جامعة آل البيت
AL al-BAYT UNIVERSITY

A Comprehensive Study of Intrusion Detection Systems for IoT Networks Using Machine Learning

Prepared by:

Rama Ibrahim Alamaireh: 2100906171
Shahd Abdallah Al-Jamal: 2100906026
Sewar Ismail Yaqoup: 2100906205

Supervisor:
Dr. Amjad Qtaish

Submitted in Partial Fulfillment of the Requirements for bachelor's degree in cyber security

Prince Al Hussein bin Abdullah Faculty of Information Technology
Al al-Bayt University

Al-Mafraq- Jordan

29/05/2025

Acknowledgment

We would like to express our sincere gratitude to Dr. Amjad Qtaish for his continuous support and dedicated follow-up throughout our graduation project in Cyber Security. His invaluable guidance and extensive expertise, particularly in the integration of Artificial Intelligence with Cyber Security, have played a crucial role in shaping our project.

Dr. Amjad's deep knowledge in both fields has greatly contributed to our understanding, allowing us to effectively apply AI techniques in our work. His ability to simplify complex concepts and provide insightful direction has been instrumental in our progress. His patience, commitment, and encouragement have motivated us to push our limits and achieve the best possible results.

We deeply appreciate the knowledge and experience Dr. Amjad has shared with us, and we are truly grateful for his continuous support.

We also extend our sincere thanks to our families, colleagues, and friends for their endless support, encouragement, and valuable discussions that contributed to our work. This journey would not have been the same without each one of them.

Declaration

We, Rama Alamaireh & Shahd Aljamal & Sewar Ismail, hereby declare that the project titled, **A Comprehensive Study of Intrusion Detection Systems for IoT Networks Using Machine Learning-Based**, submitted in partial fulfilment of the requirements for the degree of bachelor's in cyber security is our original work.

We confirm that all the information, data, and sources used in this project have been appropriately acknowledged and referenced. Any contributions or assistance received from others during this project have been duly acknowledged.

We further declare that this project has not been previously submitted for any other degree or examination in any other institution.

We take full responsibility for the content and accuracy of this project, and I understand that any false statement or omission of information may result in disciplinary action, including the cancellation of my degree.

Signed: Rama Alamaireh, **Shahd Aljamal**, **Sewar Ismail**

Student No.: 2100906171, 2100906026, 2100906205.

Date: 29-05-2025.

Abstract

An Intrusion Detection System (IDS) is a fundamental cybersecurity component designed to monitor network traffic and identify potential threats. On the Internet of Things (IoT) context, traditional IDS approaches face limitations due to the constrained nature of IoT devices and the dynamic, heterogeneous nature of network traffic. These challenges highlight the need for lightweight, accurate, and scalable detection solutions.

This study presents a comprehensive evaluation of Machine Learning-based IDS approaches tailored to IoT environments. The recently released CICIoT2023 dataset is used to ensure a realistic and diverse representation of modern IoT network behavior and attack vectors. Several supervised learning algorithms are applied, including Logistic Regression, Support Vector Machine, K-Nearest Neighbors, Decision Tree, Random Forest, and XGBoost.

Key preprocessing steps such as label encoding, feature scaling, and handling class imbalance (via SMOTE and class weight adjustment) are implemented to enhance model performance. Emphasis is placed on multiclass classification to enable the detection of a wide range of attack types beyond traditional binary categorization.

Our study offers a practical comparison of model performance using key evaluation metrics, providing insights into their strengths and suitability for real-world IoT applications. The findings serve as a valuable reference for designing resource-efficient and effective IDS solutions in modern IoT networks.

Table of Contents

List of figures

List of Tables

List of Equation

List of Abbreviations and Acronyms

item	Full Term
AI	Artificial Intelligence
ANN	Artificial Neural Network
CIC	Canadian Institute for Cybersecurity
CSV	Comma-Separated Values
DDoS	Distributed Denial of Service
DL	Deep Learning
DoS	Denial of Service
DT	Decision Tree
FN	False Negative
FP	False Positive
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IoT	Internet of Things
KNN	k-Nearest Neighbors
LR	Logistic Regression
ML	Machine Learning
PCA	Principal Component Analysis
RF	Random Forest
RBF	Radial Basis Function
SIEM	Security Information and Event Management

item	Full Term
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine
TP	True Positive
TN	True Negative
XGBoost	Extreme Gradient Boosting

CHAPTER 1: INTRODUCTION

The Internet of Things (IoT) has introduced new opportunities for smart automation and interconnectivity across various sectors, including healthcare, industry, and smart cities. However, this advancement has also increased cybersecurity exposure, requiring more intelligent and adaptive defense strategies such as Intrusion Detection Systems (IDS) [1], [2].

Many IoT devices lack built-in security mechanisms, making them vulnerable to a wide range of cyberattacks, including malware infections, data breaches, and DDoS attacks. These threats can severely impact data integrity, user privacy, and service availability. Traditional IDS solutions, which often rely on static rule sets or known signatures, struggle to scale effectively in IoT environments and are prone to high false alarm rates. Recent advances in AI, particularly ML, offer more dynamic and efficient alternatives by enabling IDSs to detect complex and evolving threats based on behavioral patterns [2], [5].

1.1 Intrusion Detection System

An IDS is a device or software application that monitors a network for malicious activity or policy violations [1]. Any detected threat or violation is typically reported or collected centrally using a Security Information and Event Management (SIEM) system. Some IDSs can also respond to detected intrusions in real time, in which case they are referred to as Intrusion Prevention Systems (IPS) [6].

In the data gathering component, input data from all parts of an IoT system is collected and analyzed to identify patterns of benign interaction, enabling early detection of malicious behavior. The analysis module can be implemented using various techniques; however, Machine Learning (ML)-based methods have become increasingly dominant due to their ability to learn both normal and anomalous behaviors based on device interactions in IoT environments.

Intrusion Detection Systems can be categorized based on their deployment and detection methodology, such as Host-based IDS (HIDS), Network-based IDS (NIDS), Signature-based, and Anomaly-based systems. However, this study does not focus on a specific IDS type but rather evaluates ML-based techniques applicable across IDS implementations in general IoT scenarios.

Furthermore, ML methods can predict unknown attacks, which are usually different from earlier attacks, because ML/DL methods can predict future unknown attacks intelligently through learning from existing legitimate samples. Figure 1.1 shows the components of typical IDS based on ML methods [1].

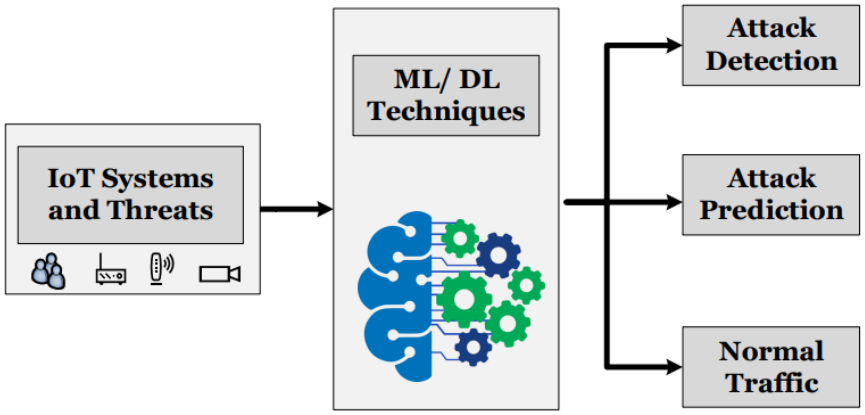


Figure 1.1: Function of ML/DL Based IDS for IoT system.

1.2 Role of Machine Learning in Intrusion Detection Systems

ML is a subfield of Artificial Intelligence (AI) that focuses on teaching systems to learn patterns from data and improve automatically over time without being directly programmed. This concept has become especially valuable in the development of IDS, particularly in the context of IoT.

Traditional IDS approaches often rely on fixed rules or known attack signatures, which limit their ability to detect new or changing threats. In contrast, ML models can analyze previous network traffic, learn from it, and identify abnormal behavior, even if the specific type of attack has not been seen before. This makes ML especially useful in IoT environments, where devices are diverse and generate large amounts of data that require constant monitoring.

In this study, several supervised learning algorithms are explored to classify network behavior. These include Logistic Regression, Support Vector Machine (SVM), k-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), and XGBoost. Each of these techniques offers a different approach to understanding the data and detecting possible threats.

By integrating ML into IDSs, it becomes possible to reduce false alerts and improve how the system reacts to real attacks. Also, by carefully selecting which features of the data are most important, the models can be made faster and more efficient, an important factor when working with IoT devices that have limited processing power.

In summary, Machine Learning helps make IDSs smarter and more responsive. It enables systems to adapt to new situations and handle threats more effectively, making it a key component in improving cybersecurity for IoT systems.

This study builds on these developments by conducting a comprehensive evaluation of multiple ML algorithms applied to IoT network traffic, aiming to identify effective and resource-efficient approaches to intrusion detection tailored to the unique constraints of IoT systems.

1.3 Problem Statement

The rapid growth of the IoT has transformed modern digital ecosystems, enabling real-time communication between billions of interconnected devices across sectors such as healthcare, transportation, industry, and smart cities. However, this expansion has also introduced significant cybersecurity challenges. Many IoT devices are resource-constrained, lack built-in security mechanisms, and are highly vulnerable to cyberattacks, including malware infections, unauthorized access, and DDoS attacks.

Traditional IDS, which rely on fixed rule sets and predefined signatures, have shown limited effectiveness in securing IoT environments. They struggle to detect zero-day attacks, often produce high false positive rates, and are computationally intensive, making them unsuitable for lightweight IoT infrastructures.

In response to these limitations, ML has emerged as a promising approach, enabling adaptive and intelligent detection of malicious patterns in network traffic. However, existing ML-based IDS solutions continue to face challenges such as poor scalability [2], [4], insufficient multiclass classification, and limited applicability to real-world IoT constraints.

This study aims to investigate and evaluate various ML-based IDS techniques to better understand their strengths, limitations, and practical applicability in the context of IoT network security.

1.4 Study Objectives

This Study aims to conduct a comprehensive study of ML-based IDS in the context of IoT networks. The goal is to evaluate the effectiveness of various ML algorithms in detecting and classifying different types of cyberattacks, while also assessing their suitability for deployment in resource-constrained IoT environments.

To achieve this, the study adopts modern data science practices and utilizes the CICIoT2023 dataset, which provides a realistic and diverse representation of IoT network traffic. Key preprocessing steps such as label encoding, normalization, and class balancing (using SMOTE and class weight adjustment) are applied to enhance model performance.

The main objectives of our study are as follows:

1. Evaluate and compare the performance of multiple ML classifiers in detecting IoT-related attacks.
2. Explore the impact of different data balancing techniques on classification accuracy and model behavior.
3. Emphasize multiclass classification to distinguish between various attack types, not just binary detection.
4. Analyze the resource efficiency of selected models to assess their feasibility for deployment on IoT devices.
5. Identify the most effective and adaptable ML model for real-world intrusion detection in IoT environments.

By testing these models under semi-realistic offline conditions, the study provides practical insights into their applicability and contributes to the development of resilient IDS solutions for IoT systems.

CHAPTER 2: Related Work

2.1 Overview of Intrusion Detection in IoT

The connected devices ecosystem has been adopted in multiple fields like healthcare, smart homes, industrial systems, and transport owing to the growth of the Internet of Things (IoT). Although it has some benefits, IoT poses serious security challenges due to its highly distributed architecture with resource-constrained devices and diverse communication protocols. Furthermore, IoT environments are susceptible to a herculean number of security threats, ranging from denial of service (DoS) and spoofing to more sophisticated data exfiltration and botnet infections.

In IoT environments, conventional security approaches like firewalls coupled with signature-based IDSs do not work efficiently. The major reason for this is the inability to manage the traffic patterns in IoT networks or identify unknown (zero day) threats. This necessitates intelligent and adaptive mechanisms for real-time intrusion detection that function well even with low hardware resources.

These issues have triggered the development of Machine Learning (ML)-based IDS, which promise to efficiently address these concerns. With powerful ML algorithms at hand for network traffic data analysis, pattern recognition, and anomaly detection, breaches are growing less common.

The efficiency of an ML-based IDS (Intrusion Detection Systems) relies heavily on the selected algorithm, the training data's quality, and how well it can adapt to novel attack vectors. This has led to evaluating and benchmarking different ML models for Intrusion detection in IoT environments to become a hot topic in cybersecurity.

2.2 Overview of Prior Research

In their work, these authors analyzed the usage of a few machine learning methods on IDS implemented for traditional and IoT networks. The study focused on the comparison of supervised learning algorithms: SVM, DT, KNN, and ensemble methods like RF . The authors pointed out that overall, RF tended to provide better accuracy and stability relative to other detectors on numerous attack detection tasks ordered by the NSL-KDD and CICIDS2017 datasets.

Also discussed in the paper are the difficulties posed by the impact of an imbalanced data and high false positive rates on standard ML techniques. It argued the need for more appropriate feature selection and processing techniques to improve the effectiveness of the IDS. The authors did conclude, however, that although machine learning techniques provide promising outcomes in greatly automating intrusion detection processes, further research is required to optimize them for the constrained computing capabilities of edge devices on the Internet of Things.

This review serves as the basis of our work in which we implement six ML models LR, SVM, KNN, DT, RF, XGBoost on a newer subset dataset CIC IoT 2023 to address the problem of ever-

changing cyber threats in IoT networks in conjunction with NSL-KDD and CICIDS2017 datasets Thaseen and Kumar 2020.

A more recent systematic review conducted by Jamshidia et al. 2025 thoroughly assessed the implementation of Deep Reinforcement Learning (DRL) based techniques for intrusion detection systems in IoT networks. The authors studied numerous architectures and frameworks focusing on enhancing detection in resource-limited and dynamic IoT environments and highlighted that DRL techniques can effectively learn and adapt to complex attack patterns without needing extensive labeled datasets, which is a hurdle most traditional supervised techniques face.

In addition, the review discusses the real-world operational constraints of IoT with DRL such as algorithmic edge cases, convergence time, and recommends optimizing those algorithms directly for edge devices. This research complements the ML-based IDS approaches implemented in this project by investigating advanced adaptive strategies aimed at enhancing detection precision and speed in next-gen IoT security.

An astonishingly innovative intrusion detection algorithm to cater for the intricacies of IoT systems using Adversarial Reinforcement Learning (ARL), has been proposed by Mahjoub et al. (2024) and integrates a lean neural net classifier alongside a policy function constructed with sophisticated reinforcement learning architecture. This setup facilitates

adaptive resilience to shifting network behavior which improves malicious traffic detection and classification. Leveraging the Bot-IoT dataset, the proposed model showcased unmatched performance compared to established techniques in detection accuracy and adaptability. This study highlighted the still untapped potential of ARL toward building advanced responsive IDS implementations for the era of IoT networks.

An IoT Intrusion Detection System utilizing KNN, DT, RF, Gradient Boosting and AdaBoost as an ensemble was developed by Mahmud et al. (2024). They incorporated feature selection methods to improve the effectiveness of their models. The evaluation of the models on an IoT network data set revealed Random Forest achieved the highest detection accuracy (99.39%), while the lowest, KNN (94.84%). The authors focused on the ability of traditional machine learning approaches.

Assorted attack vectors in IoT contexts. But they highlighted how effective feature engineering is in resolving issues like dataset imbalance and false detections. This reinforces the underserved focus on traditional ML algorithms within IoT IDS and adds to our study that applies various classifiers on the newest CIC IoT 2023 dataset.

Ashraf et al. (2025) constructed an INIDS for IoT using the BoT–IoT dataset and comparative analysis of seven ML classifiers, including Logistic Regression (LR), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), Naïve Bayes, and Artificial Neural Networks (ANN). They found that Random Forest yielded the highest classification accuracy of 99.2%, with Naïve Bayes next at 98.8%. Algorithm selection, real-time responsiveness, feature extraction, and overall system performance were among the primary focus of the authors. While the study looked at ANN, the attention was primarily on conventional ML. This is the purpose of this research which applies a wide range of machine learning models,

but no deep learning, to the CIC IoT 2023 dataset to enhance relevance for thin-client IoT environments.

In their comprehensive paper [1], Vanin et al. reviewed a broad range of IDS techniques that integrate AI and ML methods. While the study does not specifically focus on IoT networks, it presents a detailed overview of general IDS frameworks and challenges such as high false positive rates, scalability, and the inability of signature-based systems to detect zero-day attacks.

The paper discusses various ML techniques used in IDS, including supervised methods like Decision Trees, Random Forest, Support Vector Machine (SVM), and Naive Bayes, as well as unsupervised and semi-supervised approaches. It also highlights commonly used benchmark datasets such as NSL-KDD, KDDCup99, and UNSW-NB15, many of which are not IoT-specific but have laid the foundation for evaluating IDS models.

Although these datasets and models are more general-purpose, the findings helped shape our evaluation criteria, especially in terms of model selection, metric analysis (e.g., accuracy, recall, F1-score), and the importance of designing resource-efficient IDS solutions. These general insights were adapted and extended to suit the specific challenges of IoT-based IDS in our study

2.3 Comparative Analysis of Techniques

To better understand the strengths and weaknesses of different machine learning approaches used in IoT intrusion detection systems (IDS), a comparative analysis of selected recent studies is presented in the following table:

Table 1 Comparative Analysis of Previous Studies on ML-based IDS for IoT:

Ref	Authors (Year)	Used Dataset	Algorithms	Accuracy Best	Finding
[1]	Thaseen & Kumar (2020)	NSL-KDD, CICIDS2017	SVM, DT, KNN, RF	~99% (RF)	Focused on conventional datasets; emphasized RF's stability
[2]	Jamshidia et al. (2025)	Various (Review)	Deep RL models	N/A	Systematic review; mainly deep RL; limited to conceptual comparison
[3]	Mahjoub et al. (2024)	Simulated IoT	Deep RL	~98.3%	Designed adversarial RL for smart home detection
[4]	Mahmud et al. (2024)	IoT dataset	KNN, DT, RF, GB, AdaBoost	99.39% (RF)	Traditional ML focus; emphasized RF's superiority
[5]	Ashraf et al. (2025)	BoT–IoT	LR, SVM, KNN, DT, RF, NB, ANN	99.2% (RF)	Comprehensive ML comparison: ANN briefly included
[6]	Vanin et al. (2022)	NSL-KDD, KDDCup99, UNSW-NB15	DT, RF, SVM, NB, Unsupervised/Semi-supervised	N/A	General IDS focus; highlighted false positives, scalability; foundational for ML-based IDS

2.4 Limitations of prior studies

While previous research illustrates the usage of machine learning techniques for intrusion detection, there are still several gaps that remain unmapped. First, many searched works used obsolete or arbitrary datasets such as NSL-KDD or CICIDS2017 which fail to capture the IoT specific threats, and their datasets change over time. Deep Learning models, regardless of having strong performance, face difficulties in practical implementation because of their computational complexity and high resource consumption which makes real time execution on IoT devices extremely challenging.

In addition, the diverse nature of IoT environments with multitude of devices having different communication protocols as well as limited power and processing capability pose a lot of challenges. This scenario hinders the development of streamlined and efficient intrusion detection frameworks specifically designed for IoT systems.

Also, there are limited analyses of recently created, IoT oriented datasets with regards to the application of classical machine learning frameworks. There is an increasing demand for models which can effortlessly function within the strict limitations posed by IoT devices while being lightweight, efficient and precise.

Moreover, apart from achieving high accuracy of detection, practical implementation must incorporate low adaptability to changing threats while achieving low performance penalties and minimizing disruptions in network operations.

Logistic This gap will be addressed by evaluating six traditional machine learning algorithms using the CIC IoT Regression, SVM, KNN, Decision Tree, Random Forest, and XGBoost 2023 dataset for study purposes. This dataset encompasses contemporary IoT network environments and attack vectors, incorporating emerging and highly sophisticated threats. This study aims to determine the best practical model for real-world IoT intrusion detection by evaluating accuracy, computational efficiency, and model adaptability alongside the standard evaluation metrics for all models.

Additionally, while general IDS studies such as Vanin et al. (2022) provide valuable insights into the strengths and limitations of various ML techniques, their focus on legacy datasets and non-IoT environments highlights the need for more targeted evaluations on modern IoT-specific datasets like CIC IoT 2023.

CHAPTER 3: Methodology

This chapter outlines the methodological framework followed in this study to investigate the applicability and performance of various machine learning techniques for intrusion detection in IoT networks. The methodology was designed to reflect practical, real-world scenarios while maintaining a rigorous experimental approach.

The process is structured into several key stages: understanding the dataset, preparing and processing the data, handling class imbalance, building machine learning models, and evaluating their performance.

The CICIoT2023 dataset chosen for its relevance and richness serves as the foundation for this analysis. Each stage of the workflow has been carefully executed to ensure accuracy, efficiency, and relevance to the constraints and characteristics of IoT environments. A visual overview of the methodology is illustrated in Figure 3.1, which presents the complete lifecycle from data exploration to model evaluation.

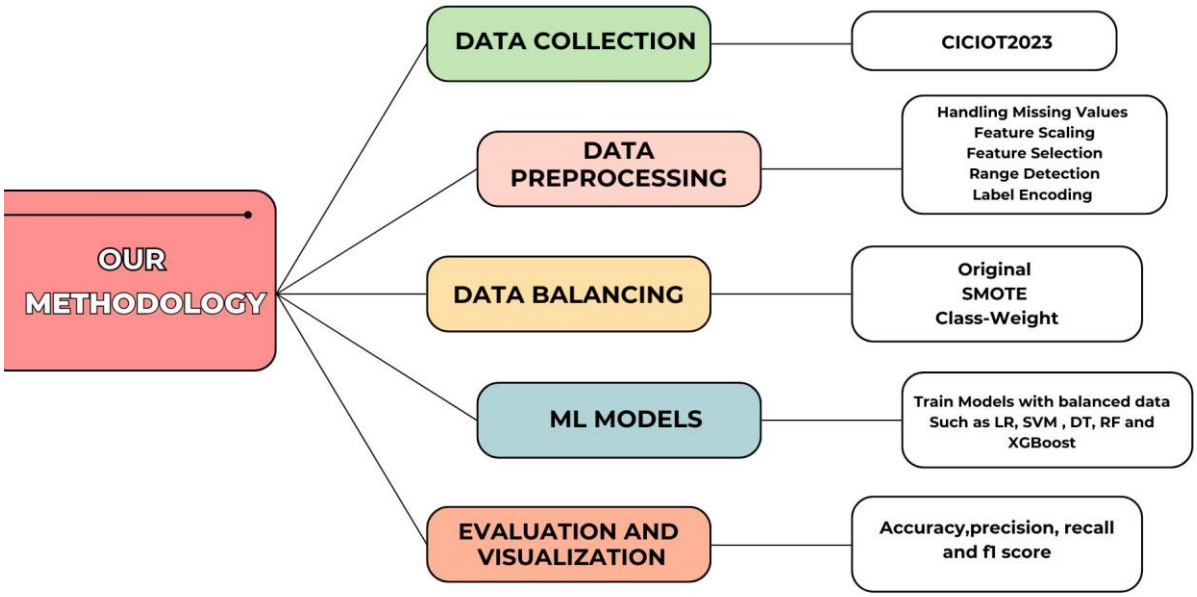


Figure 2.1: Our Proposed Methodology Workflow

3.1 Data Collection

The dataset used in this project is CICIoT2023, developed by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick. It captures realistic IoT network traffic from 105 devices, with both benign and malicious activity. A total of 33 different attacks were simulated, representing a broad range of cyber threats typically faced in IoT environments. The dataset is designed to support research and development of intrusion detection systems by providing labeled traffic data with a high degree of realism.

3.1.1 Data Loading

The dataset used in this project was obtained from Kaggle, where the CICIoT2023 dataset is available in the form of 168 separate CSV files. Each file corresponds to traffic captured from different IoT attack scenarios and devices.

To streamline the initial development and reduce computational complexity, the project initially focused on a single file **number 92**, which contains **211,835 records**. This file was selected due to its manageable size, enabling faster experimentation while still preserving the dataset's core structure and characteristics.

The data was loaded using *Google Colab* and the *Pandas* library in Python. The *read_csv()* function was used to import the data into a structured DataFrame, serving as the basis for subsequent preprocessing tasks.

3.1.2 Dataset Description

The **CICIoT2023** dataset offers a comprehensive simulation of IoT attack scenarios. It comprises over **46 million** network traffic records, each annotated with **47 features** that describe various attributes of the traffic flow, such as protocol types, packet sizes, and timing characteristics.

The dataset includes **33 distinct attacks** categorized into **seven classes**:

- **Distributed Denial of Service (DDoS)**
- **Denial of Service (DoS)**
- **Reconnaissance**
- **Web-based attacks**
- **Brute-force attacks**
- **Spoofing**
- **Mirai botnet attacks**

These seven attack categories simulate a wide range of threats in IoT networks:

- **DDoS / DoS**: Aim to overwhelm a target device or service with a massive amount of traffic, rendering it unavailable to legitimate users.
- **Reconnaissance**: Involves scanning or probing networks to gather information about services, ports, and vulnerabilities.
- **Web-based attacks**: Target vulnerabilities in HTTP services, such as SQL injection or file inclusion exploits.
- **Brute-force attacks**: Attempt to gain unauthorized access by systematically trying different combinations of credentials (all possible credentials).

- **Spoofing**: Involves falsifying identity information, such as IP addresses or MAC addresses, to mislead network defenses.
- **Mirai botnet attacks**: Malware specifically designed to exploit weakly secured IoT devices and turn them into bots for coordinated attacks.

3.1.3 Data Structure

The dataset is structured in CSV format, with each row representing a network flow and columns corresponding to the extracted features. The 47 features encompass various aspects of the traffic flow, including:

- **Flow-based features**: e.g., Flow Duration, Total Fwd Packets, Total Backward Packets
- **Packet-based features**: e.g., Packet Length Mean, Packet Length Std
- **Time-based features**: e.g., Flow IAT Mean, Flow IAT Std
- **Flag-based features**: e.g., PSH Flag Count, URG Flag Count
- **Header-based features**: e.g., Fwd Header Length, Bwd Header Length

Also, the dataset contains a **label** column, which serves as the target variable for classification. Each row is labeled either as **benign** or with **the name of a specific attack type**. During preprocessing, this categorical label was encoded into numeric values for use in machine learning models.

A snapshot of the dataset's actual structure is shown in Figure 7 using *df.head()*, which demonstrates selected features, and the corresponding label used for classification.

Also, it is important to note that the CICIoT2023 dataset is highly imbalanced, with approximately **97.6% of records labeled as attack traffic** and only **2.4% labeled as benign**. This severe imbalance can hinder the performance of machine learning classifiers, especially in detecting minority class samples, potentially leading to a high rate of false negatives. A detailed breakdown of the class distribution is presented in **Table 4.1 in the Results chapter**, and the handling of this imbalance is discussed further in the **Data Balancing section**.

3.2 Data Preprocessing

Preprocessing is a crucial step to ensure the dataset is clean, consistent, and suitable for training robust machine learning models. Several key preprocessing tasks were performed on the selected data file to enhance model performance and prevent biases.

- **Handling Missing Values:**
  The dataset contained some null values, which were handled by removing the corresponding records using the *dropna*() function. This approach was chosen to maintain data integrity without introducing bias from imputation.
- **Feature Selection:**
  The dataset included **47 numerical features**, each representing a distinct aspect of network traffic behavior. No features were dropped, as all were deemed potentially

relevant for intrusion detection. Additionally, the dataset did not include irrelevant fields such as IDs or textual descriptions that typically require removal.

- **Label Encoding:**
  The target variable (attack label) was originally in categorical text format, representing various attack types and benign traffic. To convert these labels into a numerical format suitable for supervised learning, **Label Encoding** was applied using *LabelEncoder()* from Scikit-learn. Each unique class was assigned a distinct integer value.

- **Feature Scaling:**
  To ensure uniformity across features and prevent dominance by attributes with larger numeric ranges, **Min-Max Scaling** was applied to normalize all feature values to the range [0, 1]. This step is especially important for distance-based algorithms like KNN and SVM.

- **Range Detection:**
  A preliminary range check was also performed to ensure that feature values were within expected intervals, and to detect any anomalous scale shifts before training. No abnormal values were found during this inspection.

These preprocessing steps ensured that the dataset was clean, normalized, and ready for training in a variety of machine learning models without introducing skewed bias due to data irregularities.

3.2.1 Data Balancing

One of the major challenges identified in the CICIoT2023 dataset is the severe class imbalance, with approximately **97.6%** of the samples labeled as attack traffic and only **2.4%** as benign. This imbalance can lead to biased model training, where classifiers tend to favor the majority class and perform poorly on the minority class, which is critical in intrusion detection systems.

To address this issue, two widely adopted strategies were implemented and compared:

- **SMOTE (Synthetic Minority Over-sampling Technique):**
  This technique generates synthetic examples of the minority class based on feature-space similarities. It aims to balance the dataset without duplicating existing samples.

- **Class Weight Adjustment:**
  This method adjusts the importance of each class during training, allowing the model to focus more on the minority class even with limited samples.

Both methods were applied across multiple supervised machine learning algorithms, including Logistic Regression, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree, and XGBoost. The performance of each model was evaluated using accuracy, precision, recall, and F1-score.

A detailed comparison of results showing how balancing techniques affected each algorithm's performance is presented in **Chapter 4 (Results).** This includes:

- Accuracy improvements across models using SMOTE and class weights.

- Comparative metrics for each technique applied in different classifiers.
- A visual representation comparing accuracy before and after balancing (see **Figure 4.1**).
- Summary tables with detailed metrics (see **Tables 4.2 to 4.5**).

The comparative results across all balancing strategies show that addressing class imbalance has a significant impact on model performance. **Class weight adjustment** proved to be a reliable method, often outperforming **SMOTE** while avoiding the introduction of synthetic data. Random Forest and XGBoost consistently delivered high accuracy and balanced performance across metrics, regardless of the balancing technique used.

These findings informed the decision to prioritize models and balancing strategies that maintain strong detection capabilities for both majority and minority classes. In the next phase, these selected models will be further tuned and evaluated as part of the Modeling stage.

3.3 Modeling

This section presents a detailed explanation of the six machine learning models applied in this study to detect cyber threats within IoT network traffic. Each model was selected for its unique characteristics in terms of learning strategy, decision-making, and suitability for high-dimensional datasets like CICIoT2023. The models include both linear and non-linear classifiers, as well as ensemble methods, to ensure a diverse and reliable performance comparison.

The following subsections explain each algorithm in depth, along with its configuration and training process in this project.

3.3.1 Logistic Regression (LR)

Logistic Regression is a statistical model used for classification tasks [10]. Unlike binary logistic regression, which classifies data into two categories, **Multinomial Logistic Regression** is used when the target variable involves more than two classes making it appropriate for our **multiclass attack detection** problem.

In our project, the model attempts to predict the **type of network traffic** (e.g., benign, DDoS, spoofing, etc.) based on 33 numerical features extracted from IoT traffic flows.

Mathematical Foundation:
Instead of using the **sigmoid function**, which is suited only for binary classification, Multinomial Logistic Regression uses the **SoftMax function** to compute the probability distribution over multiple classes.

Given a feature vector **x**, the probability of class **j** is computed as:

Equation 1 :Logistic Regression

$$P(y = j \mid x) = \frac{e^{w_j^T x + b_j}}{\sum_{k=1}^{K} e^{w_k^T x + b_k}}$$

Where:

- w_j is the weight vector for class j
- b_j is the bias term for class j
- K is the number of classes
- The denominator ensures that the output probabilities across all classes sum to 1

The predicted class is the one with the highest probability.

Why we Used It:
We included Multinomial Logistic Regression as a **baseline model**. It is fast, interpretable, and helps us understand whether simple linear decision boundaries are sufficient for separating different types of IoT traffic.

Training and Settings:
The model was implemented using Scikit-learn's LogisticRegression() class with multiclass='multinomial' and solver='lbfgs'. We trained it in all three dataset variations:

- Original (imbalanced)
- SMOTE-balanced
- Class-weight adjusted (using class weight='balanced')

Despite its simplicity, the model helped identify how linearly separable the classes were. However, it struggled to capture the complex relationships in the data, especially for minority classes.

Strengths:

- Computationally efficient
- Requires fewer resources suitable for IoT environments
- Easy to interpret (weights show feature importance)

Limitations:

- Poor performance on non-linear or overlapping classes
- Assumes linear separability between classes
- Sensitive to class imbalance and irrelevant features

To better understand the data distribution and assess the suitability of linear models, a Principal Component Analysis (PCA) was performed to project the high-dimensional feature space into two dimensions. many attack classes overlap significantly in the reduced 2D space, indicating that the data is not linearly separable. This visual evidence helps explain why Multinomial Logistic Regression struggled to differentiate between classes, especially when attack behaviors exhibit complex, non-linear boundaries. Consequently, more advanced models that can capture non-linear patterns such as Random Forest and XGBoost—are required for more effective classification.

3.3.2 Support Vector Machine (SVM)

Support Vector Machine is a powerful classifier that aims to find the optimal to separating hyperplanes between different classes. It's particularly effective in high-dimensional spaces and non-linear data [10].

Core Concept:
SVM looks for the decision boundary (hyperplane) that **maximizes the margin** between support vectors the nearest points from each class.

For non-linear data, SVM uses the **kernel trick** to map data into a higher-dimensional space where a linear separation is possible. In this project, we used the **Radial Basis Function (RBF)** kernel.

Mathematical Formulation:

Equation2 : SVM

Given a kernel function $K(x_i, x_j)$, SVM solves:

$$\text{minimize } \frac{1}{2}||w||^2 \quad \text{subject to } y_i(w \cdot x_i + b) \geq 1$$

Where:

- w is the normal vector of the hyperplane
- x_i is a training sample
- y_i is its class label (+1 or -1)

Why We Used It:
SVM is known for its strong performance in security and intrusion detection tasks. Its ability to model complex boundaries and small datasets makes it ideal for testing against class imbalance.

Training and Settings:
We used the SVC class in Scikit-learn with kernel='rbf'. Class balancing was addressed using class weight='balanced', which automatically adjusts the cost of misclassification for each class.

Strengths:

- Handles high-dimensional, non-linear data well
- Robust to overfitting

- Suitable for small and imbalanced datasets

Limitations:

- Computationally expensive on large datasets
- Sensitive to parameter tuning (C and gamma)
- Difficult to interpret results

3.3.3 K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a **lazy learning** algorithm that makes predictions based on the majority label among the closest training points in the feature space.

How It Works:
Given a test sample, KNN finds the closest k samples (based on a distance metric, usually Euclidean) and predicts the majority class among them.

Mathematical Intuition:
For a given point **x**, find **k** nearest neighbors using:

Equation3 : KNN

$$\text{distance}(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

Then classify based on the most frequent class among those neighbors.

Why We Used It:
KNN is simple to understand and implement. It provides a strong reference model, especially after SMOTE balancing (where minority class examples are synthetically generated).

Training and Settings:
We used Scikit-learn's KNeighborsClassifier with k=5. It was only tested on the original and SMOTE-balanced datasets (KNN doesn't support class weights). Min-Max scaling was critical due to its reliance on distance measures.

Strengths:

- Easy to implement and understand
- Non-parametric (makes no assumptions about data distribution)
- Naturally handles multiclass classification

Limitations:

- Computationally expensive at prediction time
- Sensitive to feature scaling and irrelevant features
- Doesn't perform well with imbalanced data

3.3.4 Decision Tree (DT)

A Decision Tree is a hierarchical structure where internal nodes represent feature conditions, branches represent outcomes, and leaves represent final class predictions [10].

How It Works:
It splits the dataset recursively based on the **feature that provides the highest information gain** (or lowest Gini impurity). This continues until each leaf contains only one class or another stopping condition is met.

Splitting Example:
If a feature like "Packet Length Mean" splits the data most effectively, the tree will branch based on its threshold.

Why We Used It:
Decision Trees are fast, interpretable, and a good choice for initial models. They also support multiclass classification and handle both categorical and numerical features well.

Training and Settings:
We used DecisionTreeClassifier with default settings. No pruning or depth limit was applied to observe the full tree behavior. It was tested across all data types.

Strengths:

- Easy to visualize and interpret
- Can handle both numerical and categorical data
- No need for feature scaling

Limitations:

- Prone to overfitting, especially with deep trees
- Sensitive to small data variations
- Performance depends heavily on balanced data

3.3.5 Random Forest (RF)

Random Forest is an **ensemble** of Decision Trees. Each tree is trained on a random subset of data and features, and their results are aggregated (usually by majority vote) to improve accuracy and robustness.

Why It Works:
By combining many trees and introducing randomness, Random Forest reduces overfitting and improves generalization.

Training Process:

- Random sampling with replacement (bootstrap) for each tree

- Random subset of features selected at each node split
- Final prediction is the majority vote from all trees

Why We Used It:
RF is widely recognized for its strong performance in intrusion detection. It works well even with imbalanced and noisy data and supports class weights for handling skewed distributions.

Training and Settings:
We trained Random Forest using 100 estimators (n_estimators=100) and class_weight='balanced'. It was evaluated on all three dataset variants.

Strengths:

- High accuracy and robustness
- Works well with large datasets
- Reduces overfitting through ensemble learning

Limitations:

- Less interpretable than a single Decision Tree
- Slower to train and test than simpler models
- May still be biased toward dominant classes without class balancing

3.3.6 XGBoost (Extreme Gradient Boosting)

XGBoost is an advanced, scalable gradient boosting algorithm that builds trees sequentially. Each new tree corrects errors from the previous one by minimizing a differentiable loss function.

How It Works:

- Build one tree at a time
- Each tree is optimized based on gradients of the loss function (similar to gradient descent)
- Includes regularization (L1 and L2) to avoid overfitting

Mathematical Insight:
For each round, the model tries to minimize:

Equation4 : XGBoost

$$\text{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \Omega(f_t)$$

Where:

- l is the loss function (e.g., softmax loss)
- Ω is the regularization term
- f_t is the newly added tree

Why We Used It:
XGBoost is considered state-of-the-art for structured data and is known for its **speed, regularization, and superior accuracy**. It's highly used in cybersecurity competitions and publications.

Training and Settings:
We used the default XGBoost classifier (XGBClassifier) from the XGBoost library. Although we did not perform deep tuning (due to time constraints), its default configuration yielded excellent performance on both SMOTE and class-weighted datasets.

Strengths:

- Very high accuracy and generalization
- Built-in handling of missing values
- Regularization prevents overfitting

Limitations:

- More complex to configure than other models
- Can be computationally expensive
- Harder to interpret than simpler models

3.3.6 Summary of Model Training Configuration

Table 2: Summary of Model Training Configuration

Algorithm	Normalization Used	Balancing Applied	Class Weight	Dataset Variants Used	Hyperparameter Tuning	Notes
Logistic Regression	MinMax Scaler	SMOTE, Class Weight	Yes	Original, SMOTE, Class Weighted	No	Multinomial LR used with softmax
Support Vector Machine	Standard Scaler	SMOTE, Class Weight	Yes	Original, SMOTE, Class Weighted	No	RBF kernel; strong after class weighting
K-Nearest Neighbors	MinMax Scaler	SMOTE only	No	Original, SMOTE	No	K=5; class weighting not supported
Decision Tree	None	SMOTE, Class Weight	Yes	Original, SMOTE, Class Weighted	No	Default settings; no pruning
Random Forest	None	SMOTE, Class Weight	Yes	Original, SMOTE, Class Weighted	No	100 estimators; very stable
XGBoost	None	SMOTE, Class Weight	Yes	SMOTE, Class Weighted	No	Default config; excellent performance

3.4 Evaluation and Visualization

Once the machine learning models were trained, their performance was rigorously evaluated to determine their effectiveness in detecting and classifying various types of attacks within the IoT network environment. Since the goal of this study was multiclass classification across an imbalanced dataset, a combination of qualitative and quantitative evaluation metrics was used to ensure a well-rounded understanding of each model's strengths and weaknesses.

3.4.1 Evaluation Metrics

To comprehensively assess the models, the following metrics were used:

- **Accuracy**: Measures the overall correctness of the model by dividing the number of correct predictions by the total number of predictions. While useful, accuracy alone can be misleading in imbalanced datasets.
- **Precision**: Evaluates how many of the predicted positive instances were correct. High precision indicates a low false positive rate.

$$\frac{TP}{TP + FP} = \text{Precision}$$

- **Recall (Sensitivity)**: Measures the ability of the model to identify all relevant instances. High recall indicates a low false negative rate

$$\frac{TP}{TP + FN} = \text{Recall}$$

- **F1-Score**: The harmonic meaning of precision and recall. It provides a balanced evaluation that is especially useful when the class distribution is uneven.

$$\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \cdot 2 = \text{F1-Score}$$

- **Confusion Matrix**: A matrix layout that summarizes the performance of a classification model by showing the true vs. predicted class distributions. It helps visualize misclassification patterns and evaluate per-class performance.

  These metrics were calculated for each model under all data balancing conditions to ensure consistent comparison across methods.

3.4.2 Evaluation Process

Each model was evaluated on a **held-out test set** that was not used during training. The dataset was divided using a 70/30 split, with 70% used for training and 30% reserved for testing. This split was applied uniformly across all balancing strategies to maintain experimental consistency.

The evaluation was conducted under three different data configurations:

- **Original Imbalanced Dataset**
- **SMOTE-Balanced Dataset**

- **Class Weight Adjusted Dataset**

For each scenario, the models were trained and then tested using the same evaluation metrics. This allowed a consistent comparison of how class balancing impacted model performance.

3.4.3 Comparative Analysis Approach

To enable a fair comparison, the same evaluation metrics were applied to all classifiers under each data balancing method. In addition to reporting accuracy, the focus was placed on **precision, recall, and F1-score** for the minority (benign) and attack classes. This was critical to measure how well each model could detect underrepresented classes and avoid biased predictions toward majority classes.

Furthermore, **confusion matrices** were plotted and analyzed in Chapter 4 to visually interpret how well each classifier was able to distinguish between the different types of attacks.

Chapter4 will provide a detailed presentation and discussion of all evaluation results, including comparative tables and visualizations of the metrics described above.

CHAPTER 4: Experimental Result and Discussion

4.1 Introduction

This chapter presents the results of applying various supervised Machine Learning algorithms to the CICIoT2023 dataset for intrusion detection in IoT networks. The evaluation covers different preprocessing and class balancing strategies, including the original imbalanced data, SMOTE-based oversampling, and class weight adjustment. Key performance metrics such as accuracy, precision, recall, and F1-score evaluate each model's effectiveness in identifying both majority and minority classes.

In addition to numerical results, this chapter highlights how class imbalance affected model performance and demonstrates the improvements achieved through balancing techniques. A comparative analysis is conducted to identify the most effective models in terms of accuracy and resource efficiency. These findings contribute directly to the project's goal of studying building a lightweight, practical, and effective ML-based IDS tailored to IoT environments.

4.2 The used Dataset (CIC IOT 2023)

A dataset is a structured collection of information, which can range from simple arrays to complex database tables. In this project, the **CICIoT2023** dataset is utilized a real-time and large-scale IoT attack dataset developed by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick.

The primary goal of this dataset is to provide a novel and extensive resource for fostering the development of security analytics applications in real-world IoT operations. To achieve this, 33 different cyberattacks were carried out within an IoT topology consisting of 105 devices. These attacks include **DDoS, DoS, Reconnaissance, Web-based attacks, Brute-force attacks, Spoofing**, and **Mirai botnet attacks**.

All attacks were launched by compromised IoT devices targeting other IoT devices, offering a realistic simulation of intra-IoT threats, The detailed categorization of executed attacks is shown in **figure 3**[4].

DDoS	ACK Fragmentation
	UDP Flood
	SlowLoris
	ICMP Flood
	RSTFIN Flood
	PSHACK Flood
	HTTP Flood
	UDP Fragmentation
	ICMP Fragmentation
	TCP Flood
	SYN Flood
	SynonymousIP Flood
Brute Force	Dictionary Brute Force
Spoofing	Arp Spoofing
	DNS Spoofing

DoS	TCP Flood
	HTTP Flood
	SYN Flood
	UDP Flood
Recon	Ping Sweep
	OS Scan
	Vulnerability Scan
	Port Scan
	Host Discovery
Web-Based	Sql Injection
	Command Injection
	Backdoor Malware
	Uploading Attack
	XSS
	Browser Hijacking
Mirai	GREIP Flood
	Greeth Flood
	UDPPlain

Figure 3 :Categories and Types of Attacks Executed in the CICIoT2023 Dataset

The dataset contains over **46 million** network traffic records, each annotated with **47 features** that describe various attributes of the traffic flow, such as protocol types, packet sizes, and timing characteristics. This rich feature set enables in-depth analysis and supports the application of machine learning techniques for intrusion detection.

By offering a comprehensive and diverse compilation of IoT traffic data, the CICIoT2023 dataset serves as a valuable resource for researchers and practitioners working to improve the security and resilience of IoT systems.

In the following chapters, we present how this dataset was preprocessed, analyzed, and used to train and evaluate a range of machine learning models, with the aim of assessing their effectiveness in detecting IoT-related cyber threats.

Also, about class distribution, the dataset used in this study exhibits a strong imbalance, with nearly 97.6% of records labeled as attack and only 2.4% as benign. This class skew poses significant challenges for training ML models. The distribution is shown in **Table 2.**

Table 2:Class Distribution in CICIoT2023 Dataset

Class	Number of Records	Percentage
Attack	~45,588,385	97.6%
Benign	~1,098,195	2.4%
Total	~46,686,580	100%

4.2.1 Programming Environment

The entire implementation and experimentation in this study were conducted using **Python**, one of the most widely adopted programming languages in the data science and machine learning communities. The experiments were performed in **Google Colab**, which provides a cloud-based Jupyter Notebook environment with GPU support and seamless integration with Python libraries.

The following key libraries were used:

- **Pandas** and **NumPy**: for data handling and preprocessing
- **Scikit-learn**: for implementing ML models, preprocessing techniques, and evaluation metrics
- **XGBoost**: for applying the gradient boosting model
- **Matplotlib** and **Seaborn**: for data visualization and plotting performance metrics

Python's flexibility, extensive ML ecosystem, and community support made it ideal for building and evaluating lightweight IDS models tailored to IoT environments.

4.3 The Used ML Models

The initial models were trained on the original unbalanced dataset. The results showed clear bias toward the majority class, particularly in precision and recall metrics.

The following tables illustrate the impact of different balancing techniques on model performance.

Table 3: Model Accuracy under Different Class Balancing Strategies

Algorithm	Original Accuracy	SMOTE Accuracy	Class Weight Accuracy	Discussion
LogisticReg	***17.86***	*09.90*	03.68	• Logistic Regression may not capture complex non-linear relationships as effectively as tree-based models like Decision Tree and Random Forest, leading to lower accuracy. • The model might have struggled to adjust to the minority class during training, especially with SMOTE and Class Weight adjustments, which could have led to overfitting or underfitting. • The dataset might have features that don't align well with the assumptions of Logistic Regression, such as the linearity and independence of features.
Random Forest	***99.07***	*99.34*	98.95	• Improved stability and performance over the Decision Tree model. • Better handling of minority and complex classes, with notable improvements in F1-score, such as: • *MITM-ArpSpoofing*: F1-score improved from 0.75 to 0.80 • *DNS_Spoofing*: Slightly better precision • *Recon-OSScan*: F1-score increased to 0.78
SVM	18.64	45.09	80.33	It finds the optimal hyperplane that maximizes the margin between different classes. SVM works well with non-linear data by mapping it to higher-dimensional spaces. SMOTE and Class Weight methods improved SVM's performance by addressing class imbalance, enabling it to classify minority classes more effectively and achieve higher accuracy.
K-Nearest Neighbors (KNN)	98.27	98.35	x	KNN depends on the similarity to its nearest neighbors. It performed well on the original and SMOTE-balanced datasets, with slightly higher accuracy using SMOTE due to improved representation of minority classes. KNN does not support class weighting, so it was not tested with the class weight approach.
DT	99.01	99.20	99.10	Achieved high accuracy across all datasets. • Slight variation due to class balancing methods. • Slightly higher accuracy with SMOTE as it provided better representation of minority classes. • Slightly weaker recall and precision in complex attack types like • Overall, balancing techniques contributed to consistent and reliable performance.
XGBoost	98.92	**99.50**	99.19	SMOTE yielded the best results, confirming that balancing techniques significantly boost XGBoost performance in imbalanced datasets

Table 3 summarizes the overall accuracy of each machine learning model under three different balancing strategies: original data (imbalanced), SMOTE, and class weight adjustment. It highlights how performance improved after addressing class imbalance.

4.3.1 Original Dataset

Now, **Table 4** presents detailed classification metrics (precision, recall, F1-score) for each algorithm when trained on the original imbalanced dataset. The results demonstrate significant bias toward the majority class and poor recall for the benign class.

Table 4:Classification Metrics Comparison – Original Dataset

Algorithm	Accuracy	precision_score	Recall	F1-Score
LogisticReg	17.86	04.95	17.86	06.70
RandomForest	**99.07**	98.97	99.07	98.98
SVM	18.64	06.08	18.64	07.17
K-Nearest Neighbors (KNN)	98.27	98.21	98.27	98.15
XGBoost	98.92	73.60	72.24	72.71
DT	99.01	98.99	99.01	98.99

4.3.2 Results After SMOTE

To address the imbalance, SMOTE was applied. The results showed improved recall and F1-score, though with some trade-offs in precision.

Table 5 shows the effect of using SMOTE to balance the dataset. Most algorithms showed improved recall and F1-score for the minority class, though some trade-offs in precision were observed.

Table 5:Classification Metrics Comparison – SMOTE Balanced Dataset

Algorithm	Accuracy	precision score	Recall	F1-Score
LogisticReg	09.90	03.44	17.52	05.32
RandomForest	99.34	99.31	99.33	99.29
SVM	45.09	34.19	45.08	36.32
K-Nearest Neighbors (KNN)	98.35	98.28	98.35	98.17
DT	99.20	99.18	99.20	99.18
XGBoost	99.48	99.47	99.50	99.47

4.3.3 Results After Class Weighting

Another approach involved adjusting class weights. This led to significant improvements in performance across most algorithms without adding synthetic data.

Finally, **Table 6** provides the metrics after applying class weight adjustments. This method improved performance across most metrics without adding synthetic data, making it a strong alternative to oversampling techniques.

Table 6:Classification Metrics Comparison – Class Weight Adjusted Dataset

Algorithm	Accuracy	precision_score	Recall	F1-Score
LogisticReg	03.68	15.50	03.68	04.14
RandomForest	98.95	98.91	98.95	98.83
SVM	80.33	80.75	80.33	78.17
K-Nearest Neighbors (KNN)	-	-	-	-
DT	99.10	99.10	99.10	99.10
XGBoost	**99.19**	99.15	99.19	99.16

4.3.4 Accuracy Comparison

The accuracy improvements achieved through SMOTE and class weighting are visually summarized in **Figure 4.1**.

The results clearly show that class balancing significantly influenced model behavior. For example, **SVM's accuracy increased from 18.6% (original data) to 80.3% using class weights**, while **Random Forest consistently maintained high performance across all balancing strategies**.

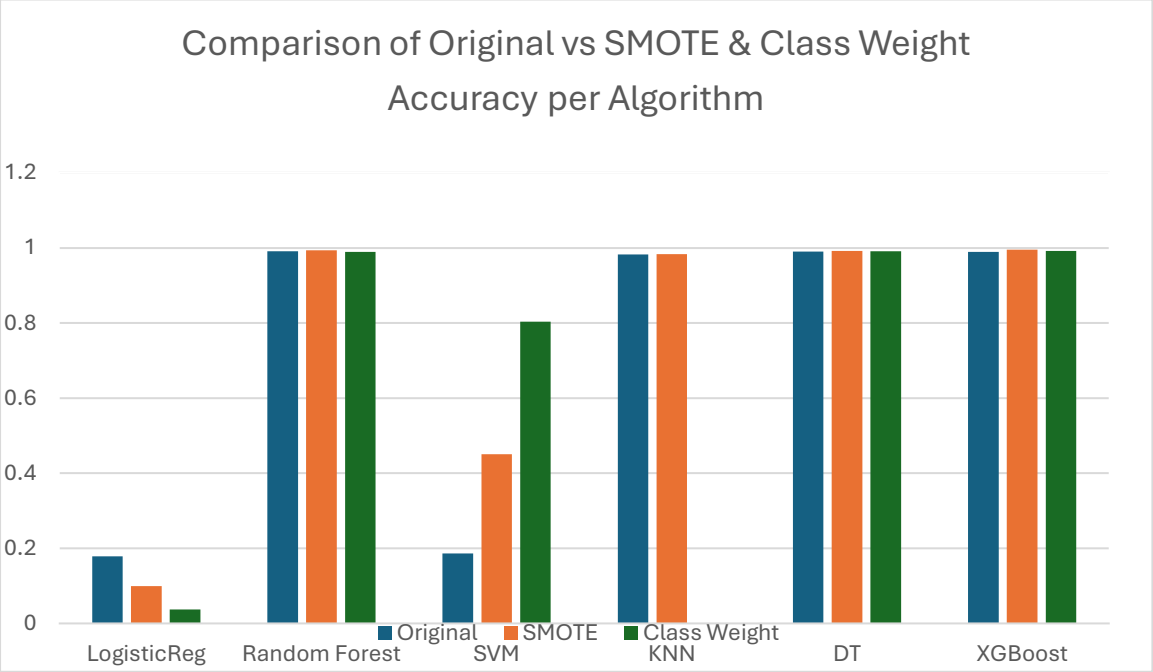


Figure4. 1: Accuracy Comparison

After evaluating all six machine learning models across different balancing strategies, XGBoost emerged as the most effective model, particularly under the SMOTE-balanced dataset. It consistently demonstrated the highest performance in terms of accuracy and F1-score across multiple attack classes. Due to its superior results and general robustness, XGBoost was selected as the final model for further testing and validation.

To validate the generalization capability of the selected model, XGBoost was trained on the main dataset and tested on a new, unseen CSV file **number 66** from the CICIoT2023 dataset. The model achieved an impressive accuracy of 0.9924, indicating strong predictive power even on data not used during training. This reinforces the model's ability to handle highly reliable real-world IoT traffic.

4.4 Results and Discussion

The results of this study demonstrate the significant impact that data preprocessing and balancing techniques can have on the performance of machine learning models in intrusion detection for IoT networks. As shown in the previous tables, models trained on the original, highly imbalanced dataset struggled to correctly classify benign traffic, resulting in low recall and F1-scores for the minority class.

After applying SMOTE and class weight adjustments, most models showed notable improvements, especially in recall and overall accuracy. For example, the SVM classifier improved from 18.6% accuracy on the original dataset to 80.3% after class weighting. Similarly, Random Forest and XGBoost consistently performed well across all data variations, with XGBoost achieving the highest accuracy (99.5%) under the SMOTE-balanced dataset.

These results highlight several important observations:

1. Balancing techniques are essential when working with highly imbalanced IoT datasets, as they allow models to better learn from minority class patterns.
2. Simpler models like Logistic Regression performed poorly, indicating that linear classifiers are not well-suited to the complex feature interactions present in IoT traffic.
3. Ensemble methods (Random Forest, XGBoost) offered the best trade-off between accuracy and generalization, making them more suitable for deployment in real-world scenarios.

In addition, the results validate the choice of using the CICIoT2023 dataset, as it provides realistic and diverse attack scenarios, enabling meaningful evaluation of intrusion detection models.

Overall, the findings of this study emphasize the importance of preprocessing and algorithm selection in building reliable IDS solutions for IoT networks, and they provide a foundation for future work that can explore deeper optimization or real-time deployment.

4.5 Challenges and Solutions

During the development and evaluation of ML-based IDS models for IoT networks, several challenges were encountered that directly impacted the design, preprocessing, and evaluation phases. These challenges and the corresponding solutions are outlined below:

• Challenge 1: Class Imbalance in the Dataset
the CICIoT2023 dataset contained a significant imbalance, with over 97% of records labeled as attacks. This skew introduced difficulty in training models that could correctly identify benign traffic.

Solution: We applied both SMOTE and Class Weight Adjustment to balance the dataset. Comparative results showed that balancing significantly improved precision and recall for the minority class.

• Challenge 2: Computational Constraints
Working with over 46 million records posed challenges in terms of processing time and memory usage, especially when using cloud platforms such as Google Colab.

Solution: We initially selected a representative subset of the data (File 92) containing ~211,000 records to conduct early experiments and establish baseline results before scaling up.

• Challenge 3: Designing Lightweight IDS for IoT
As noted in Vanin et al. [2], IoT devices often operate on wireless networks with constrained computational resources. The authors emphasize:

"To implement IDS for IoT, the IDS needs to be lightweight, require less computing power and utilize a smaller amount of data to detect threats... one of the biggest challenges for researchers will be to develop lightweight IDS that would have a high detection rate in a wireless environment" [2].

Solution: Our study focused on classical supervised ML algorithms that are known for their speed and lower computational cost, such as Logistic Regression, Random Forest, and Decision Trees, to ensure suitability for lightweight environments.

These challenges underline the complexity of building effective IDS solutions in IoT contexts, where resource limitations, data imbalance, and real-world deployment constraints all pose practical barriers. However, by employing strategic data preprocessing techniques and selecting lightweight, efficient models, this study demonstrates that it is possible to overcome these limitations and move toward more accurate and scalable IDS implementations. The insights gained from addressing these challenges directly inform the recommendations and directions proposed in the final chapter.

CHAPTER 5: Conclusion and Future Works

5.1 Conclusion

This study explored the application of supervised Machine Learning techniques for Intrusion Detection Systems (IDS) within the context of Internet of Things (IoT) networks. Using the CICIoT2023 dataset, which reflects realistic network behavior and diverse cyberattacks, a series of models were developed, trained, and evaluated under multiple data balancing strategies. The dataset was prepared to support effective learning through extensive preprocessing, including label encoding, normalization, and class balancing using SMOTE and class weight adjustment.

Our comparative analysis of six ML classifiers (Logistic Regression, SVM, KNN, Decision Tree, Random Forest, and XGBoost) revealed that Random Forest and XGBoost offered strong, consistent performance across different scenarios. Class balancing significantly improved detection accuracy, particularly for the minority class (benign traffic). While lightweight models like Logistic Regression performed acceptably under certain conditions, tree-based methods generally provided the best trade-off between accuracy and adaptability.

The findings of this research reinforce the viability of ML-based IDS solutions for IoT environments, particularly when appropriate data handling techniques are applied. Moreover, the study highlights the importance of tailoring models to IoT-specific constraints such as limited processing power, class imbalance, and the need for multiclass classification

5.2 Future Works

While this study provided a comprehensive evaluation of supervised Machine Learning models for IDS in IoT environments, several opportunities remain for future development and enhancement.

Real-Time Implementation: This project focused on offline analysis; future work can explore deploying the selected models in real-time scenarios using edge devices or gateways to assess latency, scalability, and detection speed.

Exploring Deep Learning Models: Due to resource constraints, deep learning techniques such as LSTM or CNN were not explored in this study. Future work could investigate these approaches, particularly for detecting sequential patterns and complex attack behaviors.

Feature Reduction and Optimization: Although this study retained all 47 features, future efforts may apply dimensionality reduction techniques (e.g., PCA) or feature importance methods to improve speed and generalization, especially in resource-limited IoT devices.

Integration with IoT Hardware: Future studies can simulate or deploy the IDS on actual IoT infrastructure or emulators (e.g., Cisco gateways or Raspberry Pi) to test performance in constrained environments.

Adversarial Robustness: Further exploration is needed to assess how ML models withstand adversarial attacks or poisoning attempts, particularly in security-critical IoT deployments.

Hybrid Models: Combining supervised models with unsupervised or reinforcement learning may improve the system's ability to detect unknown or evolving threats with minimal manual labeling.

These directions can help advance the integration of machine learning in IoT cybersecurity and contribute to building scalable, intelligent, and practical IDS solutions for real-world deployment.

References

[1] J. Ashraf, *Machine Learning Based Intrusion Detection System for IoT Networks*, M.Sc. Thesis, Eastern Mediterranean University, 2021.

[2] P. Vanin, T. Newe, L. L. Dhirani, E. O'Connell, D. O'Shea, B. Lee, and M. Rao, "A Study of Network Intrusion Detection Systems Using Artificial Intelligence/Machine Learning," *Sensors*, vol. 23, no. 12, p. 5639, 2023. DOI: 10.3390/s23125639

[3] M. Malhotra, "UNB CIC IoT Dataset," Kaggle, 2023. [Online]. Available: https://www.kaggle.com/datasets/madhavmalhotra/unb-cic-iot-dataset

[4] Canadian Institute for Cybersecurity, "CICIoT2023 Dataset," University of New Brunswick, 2023. [Online]. Available: https://www.unb.ca/cic/datasets/iotdataset-2023.html

[5] M. Almseidin, M. Alzubi, and Y. Bazi, "Machine Learning Techniques for Intrusion Detection Systems: A Comparative Study," *Electronics*, vol. 10, no. 3, pp. 1–22, 2021.

[6] S. Dhanabal and S. P. Shantharajah, "A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, pp. 446–452, 2015.

[7] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. O'Reilly Media, 2019.

Appendix: Source Code and Implementation Details

A.1 Data Loading and Exploration

This section includes the initial steps used to load and explore the dataset. We used pandas to load a single CSV file (File 92) from the CICIoT2023 dataset, containing 211,835 records. Before starting any preprocessing steps, we examined data types, missing values, and label distribution.

```
[2]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns

     df = pd.read_csv('/content/part-00092-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-c000.csv')

[4]  df.shape

     (124575, 47)
```

Figure4 :Dataset Loding

```
df.head()
```

_number	rst_flag_number	...	Std	Tot size	IAT	Number	Magnitue	Radius	Covariance	Variance	Weight	label
0.0	0.0	...	0.0	54.0	8.333093e+07	9.5	10.392305	0.0	0.0	0.0	141.55	DDoS-PSHACK_Flood
0.0	0.0	...	0.0	42.0	8.348239e+07	9.5	9.165151	0.0	0.0	0.0	141.55	DDoS-ICMP_Flood
0.0	0.0	...	0.0	554.0	8.376294e+07	9.5	33.286634	0.0	0.0	0.0	141.55	Mirai-udpplain
0.0	0.0	...	0.0	50.0	8.310663e+07	9.5	10.000000	0.0	0.0	0.0	141.55	DDoS-UDP_Flood
1.0	0.0	...	0.0	54.0	8.298534e+07	9.5	10.392305	0.0	0.0	0.0	141.55	DoS-SYN_Flood

Figure 5:Sample Output from CICIoT2023 Dataset

A.2 Data Preprocessing

The dataset consists of 47 features. A preliminary check using df.isnull().sum() revealed a few missing values, mostly one per column. These were handled by dropping rows containing null values, as their impact on the large dataset (~200K+ rows) was negligible.

```
# Check for missing values
df.isnull().sum()
```

	0
flow_duration	0
Header_Length	0
Protocol Type	0
Duration	0
Rate	0
Srate	0
Drate	1
fin_flag_number	1
syn_flag_number	1
rst_flag_number	1
psh_flag_number	1

Figure6 :Data Preprocessing

A.3 Full code of Model Training and Evaluation

This section provides the complete implementation code used for training, validating, and evaluating the performance of all machine learning models in this study. Each model Logistic Regression, SVM, KNN, Decision Tree, Random Forest, and XGBoost was developed and tested under three scenarios: original data, SMOTE-balanced data, and class-weight adjusted data. The code includes model configuration, fitting, prediction, and performance metric extraction, serving as a technical reference for reproducibility and further development.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from collections import Counter
df = pd.read_csv("/content/part-00092-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-c000.csv")
X = df.drop(columns=["label"])
y = df["label"]
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
value_counts = pd.Series(y_encoded).value_counts()
valid_labels = value_counts[value_counts >= 2].index
mask = pd.Series(y_encoded).isin(valid_labels)
X = X[mask]
y_encoded = y_encoded[mask]
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)
sampling_strategy = {}
for label, count in Counter(y_train).items():
    if count < 1000:
        if count < 200:
            sampling_strategy[label] = 500
        elif count < 500:
            sampling_strategy[label] = 800
        else:
            sampling_strategy[label] = 1000
min_class_size = min([count for label, count in Counter(y_train).items() if label in sampling_strategy])
k_neighbors = max(1, min(5, min_class_size - 1))
smote = SMOTE(sampling_strategy=sampling_strategy, random_state=42, k_neighbors=k_neighbors)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print("توزيع الفئات بعد SMOTE:")
for label, count in Counter(y_train_resampled).items():
    print(f"{label_encoder.inverse_transform([label])[0]}: {count}")
```

Figure7 :SMOTE code

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('/content/SMOTED DATA NEW 2.csv')
X = df.drop('label', axis=1)
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
acc_dt = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {acc_dt:.4f}")
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(6, 4))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Greens')
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
from sklearn.metrics import classification_report
print("Classification Report - Decision Tree:")
print(classification_report(y_test, y_pred_dt))
```

Figure:8 Decision Tree

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv('/content/SMOTED DATA NEW 2.csv')
X = df.drop('label', axis=1)
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, output_dict=True)

print(f"Random Forest Accuracy (SMOTEd Data): {accuracy:.4f}\n")
print("Classification Report (SMOTEd Data):")
print(classification_report(y_test, y_pred))

print("\nTotal (Weighted Average):")
print(f"Precision:  {report['weighted avg']['precision']:.4f}")
print(f"Recall:     {report['weighted avg']['recall']:.4f}")
print(f"F1-Score:   {report['weighted avg']['f1-score']:.4f}")
print(f"Support:    {report['weighted avg']['support']}")
```

Figure 9 :Random Forest

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
df = pd.read_csv("/content/SMOTED DATA NEW 2.csv")
X = df.drop('label', axis=1)
y = df['label']
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(
    X_train_smote, y_train_smote, test_size=0.3, random_state=42, stratify=y_train_smote)
log_reg = LogisticRegression(
    multi_class='multinomial',
    solver='lbfgs',
    max_iter=1000,
    random_state=42)
log_reg.fit(X_train_smote, y_train_smote)
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression Accuracy: {accuracy:.4f}")
print("Classification Report:\n")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
sns.heatmap(confusion_matrix(y_test, y_pred), cmap='Blues', annot=False, fmt='d')
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Figure 10 :Logistic Regression

```python
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
df = pd.read_csv("/content/SMOTED DATA NEW 2.csv")
X = df.drop('label', axis=1)
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
svm_model = SVC(kernel='rbf', decision_function_shape='ovr', random_state=42)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Multi-Class SVM')
plt.show()
```

Figure 11:SVM

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
df = pd.read_csv('/content/SMOTED DATA NEW 2.csv')
X = df.drop('label', axis=1)
y = df['label']
X_train_smote, X_test, y_train_smote, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)
knn_smote = KNeighborsClassifier(n_neighbors=5)
knn_smote.fit(X_train_smote, y_train_smote)
y_pred_smote = knn_smote.predict(X_test)
accuracy_smote = accuracy_score(y_test, y_pred_smote)
print(f'SMOTE Data Accuracy (KNN): {accuracy_smote:.4f}')
print("\nClassification Report (KNN - SMOTEd Data):")
print(classification_report(y_test, y_pred_smote))
cm_smote = confusion_matrix(y_test, y_pred_smote)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_smote, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - KNN (SMOTEd Data)')
plt.tight_layout()
plt.show()
```

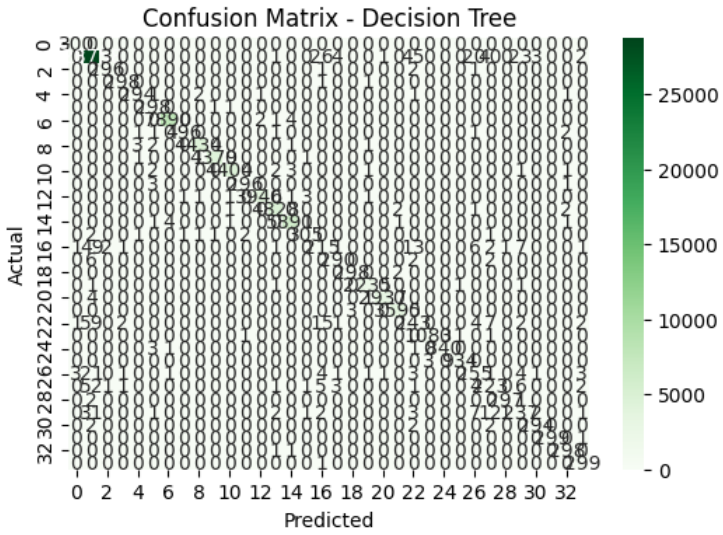
Table 7:KNN

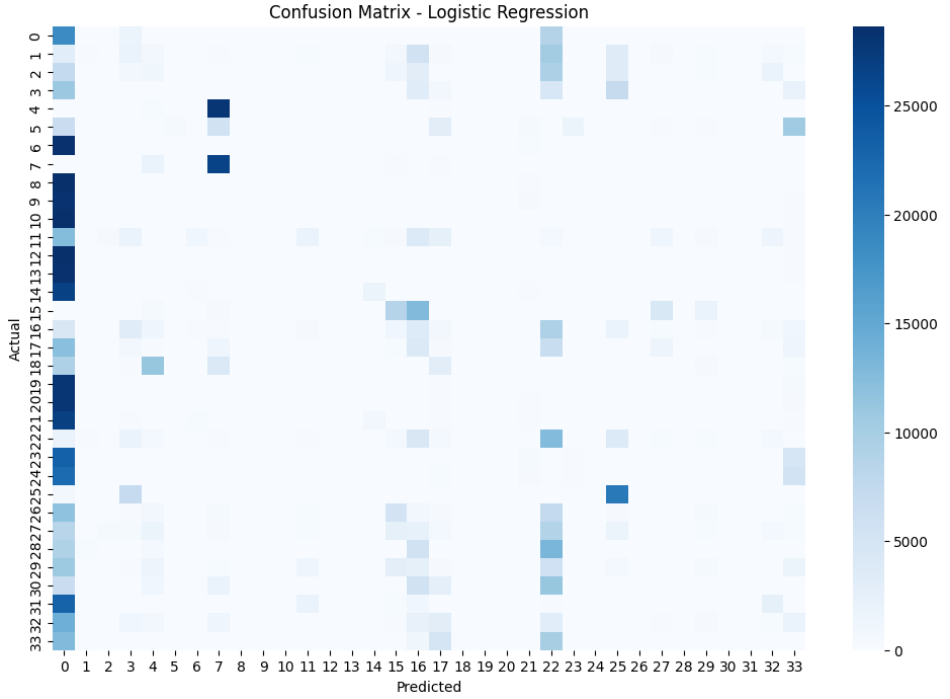
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
import joblib
df = pd.read_csv('/content/SMOTED DATA NEW 2.csv')
df = df.dropna()
class_counts = df['label'].value_counts()
single_instance_classes = class_counts[class_counts == 1].index.tolist()
df = df[~df['label'].isin(single_instance_classes)]
X = df.drop('label', axis=1)
y = df['label']
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.3, random_state=42, stratify=y_encoded)
xgb = XGBClassifier(eval_metric='mlogloss', random_state=42)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
y_test_labels = label_encoder.inverse_transform(y_test)
y_pred_labels = label_encoder.inverse_transform(y_pred)
accuracy = accuracy_score(y_test_labels, y_pred_labels)
report = classification_report(y_test_labels, y_pred_labels)
```

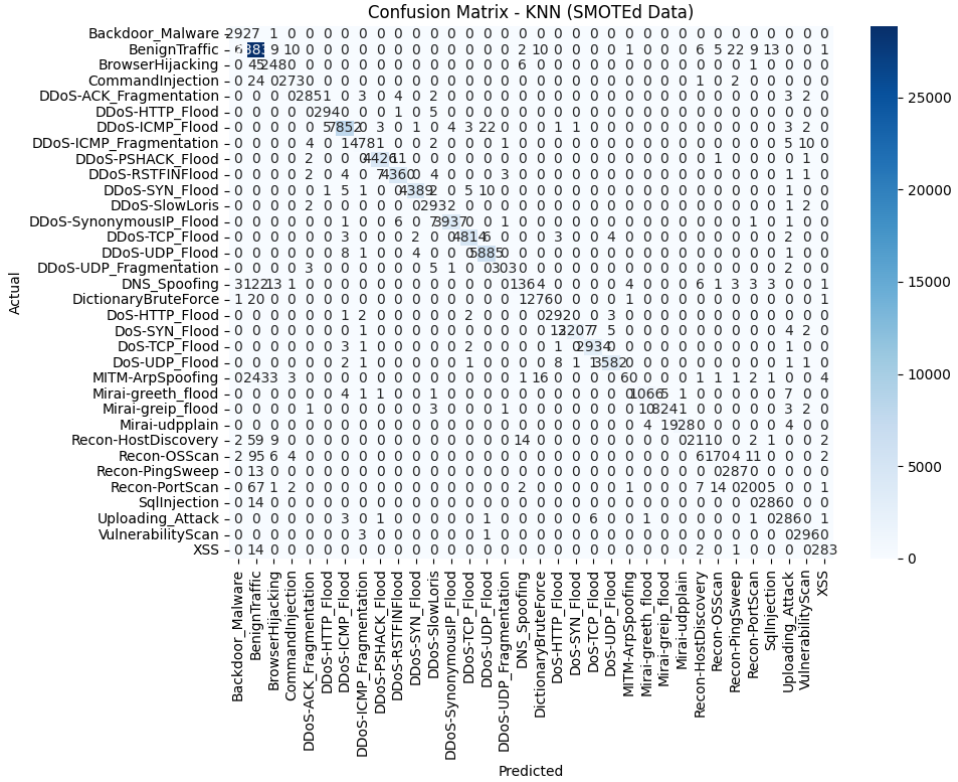
Table 8: XGBoost

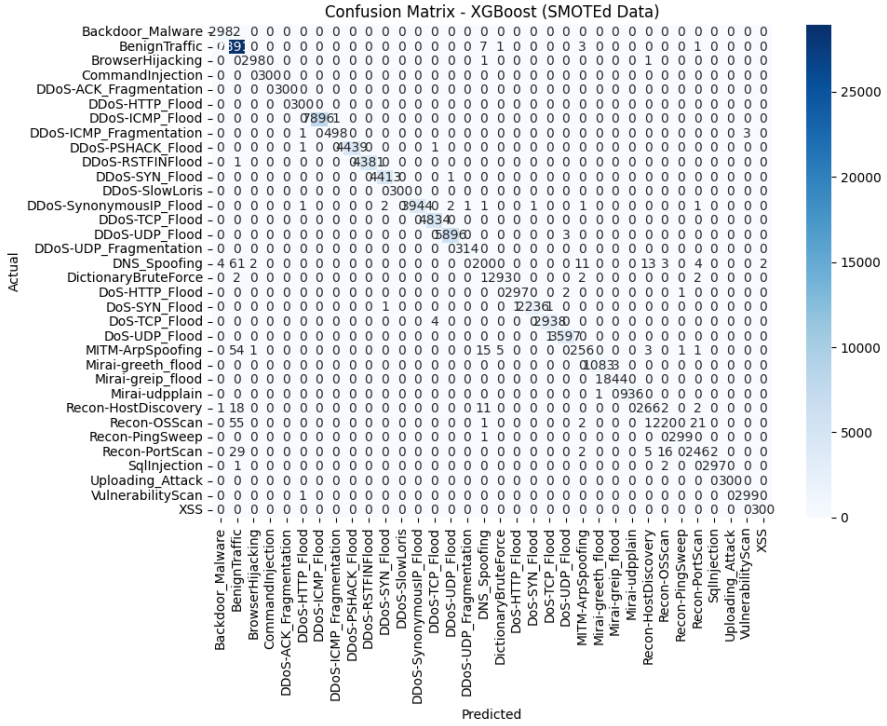
A.4 Confusion Matrix

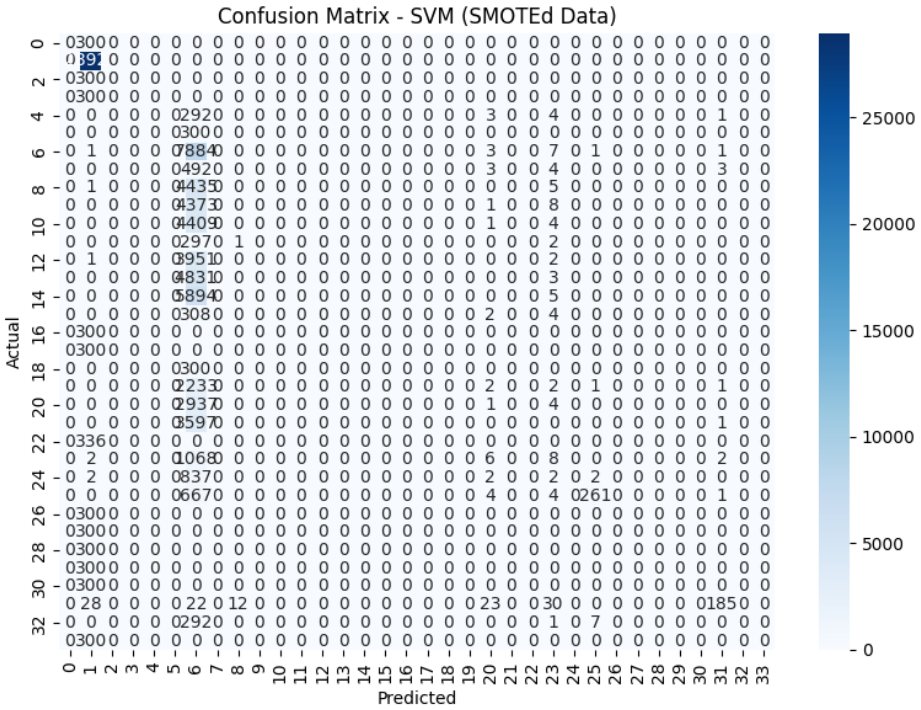
The confusion matrix is a crucial evaluation tool that offers detailed insights into how well a machine learning model distinguishes between different classes. In the context of intrusion detection, it highlights not only overall accuracy but also the distribution of true positives, false positives, and false negatives across attack types and benign traffic. The following confusion matrices illustrate the classification performance of all six models implemented in this study Logistic Regression, SVM, KNN, Decision Tree, Random Forest, and XGBoost under various class balancing strategies.

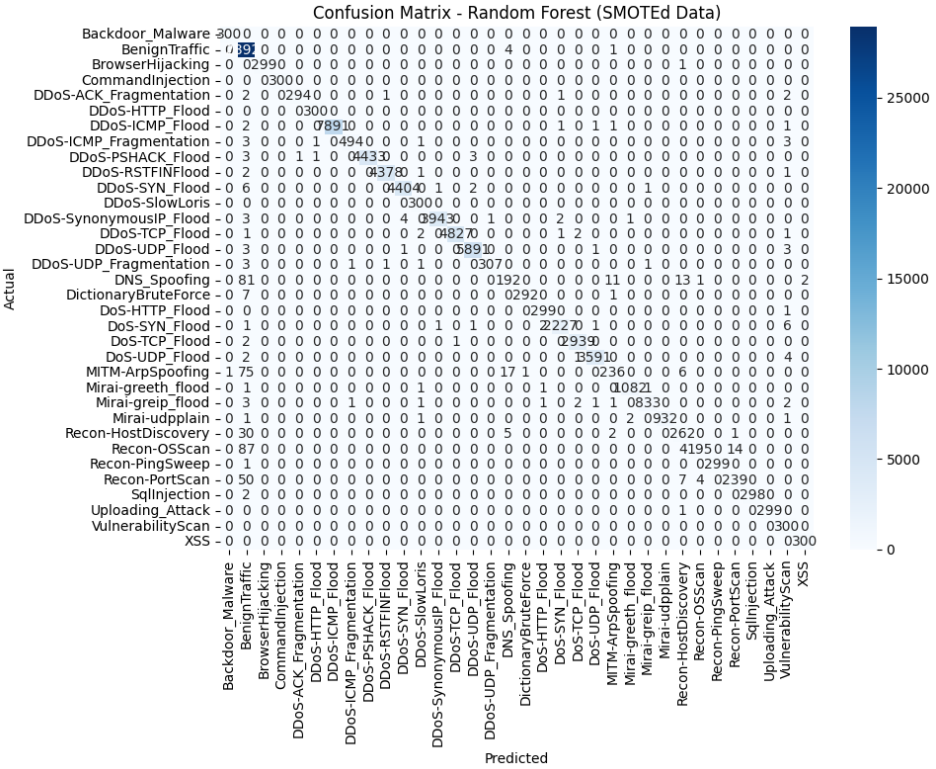



Confusion Matrix - Logistic Regression
Actual
Predicted
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
25000
20000
15000
10000
5000
0


Confusion Matrix - KNN (SMOTEd Data)
Actual
Predicted
Backdoor_Malware
BenignTraffic
BrowserHijacking
CommandInjection
DDoS-ACK_Fragmentation
DDoS-HTTP_Flood
DDoS-ICMP_Flood
DDoS-ICMP_Fragmentation
DDoS-PSHACK_Flood
DDoS-RSTFINFlood
DDoS-SYN_Flood
DDoS-SlowLoris
DDoS-SynonymousIP_Flood
DDoS-TCP_Flood
DDoS-UDP_Flood
DDoS-UDP_Fragmentation
DNS_Spoofing
DictionaryBruteForce
DoS-HTTP_Flood
DoS-SYN_Flood
DoS-TCP_Flood
DoS-UDP_Flood
MITM-ArpSpoofing
Mirai-greeth_flood
Mirai-greip_flood
Mirai-udpplain
Recon-HostDiscovery
Recon-OSScan
Recon-PingSweep
Recon-PortScan
SqlInjection
Uploading_Attack
VulnerabilityScan
XSS
25000
20000
15000
10000
5000
0


Confusion Matrix - XGBoost (SMOTEd Data)
Actual
Predicted
Backdoor_Malware
BenignTraffic
BrowserHijacking
CommandInjection
DDoS-ACK_Fragmentation
DDoS-HTTP_Flood
DDoS-ICMP_Flood
DDoS-ICMP_Fragmentation
DDoS-PSHACK_Flood
DDoS-RSTFINFlood
DDoS-SYN_Flood
DDoS-SlowLoris
DDoS-SynonymousIP_Flood
DDoS-TCP_Flood
DDoS-UDP_Flood
DDoS-UDP_Fragmentation
DNS_Spoofing
DictionaryBruteForce
DoS-HTTP_Flood
DoS-SYN_Flood
DoS-TCP_Flood
DoS-UDP_Flood
MITM-ArpSpoofing
Mirai-greeth_flood
Mirai-greip_flood
Mirai-udpplain
Recon-HostDiscovery
Recon-OSScan
Recon-PingSweep
Recon-PortScan
SqlInjection
Uploading_Attack
VulnerabilityScan
XSS
25000
20000
15000
10000
5000
0


Confusion Matrix - SVM (SMOTEd Data)
Actual
Predicted
25000
20000
15000
10000
5000
0


Confusion Matrix - Random Forest (SMOTEd Data)
Backdoor_Malware
BenignTraffic
BrowserHijacking
CommandInjection
DDoS-ACK_Fragmentation
DDoS-HTTP_Flood
DDoS-ICMP_Flood
DDoS-ICMP_Fragmentation
DDoS-PSHACK_Flood
DDoS-RSTFINFlood
DDoS-SYN_Flood
DDoS-SlowLoris
DDoS-SynonymousIP_Flood
DDoS-TCP_Flood
DDoS-UDP_Flood
DDoS-UDP_Fragmentation
DNS_Spoofing
DictionaryBruteForce
DoS-HTTP_Flood
DoS-SYN_Flood
DoS-TCP_Flood
DoS-UDP_Flood
MITM-ArpSpoofing
Mirai-greeth_flood
Mirai-greip_flood
Mirai-udpplain
Recon-HostDiscovery
Recon-OSScan
Recon-PingSweep
Recon-PortScan
SqlInjection
Uploading_Attack
VulnerabilityScan
XSS
Actual
Predicted
25000
20000
15000
10000
5000
0