue of intrusion detection in the Internet of Things (IoT) environment, which faces significant cybersecurity challenges due to the rapid proliferation of connected devices and their inherent security vulnerabilities. It introduces a deep learning-based approach using a Recurrent Neural Network (RNN) to develop an Intrusion Detection System (IDS) tailored for IoT networks. Using the NSL-KDD dataset for training and testing, the proposed model demonstrates a promising accuracy of 87%, showcasing its capability to detect various attack types, including Denial of Service (DoS), Probing, Remote to Local (R2L), and User to Root (U2R). The study emphasizes the advantages of deep learning in anomaly detection and highlights the need for optimization techniques to further enhance detection accuracy. By comparing its results with other machine learning and deep learning models, the research underscores the effectiveness of the proposed RNN approach in addressing IoT network security concerns. Future work aims to improve model efficiency through optimization algorithms.

This paper [5] presents a Convolution Neural Network (CNN) based Intrusion Detection System (IDS) to address network security challenges by accurately detecting malicious activities with a low false alarm rate (FAR). Using the CIC-IDS2017 dataset, which includes diverse attack types and normal traffic, the model achieved 99.55% accuracy and 0.12% FAR for multiclass classification, and 99.56% accuracy with 0.4% FAR for binary classification. The architecture integrates three CNN layers with Rectified Linear unit (ReLU) activation, batch normalization, and dropout to enhance feature extraction and prevent overfitting. Stratified K-Fold cross-validation and hyperparameter optimization further improved model performance. The study highlights CNN's capability for automated feature extraction and proposes future work to explore hybrid models for greater robustness and scalability.

This research [6] explored the application of machine learning (ML) and deep learning (DL) techniques to enhance network intrusion detection systems (NIDS). The NSL-KDD dataset was used to train and test various ML models, including Random Forest, Decision Tree, Logistic Regression, K-Nearest Neighbor, Gaussian Naive Bayes, CatBoost, and XGBoost. Additionally, a Recurrent Neural Network (RNN) was employed as a DL model. While both ML and DL models achieved high accuracy during training, they exhibited significant performance degradation on the test set, indicating overfitting. To address this issue, the authors suggest refining data preprocessing, feature selection, and model hyperparameter tuning.

This study [7] investigates network security for IoT devices, focusing on intrusion detection to mitigate their vulnerability to cyberattacks. It compares four deep learning algorithms—DNN, CNN, LSTM, and AE—using the UNSW-NB15 dataset with binary classification. Among these, the Deep Neural Network (DNN) achieved the highest accuracy of 99.76% with the lowest loss value of 0.006%, outperforming the others in effectively detecting and preventing network security breaches. While CNN showed competitive accuracy and minimal loss, LSTM and AE exhibited lower performance due to issues like overfitting and instability. The findings emphasize the importance of robust intrusion detection systems and highlight DNN as the most effective approach for enhancing IoT network security.

This study [8] explores the application of Deep Neural Networks (DNN) for intrusion detection in IoT networks, focusing on the Bot-IoT dataset from UNSW, which includes traffic from smart home devices and various attack types. The dataset was tested in small and large-scale configurations, preprocessed to retain numeric features, and split into training (75%) and validation (25%) sets. Results demonstrate DNN's effectiveness in detecting attacks, achieving near-perfect accuracy of 99.999% across different dataset sizes. Despite minor overfitting in certain cases, the model's performance was consistent, indicating its robustness for intrusion detection in IoT environments. This research highlights DNN's potential for enhancing security in IoT systems by reliably identifying diverse attack patterns.

This paper [9] provides an extensive survey on Network Intrusion Detection Systems (NIDS), focusing on their application in IoT and wireless networks. It addresses the limitations of traditional intrusion detection methods that rely on single learning algorithms, which struggle with issues like computational complexity, high false alarm rates, and difficulty detecting encrypted or noisy traffic. The paper discusses various advanced techniques, including machine learning (ML) and deep learning (DL) models such as GA-FRCNN, ResNet-Transformer-BiLSTM, and federated deep belief networks, aimed at improving the accuracy and efficiency of intrusion detection. Performance is assessed using key metrics like accuracy, precision, recall, false alarm rate, and Matthews Correlation Coefficient (MCC). The study highlights the advantages of hybrid approaches, such as feature selection and ensemble learning, but also points out challenges like adversarial attacks, energy consumption, and overfitting. The paper concludes that combining multiple techniques in an ensemble approach could significantly enhance detection capabilities, making NIDS more effective in the face of rapidly evolving threats in IoT and wireless networks.

This paper [10] addresses the growing security concerns in Internet of Things (IoT) networks, particularly with regard to detecting malicious activities. Given the limited computational power of many IoT devices, applying traditional security measures such as encryption is challenging. To tackle this, the paper focuses on anomaly-based Intrusion Detection Systems (IDS), utilizing machine learning algorithms to identify malicious activities within IoT networks. The authors experiment with multiple machine learning approaches, including Logistic Regression, Support Vector Machine, K-Nearest Neighbors, Random Forest, and XGBoost, on the IoT Network Intrusion Dataset. The study shows that the Random Forest approach outperforms others, achieving 100% accuracy in detecting attacks, including Denial-of-Service and Mirai Botnet attacks. The paper highlights the importance of applying machine learning techniques to IoT network security and demonstrates the potential of IDS to accurately detect anomalies with high efficiency.

This paper [13] presents a comprehensive analysis of Intrusion Detection Systems (IDS) for Internet of Things (IoT) networks, focusing on the use of machine learning and deep learning techniques to detect malicious activities. Recognizing the limitations of traditional security measures on resource-constrained IoT devices, the authors explore various ML and DL algorithms, including Random Forest, Decision Tree, AdaBoost, XGBoost, Convolutional Neural Networks, Deep Convolutional Neural Networks, Long Short-Term Memory, and Deep Neural Networks. The paper offers a taxonomy of IDS approaches, evaluates their performance, and highlights the benefits and challenges of each method. It finds that while ML models offer efficient detection, DL models excel at feature extraction and handling complex attacks, but often demand higher computational resources. The study emphasizes the need for further research to address challenges such as high false positive rates and model generalisation, reinforcing the importance of intelligent IDS for robust IoT network security.

This paper [14] addresses the challenges of applying machine learning to intrusion detection systems (IDSs) due to limitations such as unbalanced datasets and the need for high technical expertise. To tackle these issues, the study enhances the performance of deep learning (DL) structures for IDS by using data augmentation techniques. The authors experiment with various deep learning architectures, including CNNs, LSTMs, and GRUs, on the UNSW-NB15 dataset. The results indicate that even a basic CNN-based architecture can achieve very efficient network-attack-detection, with more complex designs showing only slight performance increases. The paper demonstrates that data augmentation improves the accuracy of intrusion detection models, with augmented CIC-IDS-2017 dataset showing an accuracy of up to 92%, highlighting the potential of deep learning-based IDS to effectively identify and counteract complex network threats.

This paper [15] addresses the growing security concerns in Internet of Things (IoT) networks, particularly regarding the detection of malicious activities. The eccentric and heterogeneous nature of IoT networks demands specific security requirements and algorithms, which differ from those in traditional networks. To tackle this, the paper explores the use of Machine Learning (ML) and Deep Learning (DL) methodologies to overcome security problems in IoT networks and preserve data privacy. It reviews learning-based Intrusion Detection Systems proposed as countermeasures to security breaches like Denial of Service, spoofing, or eavesdropping attacks in IoT environments. The paper discusses the application of various ML algorithms like Decision Tree, Support Vector Machine, Naïve Bayes, Random Forest, K-Nearest Neighbour, and K-Means Clustering, as well as DL models like Convolutional Neural Networks, Recurrent Neural Networks, and Deep Encoders for intrusion detection. It also highlights the importance of reliable datasets like DARPA98, KDD Cup 99/KDD99, and NSL-KDD for evaluating the efficiency of Intrusion Detection Systems.

This paper [16] addresses the security concerns in cloud-enabled IoT networks, where IoT devices, gateways, and the cloud communicate, making them vulnerable to attacks. It proposes a Robust Intrusion Detection and Prevention System (RIDPS) that uses machine learning (ML) techniques to secure communication between IoT devices and the cloud. The paper explores the application of various ML algorithms: K-means clustering for anomaly/intrusion detection, Support Vector Machines (SVM) for attack detection and mitigation, C4.5 Decision Tree for defending against Distributed Denial of Service (DDoS) attacks, and Random Forest for malware analysis. The proposed RIDPS aims to filter traffic, detect attacks, and predict future intrusions, enhancing the security and privacy of IoT systems.

This paper [17] reviews the application of Intrusion Detection Systems (IDS) in IoT networks, addressing the vulnerability of IoT devices to security attacks. It discusses the necessity of employing machine learning techniques to develop effective IDS for IoT environments, given the unique challenges posed by these networks. The review analyses various machine learning algorithms used in IDS-IoT research from 2015 to 2020, with Artificial Neural Networks (ANN) being the most frequently utilised, followed by Recurrent Neural Networks (RNN) and Deep Neural Networks (DNN). The paper also examines the types of datasets used for training and evaluating IDS, highlighting the importance of using datasets that are relevant to the specific characteristics of IoT networks.

This paper [18] addresses the increasing security vulnerabilities in Internet of Things (IoT) systems, emphasizing the importance of Intrusion Detection Systems (IDSs) in this context. It explores the use of machine learning (ML) techniques to develop an effective IDS, utilizing the TON\_IoT network dataset, a relatively new and comprehensive dataset designed to simulate real-world IoT environments. The authors compare ten different ML algorithms to identify the optimal solution for anomaly detection. The key result is the selection of XGBoost as the top-performing algorithm, demonstrating its ability to achieve high accuracy in both binary (normal vs. malicious) and multiclass (specific attack type) classifications. Specifically, the XGBoost model achieved a Matthews correlation coefficient (MCC) of 99.84% in binary classification and 99.17% in multiclass classification, highlighting its effectiveness in accurately detecting and classifying intrusions in IoT networks. The paper concludes by emphasizing the significance of these findings for the development of real-world IoT IDSs.

This paper [19] addresses the rising security concerns in IoT by proposing an improved intrusion detection system. It utilizes a combination of a feed-forward artificial neural network for feature extraction and the eXtreme Gradient Boosting (XGBoost) algorithm for classification, leveraging the BoT-IoT dataset. The results of the proposed model show satisfactory performance in intrusion detection, achieving approximately 99% accuracy, which is better than the approximately 97% accuracy obtained by a feed-forward artificial neural network alone. The proposed model also demonstrates improvements in detection rate and a reduction in the false alarm rate, particularly for DoS attacks and normal traffic, compared to using only an artificial neural network.

This paper [20] addresses the growing concern of security in IoT networks due to the increasing number of internet-connected devices. It focuses on the development of a deep learning algorithm-based model designed specifically for the detection of Denial-of-Service (DoS) attacks, a significant threat to IoT systems. The model is implemented using Python libraries such as Seaborn, TensorFlow, and scikit-learn, demonstrating the applicability of these tools in enhancing IoT security. The results of this approach indicate that the proposed model achieves efficient DoS attack mitigation and improved accuracy in IoT networks, highlighting the potential of deep learning techniques to bolster the security of these increasingly vital systems.

This paper [21] addresses the increasing problem of cyber-attacks on IoT networks by proposing an intrusion detection model based on anomaly detection. The study uses convolutional neural networks (CNN) to categorize input data and identify anomalies. A multiclass classification framework is developed using a CNN model, and further implemented with a 3D CNN to enhance the analysis of image data for intrusion detection. The results of the proposed multiclass and binary classification framework demonstrate improved performance, achieving higher F1 scores, recall, precision and accuracy compared to conventional deep learning models. This enhanced performance highlights the potential of the CNN-based approach to more effectively detect and classify various types of intrusions in IoT networks, contributing to stronger security for these systems.

This paper [22] introduces a unique intrusion detection model for IoT networks, employing deep learning methods such as Convolutional Neural Networks (CNN), TabNet, and the Transient Search Optimization (TSO) algorithm. The model involves developing and evaluating distinct intrusion detection components: CNN for feature extraction in network traffic analysis, TabNet for capturing intricate relationships in structured data, and the TSO algorithm for enhancing the system's ability to detect temporary anomalies. The results of the study indicate that TabNet excels in identifying patterns in structured IoT data, outperforming CNN, and the TSO algorithm improves the recognition of temporary anomalies, thus enhancing IoT network security. Furthermore, an ensemble technique combining CNN and TabNet predictions achieves an accuracy rate exceeding 99%, demonstrating significant improvements in intrusion detection performance.

**1.2 OBJECTIVE**

To identify a real-time intrusion detection system model using Machine Learning (ML) and Deep Learning (DL) that classifies network traffic as normal or malicious for the IoT environment.

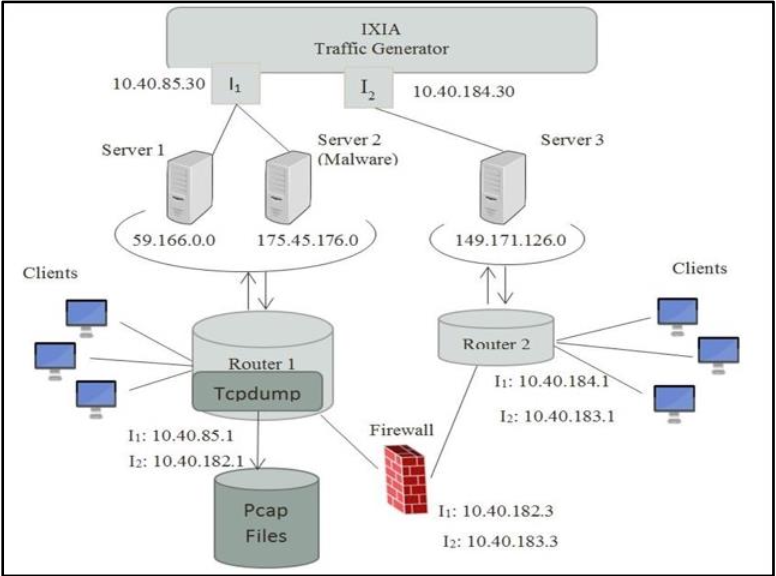
**2.0 METHODOLOGY**

**2.1 UNSW-NB15 Dataset**

The **UNSW-NB15 dataset** is a widely recognized benchmark in the domain of **Network Intrusion Detection Systems (NIDS)** and cybersecurity research. It is specifically designed to simulate contemporary network traffic patterns, encompassing both legitimate activities and a diverse range of malicious attack behaviours. This comprehensive representation of network interactions makes the UNSW-NB15 dataset highly suitable for the development and evaluation of **Machine Learning (ML)** and **Deep Learning (DL)** models aimed at intrusion detection.

Figure 2 illustrates the **testbed configuration** and the **feature engineering process** employed in the creation of the UNSW-NB15 dataset, as detailed in [11]. This structured configuration ensures the dataset accurately reflects real-world network environments, providing a robust foundation for model training and performance assessment.

Figure 3. IXIA Traffic Generator Overview



### Key Features of the UNSW-NB15 Dataset

1. **Purpose**:
   * The dataset was created to address the limitations of older datasets like KDD99 and NSL-KDD, which were outdated and did not reflect modern network traffic patterns or attack types.
   * It provides a hybrid of real normal network activities and synthetic attack behaviours, making it more representative of contemporary IoT-based networks.
2. **Data Generation**:
   * The raw network packets were generated using the **IXIA PerfectStorm tool** in the Cyber Range Lab of the **Australian Centre for Cyber Security (ACCS)**.
   * The **tcpdump** tool was used to capture 100 GB of raw network traffic in the form of **Pcap files**.
   * The dataset was further processed using tools like **Argus** and **Bro-IDS** (now known as Zeek) to extract features and label the data.
3. **Attack Categories**:
   * The dataset includes **nine categories of attacks** and one category for normal traffic. The attack types are:
     + **Fuzzers**: Attackers send large amounts of random data to crash systems.
     + **Analysis**: Includes port scanning, spam, and phishing.
     + **Backdoors**: Malicious code that allows unauthorized access to systems.
     + **DoS (Denial of Service)**: Overwhelms systems to disrupt services.
     + **Exploits**: Exploits vulnerabilities in systems to gain unauthorized access.
     + **Generic**: Represents general types of attacks.
     + **Reconnaissance**: Gathering information about the target system.
     + **Shellcode**: Malicious code used to exploit vulnerabilities.
     + **Worms**: Self-replicating malware that spreads across networks.
4. **Dataset Structure**:
   * The dataset is provided in **CSV format** and consists of four files:
     + UNSW-NB15\_1.csv, UNSW-NB15\_2.csv, UNSW-NB15\_3.csv, and UNSW-NB15\_4.csv.
   * It is pre-partitioned into:
     + A **training set**: UNSW\_NB15\_training-set.csv (175,341 records).
     + A **testing set**: UNSW\_NB15\_testing-set.csv (82,332 records).
   * The dataset contains **49 features** (attributes) and a **class label** indicating whether the record is normal or an attack.
5. **Features**:
   * The 49 features include a mix of **flow-based features** (e.g., duration, protocol, service) and **content-based features** (e.g., packet length, packet count).
   * Examples of features:
     + **srcip**: Source IP address.
     + **sport**: Source port.
     + **dstip**: Destination IP address.
     + **dsport**: Destination port.
     + **proto**: Protocol type (e.g., TCP, UDP).
     + **state**: State of the connection.
     + **dur**: Duration of the flow.
     + **sbytes**: Number of bytes sent from source to destination.
     + **dbytes**: Number of bytes sent from destination to source.
     + **attack\_cat**: Category of the attack (e.g., DoS, Exploits).
     + **label**: Binary classification (0 for normal, 1 for attack).
6. **Testbed Configuration**:
   * The dataset was generated using a testbed with three virtual servers:
     + **Server 1 and Server 3**: Configured to generate normal traffic.
     + **Server 2**: Configured to generate malicious/abnormal traffic.
   * The **IXIA traffic generator** was used to simulate the hybrid traffic.

### Advantages of UNSW-NB15

* **Modern Representation**:
  + It reflects contemporary network traffic and attack behaviors, unlike older datasets.
* **Diverse Attack Types**:
  + Covers a wide range of attack categories, making it suitable for comprehensive evaluation.
* **Pre-Partitioned**:
  + The dataset is already split into training and testing sets, simplifying the evaluation process.
* **Feature-Rich**:
  + The 49 features provide a detailed representation of network traffic, enabling robust model training.

### Challenges and Limitations

* **Imbalanced Data**:
  + The dataset is imbalanced, with some attack types having significantly fewer samples than others.
* **Feature Engineering**:
  + Some features may require preprocessing or normalisation before being used in machine learning models.

[11] provides a detailed description of the dataset creation process, features, and attack categories, and [12] explores the dataset through visualisation techniques, providing insights into its structure and characteristics.

**2.2 DATA PREPARATION**

For this project, a pre-partitioned dataset is utilized, comprising a **training set** and a **testing set** for model development and evaluation, respectively. The training set consists of **175,341 records**, while the testing set includes **82,332 records**, each annotated with a target response indicating the **traffic behaviour**, classified as either **attack** or **normal**. The dataset encompasses a total of **44 features**, of which **40 are numerical** and **4 are categorical**. These features capture various network characteristics, enabling the machine learning model to effectively learn and predict intrusion patterns.

 **Data Collection & Loading:**

* The first step involved gathering data from sources such as CSV files. The collected data was then loaded into a suitable format, such as a Pandas DataFrame, for further processing.

 **Understanding the Dataset:**

* To explore the dataset structure, feature types, and value distributions, I used commands like data.head(), data.info(), and data.shape().

 **Data Cleaning:**

* The **"id"** column was dropped as it served only as a serial number and held no significance for prediction.
* The **"service"** column contained the value '-' in **94,168 rows**. According to the research paper, this indicates underutilized services. I replaced '-' with None for better handling.

Table 4. Value count for service column

|  |  |
| --- | --- |
| **Service** | **Count** |
| - | 94168 |
| dns | 47294 |
| http | 18724 |
| smtp | 5058 |
| ftp-data | 3995 |
| ftp | 3428 |
| ssh | 1302 |
| pop3 | 1105 |
| dhcp | 94 |
| snmp | 80 |
| ssl | 56 |
| irc | 25 |
| radius | 12 |

* Missing values were identified using data.isnull().sum(). Common techniques to handle missing values include:
  1. Removing rows or columns with excessive missing values.
  2. Imputing missing values using mean, median, or mode.
* The **"is\_ftp\_login"** column is a binary feature (0 or 1). However, some entries contained incorrect values (4 and 2). These rows were removed.

 **Feature Encoding & Scaling:**

* Machine Learning and Deep Learning models require numerical input. Therefore, categorical variables were converted into numerical form using **One-Hot Encoding**, while numerical features were standardized using **StandardScaler**.
* **One-Hot Encoding (OHE):**
  + Converts categorical variables into binary vectors (0 or 1) to prevent models from assuming ordinal relationships.
* **StandardScaler:**
  + Standardizes numerical features by transforming them to have **zero mean** and **unit variance**, ensuring all features contribute equally to the model.

 **Data Transformation Pipeline:**

* Instead of applying transformations separately, I used **Pipeline** and **ColumnTransformer** to streamline the data preprocessing workflow efficiently.

**2.3 DATA VISUALIZATION**

In the context of binary classification, the target variable in the dataset is represented by the attribute label, which consists of two distinct classes: **Normal (class 0)** and **Attack (class 1)**. The distribution of these classes is illustrated in the pie chart (Figure 4) and the graph (Figure 5), revealing that **32% of the network traffic** is classified as normal (class 0), while the remaining **68% is categorized as attack traffic** (class 1) within the training dataset. This distribution highlights an imbalance, which may influence model performance and necessitate appropriate handling during the training process.

Figure 4. Pie chart distribution for normal and attack labels

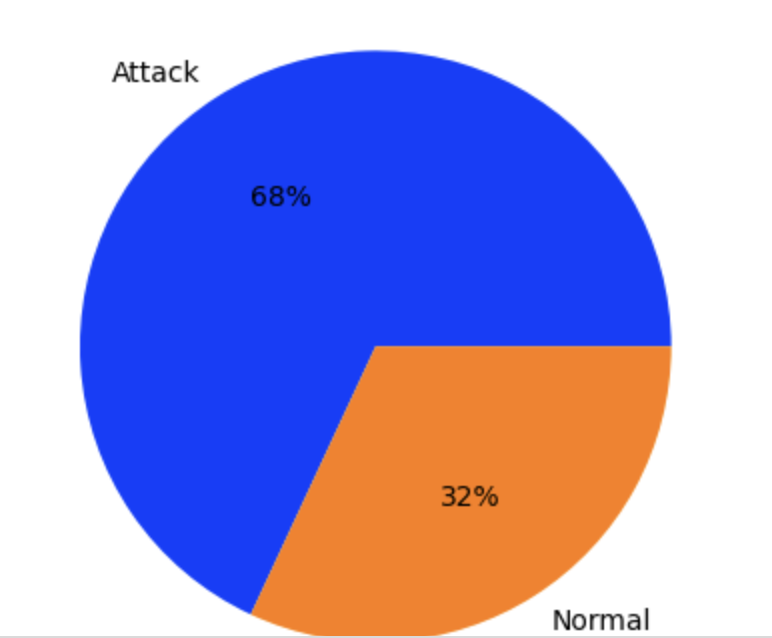
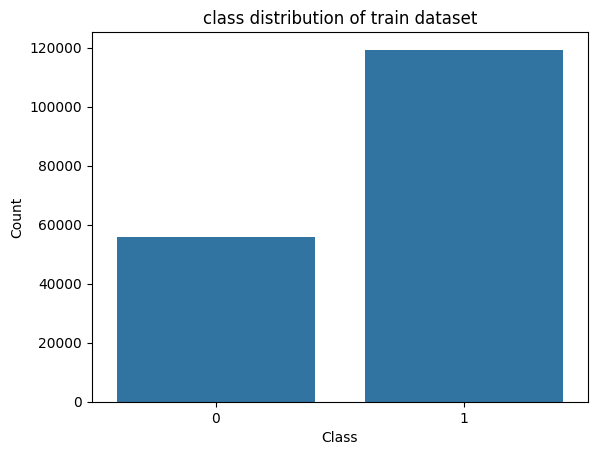
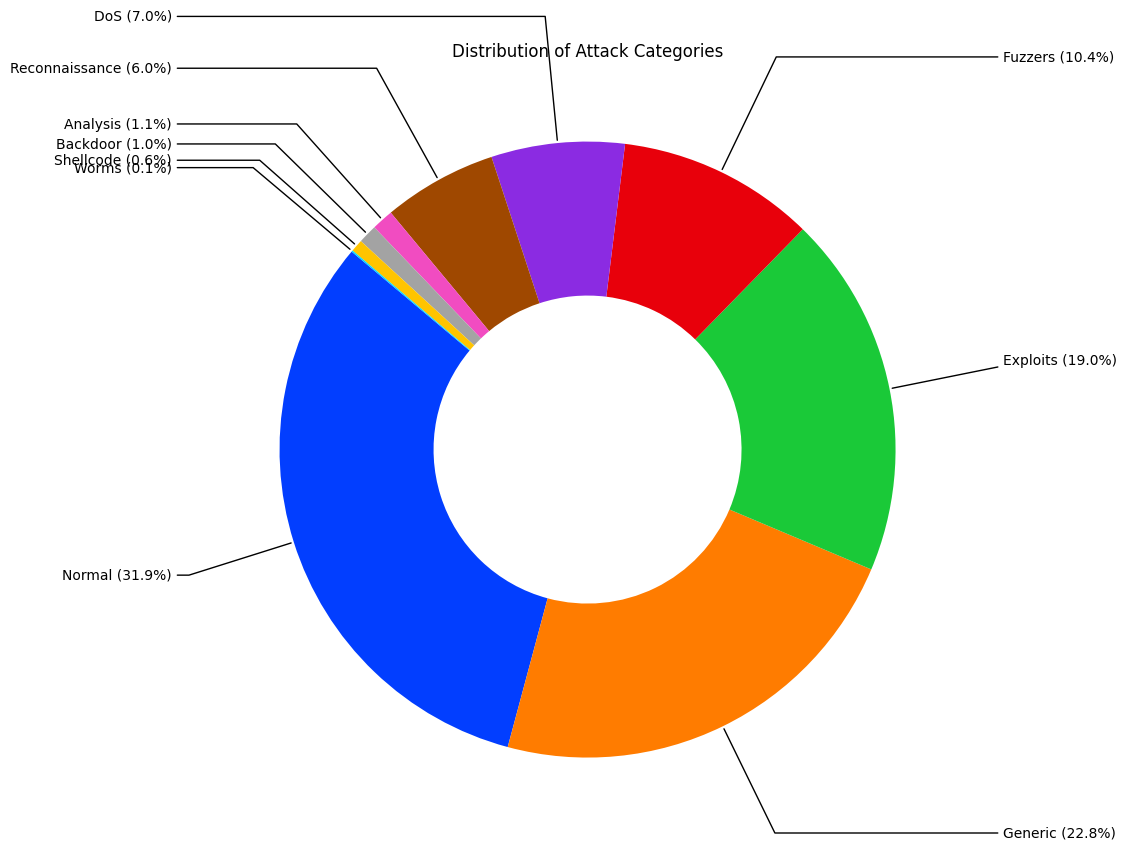


Figure 5. Class distribution graph for the train dataset



The dataset includes an attack\_cat column, which categorically represents various types of network attacks, enabling the potential for multi-class classification. Given that attack\_cat is a categorical variable, one-hot encoding is applied using pandas.get\_dummies(), resulting in an expanded DataFrame with binary-encoded columns. These columns, such as attack\_cat\_Normal, attack\_cat\_Backdoor, attack\_cat\_Fuzzers, attack\_cat\_Reconnaissance, attack\_cat\_Exploits, attack\_cat\_Analysis, attack\_cat\_DoS, attack\_cat\_Worms, and attack\_cat\_Generic, facilitate seamless integration into machine learning models by providing a structured numerical format. The distribution of these attack categories is visually represented in the pie chart shown in Figure 6, offering insights into the prevalence of each class within the dataset.

Figure 6: Pie chart distribution of multi-class labels



**2.4 OVERVIEW OF ML METHODS**

In this project, two **supervised machine learning (ML) algorithms** were employed to construct a **binary classification model**:

1. **Decision Tree Classifier**
2. **Random Forest Classifier**

These algorithms were selected based on their proven effectiveness in handling classification tasks, interpretability, and ability to manage large datasets with high-dimensional features.

**Decision Tree Classifier**

Decision Trees are a versatile and powerful machine learning algorithm used for both classification and regression tasks. They are capable of fitting complex datasets by recursively splitting the data into smaller subsets based on feature values. In this project, a Decision Tree classifier is employed to classify network behavior as either "Normal" or "Attack" based on selected features. The features were chosen by considering their linear relationship with the target variable using a correlation matrix.

#### Decision Tree Algorithm Overview

The Decision Tree algorithm works by recursively partitioning the data into subsets based on the values of input features. The goal is to create subsets that are as pure as possible with respect to the target variable. The process involves the following steps:

1. **Choosing the Best Feature to Split**: The algorithm selects the feature that best separates the classes. This is typically done using impurity measures such as Gini impurity or Entropy (Information Gain).
2. **Splitting the Dataset**: The dataset is split into subsets based on the selected feature.
3. **Repeating the Process**: The splitting process is repeated recursively for each subset until all data points are classified or certain stopping conditions are met (e.g., maximum depth, minimum samples per leaf).

#### Explanation of Hyperparameters used to train the model:

1. Criterion ("**entropy**"):
   * Description: The criterion used to measure the quality of a split. "Entropy" is used here, which measures the information gain from each split. It aims to maximize the information gain, leading to more homogeneous subsets.
   * Impact: Using entropy helps in creating splits that result in subsets with higher purity, which can improve the classifier's accuracy.
2. Max Depth (**max\_depth=12):**
   * Description: The maximum depth of the tree. This parameter controls the maximum number of levels in the tree.
   * Impact: Limiting the depth helps prevent overfitting by restricting the tree from growing too complex. A depth of 12 allows the tree to capture intricate patterns without becoming overly complex.
3. Minimum Samples Split (**min\_samples\_split=10**):
   * Description: The minimum number of samples required to split an internal node.
   * Impact: Setting this to 10 ensures that splits are only made if there are at least 10 samples in a node, which helps in preventing overfitting by avoiding splits on nodes with very few samples.
4. Minimum Samples Leaf (**min\_samples\_leaf=5**):
   * Description: The minimum number of samples required to be at a leaf node.
   * Impact: This parameter ensures that each leaf node has at least 5 samples, which helps in creating more robust and generalizable trees.
5. Maximum Features (**max\_features=0.8**):
   * Description: The number of features to consider when looking for the best split. Here, 80% of the features are considered.
   * Impact: Limiting the number of features considered for each split introduces randomness and can help in reducing overfitting, especially in datasets with a large number of features.
6. Cost-Complexity Pruning (**ccp\_alpha=0.001**):
   * Description: The complexity parameter used for Minimal Cost-Complexity Pruning. It controls the trade-off between the tree's complexity and fits to the training data.
   * Impact: A small value of ccp\_alpha (0.001) allows for some pruning, which helps in reducing the tree's complexity and improving generalization.
7. Class Weight ("**balanced**"):
   * Description: Weights associated with classes. "Balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies.
   * Impact: This is particularly useful in imbalanced datasets, where one class might dominate. By balancing the weights, the classifier pays more attention to the minority class, improving its performance.
8. Random State (**random\_state=42**):
   * Description: A seed value used to initialize the random number generator. This ensures that the results are reproducible.
   * Impact: Setting a fixed random state ensures that the same results are obtained each time the code is run, which is crucial for debugging and comparison purposes.

#### **Training the Model**

The model is trained using the fit method, where **X\_train\_prepared** represents the prepared feature matrix and **y2\_train\_prepared** represents the corresponding target labels. The training process involves finding the optimal splits based on the specified hyperparameters to create a decision tree that can accurately classify the data.

**Random Forest Classifier**

Random Forest is a powerful ensemble learning method that is widely used for both classification and regression tasks. It operates by constructing multiple decision trees during training and outputs the mode of the classes (for classification) or the mean prediction (for regression) of the individual trees. The Random Forest algorithm is known for its robustness, accuracy, and ability to handle large datasets with higher dimensionality. It also provides methods for feature importance, making it a valuable tool for feature selection.

## **Key Concepts:**

### **Ensemble Learning**

Ensemble learning involves combining the predictions of multiple models to improve overall performance. Random Forest is a type of ensemble method that uses multiple decision trees to make more accurate and stable predictions.

### **Bagging (Bootstrap Aggregating)**

Bagging is a technique where multiple subsets of the training data are created by randomly sampling with replacement from the original dataset. Each subset is used to train an individual model, and the final prediction is obtained by aggregating the predictions of all models. This reduces variance and helps prevent overfitting.

### **Pasting**

Pasting is similar to bagging but without replacement. Each training instance appears in only one subset. While less common than bagging, pasting can be useful in certain scenarios.

### **Random Subspaces Method**

In Random Forest, each tree is trained on a random subset of features. This introduces diversity among the trees and reduces correlation, making the ensemble more robust.

### **Decision Tree Training**

Each Decision Tree in the forest learns from its sampled data and produces its own prediction. The trees are typically grown deep, and no pruning is done, which allows them to capture complex patterns in the data.

### **Majority Voting (Classification) or Averaging (Regression)**

For classification tasks, the final prediction is determined by majority voting (the most common class label among the trees). For regression tasks, the average of all three predictions is used.

#### Explanation of Hyperparameters used to train the model:

1. **n\_estimators=100**:
   * This specifies the number of decision trees in the Random Forest.
   * A higher number of trees generally improves performance but increases computational cost.
   * Here, 100 trees are used, which is a common default value.
2. **max\_depth=12**:
   * This controls the maximum depth of each decision tree in the forest.
   * Limiting the depth prevents overfitting by restricting the tree from growing too deep.
   * If not specified, trees will grow until all leaves are pure or contain fewer than min\_samples\_split samples.
3. **min\_samples\_split=10**:
   * This sets the minimum number of samples required to split an internal node.
   * A higher value prevents the model from creating splits with very few samples, which can lead to overfitting.
4. **min\_samples\_leaf=5**:
   * This specifies the minimum number of samples required to be at a leaf node.
   * A higher value ensures that each leaf has enough samples, reducing overfitting.
5. **max\_features=0.8**:
   * This determines the number of features to consider when looking for the best split.
   * Here, 80% of the features are randomly selected for each split.
   * This introduces randomness and reduces correlation between trees, improving the model's robustness.
6. **class\_weight='balanced'**:
   * This adjusts the weights of classes inversely proportional to their frequencies.
   * It is useful when dealing with imbalanced datasets, as it gives more importance to the minority class.
7. **n\_jobs=-1**:
   * This specifies the number of CPU cores to use for training.
   * -1 means using all available cores, which speeds up training for large datasets.
8. **random\_state=42**:
   * This sets a seed for the random number generator, ensuring reproducibility of results.
   * The same random state will produce the same results across multiple runs.
9. **bootstrap=True (default)**:
   * This indicates that bagging is used, where each tree is trained on a bootstrap sample (random sampling with replacement) of the training data.
   * If set to False, pasting is used instead, where each tree is trained on a subset of the data without replacement.

**2.5 OVERVIEW OF DL METHODS**

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of biological neurons. They serve as the foundation of **Deep Learning**, enabling machines to learn complex patterns from data. ANNs are widely used for tasks such as **image classification, speech recognition, natural language processing, and cybersecurity,** including **anomaly detection in network intrusion detection systems (NIDS).**

Unlike traditional Machine Learning models, which rely on handcrafted features, ANNs **automatically extract features** and identify non-linear patterns in large datasets. This ability makes them highly effective for detecting malicious activities in network traffic.

**A typical Artificial Neural Network consists of multiple layers**:

1. Input Layer:
   * The first layer of the network that receives raw data (network traffic features).
   * Each neuron in this layer represents an individual feature from the dataset (e.g., packet size, protocol type, service, etc.).
2. Hidden Layers:
   * One or more layers between the input and output layers that perform complex transformations on the input data.
   * Each neuron in a hidden layer applies a weighted sum operation followed by an activation function (such as ReLU, sigmoid, or tanh).
   * These layers allow the model to capture intricate relationships between network traffic attributes.
3. Output Layer:
   * Produces the final classification output.
   * In this particular case, the model predicts whether a given network flow is Normal (0) or Attack (1).
   * The Softmax or Sigmoid activation function is commonly used in binary classification tasks like this.

The deeper the network (i.e., the more hidden layers it has), the better its ability to learn complex patterns, but it also increases computational requirements and risks overfitting.

### **Training Process of a Neural Network**

#### Step 1: Forward Propagation

* Input data is passed through the network, layer by layer.
* Each neuron computes a weighted sum of inputs, applies an activation function, and passes the output to the next layer.
* The final layer produces a predicted class label (Normal or Attack).

#### Step 2: Loss Calculation

* The model compares its predictions with actual labels from the dataset.
* The Loss Function (such as Binary Cross-Entropy) quantifies the difference between the predicted and true labels.
* The goal is to minimize this loss to improve model accuracy.

#### Step 3: Backpropagation and Optimization

* Backpropagation calculates how much each weight in the network contributed to the loss.
* The gradients of the loss with respect to each weight are computed using Gradient Descent.
* Optimization algorithms like Adam, SGD (Stochastic Gradient Descent), or RMSprop adjust the weights to reduce the loss and improve predictions.

#### Step 4: Model Training and Evaluation

* The neural network undergoes multiple iterations (epochs) of forward propagation and backpropagation.
* The dataset is often split into training, validation, and test sets to measure model performance.
* Evaluation metrics like accuracy, precision, recall, and F1-score determine how well the model detects attacks.

Three different Deep Learning models were trained and evaluated for network intrusion detection:

#### Model 1: Basic Sequential Neural Network

* A simple Artificial Neural Network (ANN) with three hidden layers, each containing 64 neurons.
* Used ReLU activation for hidden layers to introduce non-linearity.
* The output layer used sigmoid activation for binary classification (Normal vs. Attack).
* Trained using the Adam optimizer with a predefined learning rate of 0.009.
* Loss function: Binary Cross-Entropy.

#### Model 2: Deep Neural Network (DNN) with Increased Depth

* Increased depth by adding six hidden layers, each with 32 neurons.
* Used ReLU activation for all hidden layers.
* The output layer used sigmoid activation for binary classification.
* Trained using the SGD (Stochastic Gradient Descent) optimizer with a learning rate of 0.009.
* Loss function: Binary Cross-Entropy.

#### Model 3: Optimized Deep Neural Network

* Deepest model with six hidden layers, each containing 96 neurons.
* Used ReLU activation for hidden layers and sigmoid activation for the output layer.
* Trained using SGD optimizer with a lower learning rate of 0.0012 for stable convergence.
* Loss function: Binary Cross-Entropy.
* Achieved improved performance by fine-tuning the number of neurons, layers, and learning rate.

**2.6 Flask API Development for NIDS Deployment**

To enable real-world deployment of the **Network Intrusion Detection System (NIDS)**, the trained **machine learning (ML)** and **deep learning (DL)** models were encapsulated within a **RESTful API** using **Flask**, a lightweight and flexible web framework for Python. Flask provides the necessary infrastructure to expose the models as web services, allowing them to receive incoming network traffic data, process it, and return predictions on potential intrusions in real time.

This architecture facilitates seamless communication between the NIDS and external applications or monitoring systems, enabling real-time intrusion detection and proactive security measures. Additionally, Flask's modular design supports scalability and integration with cloud-based platforms, enhancing the system's robustness and reliability in production environments. By deploying the NIDS as a Flask API, it is equipped to handle HTTP requests efficiently, making it suitable for high-throughput network monitoring and scalable distributed applications.

### **Flask API Development Process**

#### Step 1: Setting Up the Environment

To initiate the development of the **Flask API**, a dedicated **Python environment** was configured with all necessary dependencies to ensure seamless application deployment. A requirements.txt file was generated to systematically manage these dependencies, enabling streamlined installation and version control. This approach enhances reproducibility and simplifies environment setup for future deployments.

The file typically includes:

* Flask (for building the API)
* joblib (for serializing/deserializing traditional ML models)
* TensorFlow/Keras (for deep learning models)
* Other libraries like NumPy, pandas, or scikit-learn for data processing.

Subsequently, the required dependencies were installed by executing the following command:

pip install -r requirements.txt

Furthermore, the trained **Machine Learning (ML)** models were serialized using **Joblib**, while the **Deep Learning (DL)** models were saved employing **TensorFlow/Keras model serialization techniques**. This approach optimizes model storage and facilitates efficient loading during API execution, enabling seamless and rapid prediction generation.

#### Step 2: Creating the Flask Application

The Flask application was architected to expose the model's prediction capabilities as a **RESTful API service**. This design allows external systems to interact with the **Network Intrusion Detection System (NIDS)** by sending HTTP requests containing network traffic data, subsequently receiving real-time intrusion detection predictions as HTTP responses. This structured approach facilitates seamless integration into broader network security infrastructures and enables scalable, real-time monitoring capabilities.

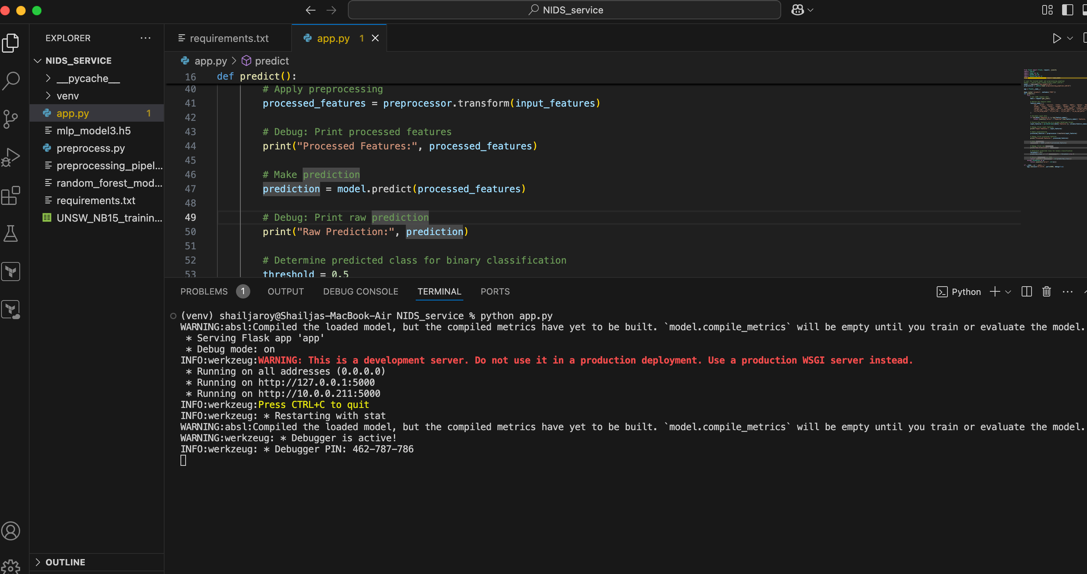
1. **Project Structure**
   * Created a new project folder: **nids\_service**
   * Inside the folder, added app.py to develop the API functionality using Python.
2. **Flask API Implementation**
   * The app.py script initializes a Flask application, loads the trained ML and DL models, and defines an endpoint to handle incoming requests.
   * The API receives network traffic data, processes it, and returns a prediction indicating whether the traffic is normal or an intrusion.

#### Step 3: Testing the Service Locally

Following the implementation of the **Flask API**, its functionality was rigorously tested through local HTTP requests to validate its performance and accuracy. The testing process involved the following methods:

* **cURL Command:** Utilized to send sample network traffic data to the API directly from the terminal, allowing for command-line-based interaction and response verification.

These tests ensured that the API effectively processed input data, seamlessly interacted with the underlying machine learning models, and generated accurate intrusion detection predictions.  
Figure 7 illustrates the API development environment configured for local testing and validation.

Figure 7. API development environment  
  


**3.0 RESULTS AND DISCUSSION**

Table 2 presents the evaluation results for the **UNSW-NB15 training dataset**, measured across key performance metrics.

Accuracy: This metric represents the ratio of correctly predicted instances (both true positives and true negatives) to the total number of predictions. It serves as a fundamental measure of the model's overall effectiveness in distinguishing between normal and malicious network traffic.

Precision: Precision measures how many of the predicted positive samples are actually positive. It tells you how accurate the positive predictions are.

Recall: Recall measures how many of the actual positive samples are correctly identified by the model. It tells you how sensitive the model is in identifying the positive class.

F1-Score: The F1-score is the harmonic mean of precision and recall. It balances both the concerns of precision and recall into a single metric. The F1 score is useful when you need to balance both false positives and false negatives.

Table 2: UNSW-NB15 Training Set Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **F1-score** | **Recall** | **Precision** |
| Decision Tree | 0.9275925 | 0.89(Class 0) 0.95(Class 1) | 0.96(Class 0) 0.91(Class 1) | 0.84(Class 0) 0.98(Class 1) |
| Random Forest | 0.9485573 | 0.92(Class 0) 0.96(Class 1) | 0.97(Class 0) 0.94(Class 1) | 0.88(Class 0) 0.98(Class 1) |
| Neural Network Model 1 | 0.9430 |  | 0.9760 | 0.9424 |
| Neural Network Model 2 | 0.9433 |  | 0.9763 | 0.9426 |
| Neural Network Model 3 | 0.9414 |  | 0.9796 | 0.9372 |

**Table 3** presents the evaluation results of the **validation set** for the **Neural Network models**. The performance metrics illustrate the model's capability to generalize its learning to unseen data, reflecting its predictive accuracy and robustness in identifying network intrusions.

Table 3: UNSW-NB15 Validation Set Results

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy** | **Recall** | **Precision** |
| Neural Network Model 1 | 0.9407 | 0.9677 | 0.9463 |
| Neural Network Model 2 | 0.9390 | 0.9883 | 0.9268 |
| Neural Network Model 3 | 0.9376 | 0.9772 | 0.9340 |

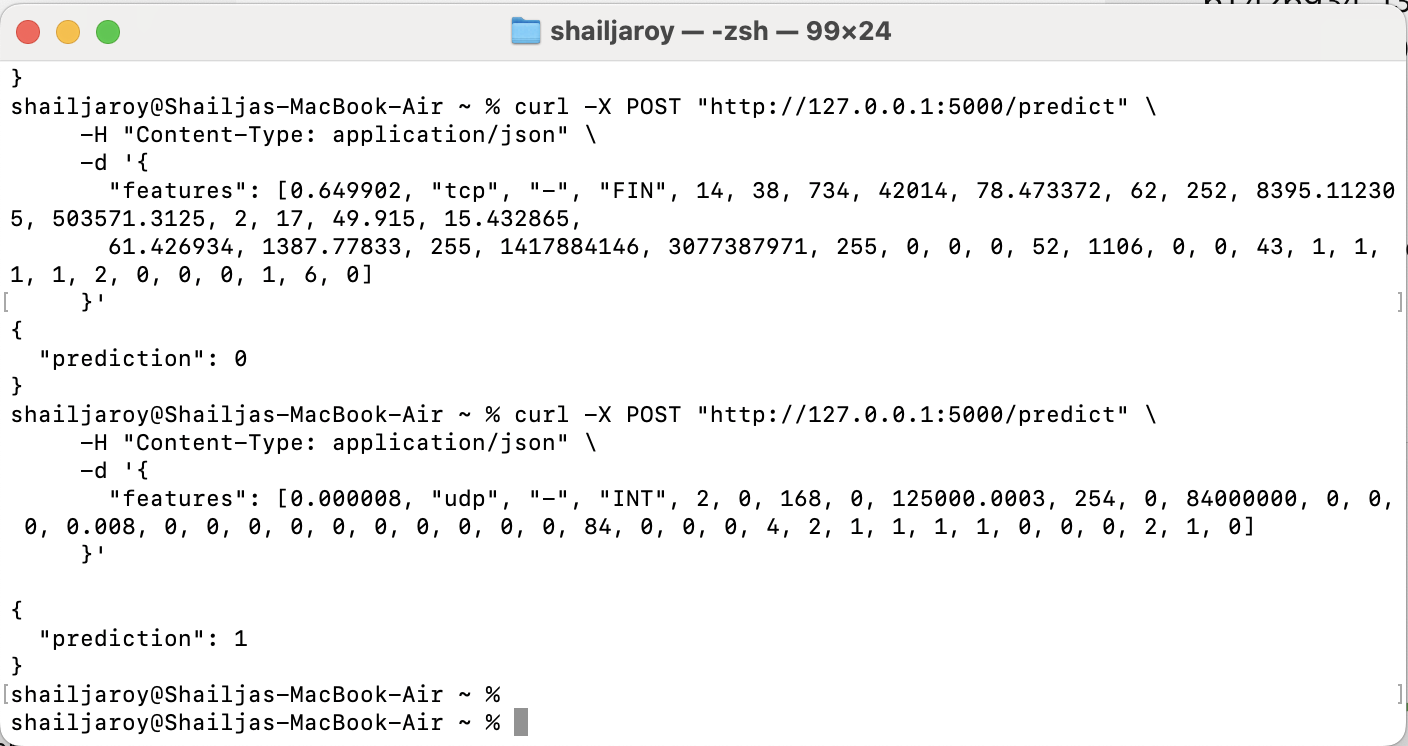
**Table 4** presents the evaluation results for the **UNSW-NB15 test set** across all implemented models. Among them, the **Neural Network 2 model** demonstrated the highest performance, achieving superior metrics in terms of accuracy, precision and recall indicating its effectiveness in detecting network intrusions within the test dataset.

Table 4: UNSW-NB15 Test Set Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **F1-score** | **Recall** | **Precision** |
| Decision Tree | 0.9122698 | 0.90(Class 0) 0.92(Class 1) | 0.87(Class 0) 0.95(Class 1) | 0.93(Class 0) 0.90(Class 1) |
| Random Forest | 0.9166909 | 0.90(Class 0) 0.93(Class 1) | 0.86(Class 0) 0.97(Class 1) | 0.95(Class 0) 0.89(Class 1) |
| Neural Network Model 1 | 0.9265 |  | 0.9368 | 0.9526 |
| Neural Network Model 2 | 0.9524 |  | 0.9896 | 0.9403 |
| Neural Network Model 3 | 0.9472 |  | 0.9803 | 0.9417 |

Upon completing the model training phase, the models were deployed as **service-oriented APIs**, enabling seamless integration across multiple platforms for real-time intrusion detection. This deployment architecture facilitates efficient communication with external applications, allowing for live predictions based on incoming network traffic data. **Figure 8** illustrates sample API requests along with the corresponding predicted responses, demonstrating the system's real-time decision-making capability.

Figure 8. Sample API requests and responses



4.0 **CONCLUSION AND DISCUSSION**

The primary objective of this project was to develop an effective Network Intrusion Detection System (NIDS) using the UNSW-NB15 dataset, which contains a comprehensive collection of network traffic data, including both normal and malicious activities. The dataset was pre-partitioned into training and testing sets, with 175,341 records for training and 82,332 records for testing. The dataset comprises 44 features, including 40 numerical and four categorical columns, making it a robust foundation for building and evaluating intrusion detection models. The target feature was a binary classification of network traffic behaviour, distinguishing between "Normal" and "Attack" activities.

To achieve the project's goals, a combination of Machine Learning (ML) and Deep Learning (DL) techniques was employed. Two supervised ML models—Decision Tree and Random Forest classifiers—were implemented for binary classification. Additionally, three Deep Learning models—were developed and evaluated. The models were trained and tested on the UNSW-NB15 dataset, and their performance was measured using key evaluation metrics such as accuracy, precision, recall, and F1-score.

Key Findings:

1. Machine Learning Models:
   * **Decision Tree Classifier**: Achieved an accuracy of 0.9276 on the training set and 0.9123 on the test set. The model demonstrated strong recall for Class 0 (Normal) at 0.96 and Class 1 (Attack) at 0.91, indicating its ability to correctly identify both normal and attack behaviors. However, the precision for Class 0 was relatively lower at 0.84, suggesting some false positives in classifying normal traffic as attacks.
   * **Random Forest Classifier**: Outperformed the Decision Tree with an accuracy of 0.9486 on the training set and 0.9167 on the test set. The Random Forest model showed improved precision and recall for both classes, with a recall of 0.97 for Class 0 and 0.94 for Class 1, and a precision of 0.88 for Class 0 and 0.98 for Class 1. This indicates that the Random Forest model was more robust in handling imbalanced data and reducing false positives.
2. Deep Learning Models:
   * **Neural Network Model 1**: Achieved an accuracy of 0.9430 on the training set and 0.9265 on the test set. The model demonstrated strong recall (0.9760) and precision (0.9424), indicating its effectiveness in detecting attacks while maintaining a low false positive rate.
   * **Neural Network Model 2**: Demonstrated the best overall performance, with an accuracy of 0.9433 on the training set, 0.9390 on the validation set, and 0.9524 on the test set. The model achieved a recall of 0.9883 on the validation set and 0.9896 on the test set, while maintaining a precision of 0.9268 and 0.9403, respectively. This reflects its superior ability to detect attacks accurately, while also minimizing false negatives.
   * **Neural Network Model 3**: Achieved an accuracy of 0.9414 on the training set and 0.9472 on the test set. The model demonstrated high recall (0.9796) and precision (0.9372), showing its strength in attack detection but falling slightly behind Model 2 in terms of test set accuracy and recall.
3. Validation and Test Set Performance:
   * The validation set results for the Neural Network models showed strong generalization capabilities, with **Model 2** achieving the highest recall of 0.9883 and a precision of 0.9268, reflecting its robustness against unseen data.
   * On the test set, **Neural Network Model 2** outperformed all other models with an accuracy of 0.9524, a recall of 0.9896, and a precision of 0.9403. This confirms that the optimized deep learning model is the most effective in detecting network intrusions with minimal false negatives and strong overall accuracy.
4. Flask API Deployment:
   * To make the NIDS accessible for real-time predictions, the best-performing models were deployed as a RESTful API using Flask. The API allows users to send network traffic data and receive predictions on whether the traffic is normal or an attack. The API was tested locally using the CURL command, ensuring its functionality and reliability.

#### **Future Work:**

1. **Real-Time Performance**: While the Flask API provides a framework for real-time predictions, further optimisation may be required to handle high-volume network traffic in real-world scenarios. Future work could involve deploying the API on cloud platforms with scalable resources to handle large-scale data. Also, the API can be containerised to be used across multiple environments and platforms.
2. **Multi-Class Classification**: This project focused on binary classification (Normal vs. Attack). Future work could extend the models to perform multi-class classification, identifying specific types of attacks (e.g., DoS, Exploits, Worms) to provide more detailed insights into network threats.

In conclusion, this project successfully developed and evaluated a Network Intrusion Detection System using the UNSW-NB15 dataset. The combination of Machine Learning and Deep Learning models demonstrated strong performance in detecting network intrusions, with the Model 2 emerging as the best-performing model. The deployment of the models as a Flask API further enhances their practical applicability, making them suitable for real-world use. While there are limitations, the results are highly encouraging, and future work can build on this foundation to further improve the system's performance and scalability. Overall, this project contributes to the ongoing efforts to enhance cybersecurity by providing an effective tool for detecting and mitigating network threats.

**REFERENCES**

1. Sharma, A., & Babbar, H. (2023). Enhancing IoT security: Machine learning-based network intrusion detection. 2023 3rd Asian Conference on Innovation in Technology (ASIANCON), Ravet IN, India, 1–6. <https://doi.org/10.1109/ASIANCON58793.2023.10269850>
2. Swarna Sugi, S. S., & Ratna, S. R. (2020). Investigation of machine learning techniques in intrusion detection system for IoT network. 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 1164–1167. <https://doi.org/10.1109/ICISS49785.2020.9315900>
3. P. A. A., Maryposonia, A., & P. V. S. (2023). An efficient network intrusion detection system for distributed networks using machine learning technique. 2023 7th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 1258–1263. <https://doi.org/10.1109/ICOEI56765.2023.10126055>
4. Almutairi, A. F., & Alshargabi, A. A. (2022). Using deep learning technique to protect internet network from intrusion in IoT environment. 2022 2nd International Conference on Emerging Smart Technologies and Applications (eSmarTA), Ibb, Yemen, 1–6. <https://doi.org/10.1109/eSmarTA56775.2022.9935467>
5. Halbouni, A. H., Gunawan, T. S., Halbouni, M., Assaig, F. A. A., Effendi, M. R., & Ismail, N. (2022). CNN-IDS: Convolutional neural network for network intrusion detection system. 2022 8th International Conference on Wireless and Telematics (ICWT), Yogyakarta, Indonesia, 1–4. <https://doi.org/10.1109/ICWT55831.2022.9935478>
6. Abraham, J. A., & Bindu, V. R. (2021). Intrusion detection and prevention in networks using machine learning and deep learning approaches: A review. 2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA), Coimbatore, India, 1–4. <https://doi.org/10.1109/ICAECA52838.2021.9675595>
7. Ikhwan, S., Purwanto, P., & Rochim, A. F. (2023). Comparison analysis of intrusion detection using deep learning in IoT networks. 2023 11th International Conference on Information and Communication Technology (ICoICT), Melaka, Malaysia, 339–344. <https://doi.org/10.1109/ICoICT58202.2023.10262603>
8. Ikhwan, S., Wibowo, A., & Warsito, B. (2022). Intrusion detection using deep neural network algorithm on the Internet of Things. 2022 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), Solo, Indonesia, 84–87. <https://doi.org/10.1109/COMNETSAT56033.2022.9994499>
9. Ch, S., & Kare, S. B. (2024). A comprehensive analysis of network intrusion detection in Internet of Things and wireless networks. 2024 International Conference on Data Science and Network Security (ICDSNS), Tiptur, India, 1–5. <https://doi.org/10.1109/ICDSNS62112.2024.10691047>
10. Liu, Z., Thapa, N., Shaver, A., Roy, K., Yuan, X., & Khorsandroo, S. (2020). Anomaly detection on IoT network intrusion using machine learning. 2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD), 1–5. <https://doi.org/10.1109/icABCD49160.2020.9183842>
11. Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, Australia, 1–6. <https://doi.org/10.1109/MilCIS.2015.7348942>
12. Zoghi, Z., & Serpen, G. (2024). UNSW-NB15 computer security dataset: Analysis through visualization. Security and Privacy, e331 (29 pp.). <https://doi.org/10.1002/spy2.331>
13. V. R, P. C. A., & V. M. (2024). A comprehensive analysis of intrusion detection system using machine learning and deep learning algorithms. 2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS), Hassan, India, 1–5. <https://doi.org/10.1109/IACIS61494.2024.10721636>
14. Muller, P. S., Madhavi, K., Durga Saile, K. N., Prasanthi, B., Jayaram, B., & Sathish, G. (2025). A hybrid deep learning framework for efficient network intrusion detection systems using data augmentation. 2025 3rd International Conference on Integrated Circuits and Communication Systems (ICICACS), Raichur, India, 1–5. <https://doi.org/10.1109/ICICACS65178.2025.10968940>
15. Abdelhamid, S., Aref, M., Hegazy, I., & Roushdy, M. (2021). A survey on learning-based intrusion detection systems for IoT networks. 2021 Tenth International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, Egypt, 278–288. <https://doi.org/10.1109/ICICIS52592.2021.9694226>
16. Wankhade, A. D., & Wagh, K. (2021). Machine learning techniques for defending cloud enabled IoT network. 2021 International Conference on Computational Intelligence and Computing Applications (ICCICA), Nagpur, India, 1–5. <https://doi.org/10.1109/ICCICA52458.2021.9697209>
17. Nugroho, E. P., Djatna, T., Sitanggang, I. S., Buono, A., & Hermadi, I. (2020). A review of intrusion detection system in IoT with machine learning approach: Current and future research. 2020 6th International Conference on Science in Information Technology (ICSITech), Palu, Indonesia, 138–143. <https://doi.org/10.1109/ICSITech49800.2020.9392075>
18. Guo, G. (2022). An intrusion detection system for the Internet of Things using machine learning models. 2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), Xi’an, China, 332–335. <https://doi.org/10.1109/ICBAIE56435.2022.9985800>
19. Majhi, B., & Prastavana. (2022). An improved intrusion detection system using BoT-IoT dataset. 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT), Indore, India, 488–492. <https://doi.org/10.1109/CSNT54456.2022.9787639>
20. L. B., Azath, H., Beulah David, D., Chandra Blessie, E., Jayapradha, A., & Sheeba Rani, S. (2021). BoT-IoT based denial of service detection with deep learning. 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 221–225. <https://doi.org/10.1109/I-SMAC52330.2021.9640789>
21. Varalakshmi, S., P. S. P., Y. V., V. P., Kavitha, V. R., & V. G. (2021). Design of IoT network using deep learning-based model for anomaly detection. 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 216–220. <https://doi.org/10.1109/I-SMAC52330.2021.9640700>
22. Ananthi, P., Nirmaladevi, K., & M. G. (2024). Deep learning based intrusion detection for IoT networks. 2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS), Gobichettipalayam, India, 1690–1694. <https://doi.org/10.1109/ICUIS64676.2024.10866629>

**Appendix A: Dataset Features and Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Name** | **Type** | **Description** |
| 1 | srcip | nominal | Source IP address |
| 2 | sport | integer | Source port number |
| 3 | dstip | nominal | Destination IP address |
| 4 | dsport | integer | Destination port number |
| 5 | proto | nominal | Transaction protocol |
| 6 | state | nominal | Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state) |
| 7 | dur | Float | Record total duration |
| 8 | sbytes | Integer | Source to destination transaction bytes |
| 9 | dbytes | Integer | Destination to source transaction bytes |
| 10 | sttl | Integer | Source to destination time to live value |
| 11 | dttl | Integer | Destination to source time to live value |
| 12 | sloss | Integer | Source packets retransmitted or dropped |
| 13 | dloss | Integer | Destination packets retransmitted or dropped |
| 14 | service | nominal | http, ftp, smtp, ssh, dns, ftp-data ,irc and  (-) if not much used service |
| 15 | Sload | Float | Source bits per second |
| 16 | Dload | Float | Destination bits per second |
| 17 | Spkts | integer | Source to destination packet count |
| 18 | Dpkts | integer | Destination to source packet count |
| 19 | swin | integer | Source TCP window advertisement value |
| 20 | dwin | integer | Destination TCP window advertisement value |
| 21 | stcpb | integer | Source TCP base sequence number |
| 22 | dtcpb | integer | Destination TCP base sequence number |
| 23 | smeansz | integer | Mean of the how packet size transmitted  by the src |
| 24 | dmeansz | integer | Mean of the how packet size transmitted  by the dst |
| 25 | trans\_depth | integer | Represents the pipelined depth into the  connection of http request/response transaction |
| 26 | res\_bdy\_len | integer | Actual uncompressed content size of the data  transferred from the serverís http service. |
| 27 | Sjit | Float | Source jitter (mSec) |
| 28 | Djit | Float | Destination jitter (mSec) |
| 29 | Stime | Timestamp | record start time |
| 30 | Ltime | Timestamp | record last time |
| 31 | Sintpkt | Float | Source interpacket arrival time (mSec) |
| 32 | Dintpkt | Float | Destination interpacket arrival time (mSec) |
| 33 | tcprtt | Float | TCP connection setup round-trip time,  the sum of ísynackí and íackdatí. |
| 34 | synack | Float | TCP connection setup time, the time between  the SYN and the SYN\_ACK packets. |
| 35 | ackdat | Float | TCP connection setup time, the time between  the SYN\_ACK and the ACK packets. |
| 36 | is\_sm\_ips\_ports | Binary | If source (1) and destination (3)IP addresses  equal and port numbers (2)(4) equal then,  this variable takes value 1 else 0 |
| 37 | ct\_state\_ttl | Integer | No. for each state (6) according to specific  range of values for source/destination time to  live (10) (11). |
| 38 | ct\_flw\_http\_mthd | Integer | No. of flows that has methods such  as Get and Post in http service. |
| 39 | is\_ftp\_login | Binary | If the ftp session is accessed by user  and password then 1 else 0. |
| 40 | ct\_ftp\_cmd | integer | No of flows that has a command in  ftp session. |
| 41 | ct\_srv\_src | integer | No. of connections that contain the same  service (14) and source address (1) in 100  connections according to the last time (26). |
| 42 | ct\_srv\_dst | integer | No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26). |
| 43 | ct\_dst\_ltm | integer | No. of connections of the same destination address (3) in 100 connections according to the last time (26). |
| 44 | ct\_src\_ ltm | integer | No. of connections of the same source address (1) in 100 connections according to the last time (26). |
| 45 | ct\_src\_dport\_ltm | integer | No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26). |
| 46 | ct\_dst\_sport\_ltm | integer | No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26). |
| 47 | ct\_dst\_src\_ltm | integer | No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26). |
| 48 | attack\_cat | nominal | The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms |
| 49 | Label | binary | 0 for normal and 1 for attack records |

## **Appendix B: Model Training and Hyperparameters**

The machine learning models were trained with the following hyperparameter configurations:

### Decision Tree Classifier:

* Criterion: Entropy
* Max Depth: 12
* Minimum Samples Split: 10
* Minimum Samples Leaf: 5
* Max Features: 0.8
* Cost complexity pruning: 0.001
* Class weight : Balanced

### Random Forest Classifier:

* Number of Estimators: 100
* Max Depth: 12
* Minimum Samples Split: 10
* Minimum Samples Leaf: 5
* Max Features: 0.8
* Class weight : Balanced

### Neural Network Model 1:

* Number of Hidden Layers: 3
* Neurons per Layer: 64
* Activation Function: ReLU
* Optimizer: Adam
* Loss Function: Binary Cross-Entropy
* Learning Rate: 0.00905127409782462

### Neural Network Model 2:

* Number of Hidden Layers: 6
* Neurons per Layer: 32
* Activation Function: ReLU
* Optimizer: SGD
* Loss Function: Binary Cross-Entropy
* Learning Rate: 0.00905127409782462

### Neural Network Model 3:

* Number of Hidden Layers: 6
* Neurons per Layer: 96
* Activation Function: ReLU
* Optimizer: SGD
* Loss Function: Binary Cross-Entropy
* Learning Rate: 0.0012482904754698163

## **Appendix C: Flask API Deployment Instructions**

To deploy the Flask API for real-time intrusion detection, follow these steps:

1. **Install Dependencies**

pip install flask numpy pandas joblib tensorflow

1. **Run the API**

Python app.py

1. **Example API Request**

curl -X POST "http://127.0.0.1:5000/predict" \

-H "Content-Type: application/json" \

-d '{

"features": [15.40279, "tcp", "-", "REQ", 14, 0, 630, 0, 0.844003, 254, 0, 303.841064, 0, 13, 0, 1184.83, 0, 2188.4775, 0, 255, 0, 0, 0, 0, 0, 0, 45, 0, 0, 0, 18, 6, 9, 3, 3, 17, 0, 0, 0, 10, 17, 0]

}'

1. **Expected Response Format**

{

"prediction": 0

}