Comparative Multi-Class Intrusion Detection Using Machine Learning with SHAP-based Interpretability on CIC-IDS-2017

Small Project in Artificial Intelligence and Cybersecurity (650.060)
Summer Term, 2025

Babayev Emil
12350332

1. Introduction

In recent years, the rapid proliferation of internet-connected systems and applications has led to a significant increase in the number and sophistication of cyberattacks. Among the various categories of network intrusions, distributed denial-of-service (DDoS), port scanning, brute force attacks, and web-based threats represent some of the most prevalent and harmful activities targeting information infrastructure [1]. Efficient detection of these activities is critical for maintaining network integrity, confidentiality, and availability. Consequently, network intrusion detection systems (NIDS) have become essential components of modern cybersecurity architectures [2].

Machine learning (ML) methods offer powerful tools for automating attack detection by learning patterns from labeled datasets [3]. In this project, I investigated multiple supervised learning algorithms for multi-class classification of network traffic records and focused on predicting specific types of malicious behavior based on a labeled dataset consisting of diverse attacks and normal traffic. My goal is to evaluate the effectiveness of different models, analyze their strengths and weaknesses, and interpret their decision-making using explainability techniques [4].

This report documents the implementation, experimentation, and evaluation of machine learning classifiers trained on the CIC-IDS-2017 dataset [5]. I paid particular attention to model performance across highly imbalanced classes, using undersampling techniques to mitigate skewed distributions [6]. Ultimately, the goal was to identify which models perform best in real-world scenarios where distinguishing among many similar attack types is crucial for practical deployment.

2. Intrusion Detection Systems in Cybersecurity

An **Intrusion Detection System (IDS)** is a critical component of modern cybersecurity architectures, designed to monitor and analyze network or system activity for signs of malicious behavior [1]. IDS tools detect unauthorized access attempts, policy violations, or anomalous traffic patterns that may indicate cyberattacks. More broadly, IDS technologies can be classified into two categories:

- **Host-based IDS (HIDS):** Monitors activity on individual devices or endpoints. It inspects system calls, application logs, and file integrity to identify intrusions on a specific host.
- **Network-based IDS (NIDS):** Analyzes network traffic in real time to detect suspicious flows or packets. NIDS is typically deployed at strategic points within a network to monitor inbound and outbound traffic.

NIDS plays an especially crucial role in enterprise and critical infrastructure environments, where it can identify large-scale threats such as Distributed Denial-of-Service (DDoS) attacks, brute-force login attempts, scanning activities, and malware propagation [2].

Traditional IDS approaches rely on signature-based detection, which matches observed traffic against a database of known attack patterns. However, this method struggles to detect new, unknown, or obfuscated threats. To address these limitations, machine learning-based IDS systems have gained popularity. These systems can learn to recognize previously unseen attack behaviors based on statistical patterns and contextual features within network traffic [3].

This project focuses on developing and evaluating a machine learning-based NIDS solution capable of multi-class classification of network intrusions. By leveraging labeled data and interpretable algorithms, the goal is to build a system that not only detects threats but also explains the reasoning behind its alerts.

3. Dataset Generation Process: CIC-IDS-2017

The CIC-IDS-2017 dataset, developed by the Canadian Institute for Cybersecurity (CIC), represents one of the most comprehensive and realistic datasets for evaluating intrusion detection systems [5]. Unlike many earlier benchmarks, CIC-IDS-2017 was designed to reflect both legitimate user behavior and a wide variety of contemporary cyberattacks within a controlled yet authentic enterprise network setting.

To simulate real-world conditions, the dataset was generated using a closed environment modeled after a typical organizational network infrastructure. This testbed included servers, routers, switches, and end-user devices running both Windows and Linux operating systems. Simulated users were configured to perform standard daily activities such as browsing the web, using email services, accessing databases, chatting over instant messengers, transferring files via FTP, and streaming multimedia content. These user actions were not artificially scripted but generated through tools such as IXIA PerfectStorm and B-Profile, which ensure temporal and protocol-level realism in the captured traffic [7].

Over a period of five consecutive working days (Monday to Friday), the system recorded network traffic under different usage scenarios, with each day dedicated to distinct attack

categories or normal activities. Traffic was captured in raw format using packet capture tools like Wireshark and tcpdump. These PCAP files were subsequently processed using CICFlowMeter — a flow-based feature extractor developed by CIC — to compute over 80 features per connection. These features included low-level TCP/IP and UDP metrics (packet inter-arrival times, header flags, etc.), statistical descriptors (e.g., minimum and maximum packet length), and derived metrics such as the number of packets in the forward and backward directions.

Each network flow was then labeled according to the traffic scenario and the time window in which it occurred. This labeling process was manual yet precise, relying on prior knowledge of the attack schedule and traffic content to assign the correct class to each record. The final output consisted of structured CSV files that contained preprocessed and labeled data suitable for machine learning applications.

Importantly, the dataset covers a diverse range of protocols and applications, providing multi-protocol feature distributions and enabling the modeling of both volumetric (e.g., DDoS) and behavioral (e.g., infiltration) attack patterns. This diversity not only enhances the quality of machine learning training but also makes the dataset relevant for practical deployment in modern network environments.

In summary, CIC-IDS-2017 stands out for its:

- High fidelity to real-world enterprise traffic.
- Detailed and multi-layered flow features.
- Balanced simulation of both benign and malicious behaviors.
- Manual yet accurate labeling grounded in scenario-based testing.

This richness makes CIC-IDS-2017 a benchmark dataset widely accepted in cybersecurity research and ideal for training, validating, and interpreting intrusion detection models.

4. Attack Types and Protocols in CIC-IDS-2017

The CIC-IDS-2017 dataset was designed not only to replicate normal enterprise network behavior but also to simulate a wide range of cyberattacks targeting different layers of the OSI model. Each day of data collection introduced a set of specific attack scenarios with known vectors, protocols, and payload characteristics. This ensured that the resulting dataset would encompass both high-volume and stealthy attack patterns relevant to modern cybersecurity defense [5].

The following categories of attacks were included in the dataset:

4.1 Denial of Service (DoS) and Distributed Denial of Service (DDoS)

DoS and DDoS attacks attempt to overwhelm the target network or service by flooding it with excessive traffic, leading to resource exhaustion and service unavailability. In CIC-IDS-2017, these attacks were simulated using popular tools such as LOIC and HULK. Different subtypes include:

- **DoS Hulk** – an aggressive attack using HTTP flooding with randomized parameters to bypass caching and overload web servers.

- **DoS Slowloris** – a low-bandwidth attack exploiting HTTP header timeouts to keep many connections open simultaneously, consuming server threads.
- **DoS GoldenEye** – a similar HTTP-based flood attack, sending incomplete requests to hold server resources.
- **DDoS** – a distributed version of flooding conducted from multiple simulated machines.

4.2 Brute Force Attacks

These attacks aim to gain unauthorized access by systematically guessing passwords. The dataset includes:

- **FTP-Patator** and **SSH-Patator** – tools used to perform dictionary attacks on FTP and SSH services, respectively. They generate multiple failed login attempts, which can be detected via connection failure patterns.

4.3 Port Scanning

PortScan simulates reconnaissance behavior in which attackers scan IP addresses and ports to identify open services and vulnerable configurations. Tools like Nmap were used to generate these scans, which produce characteristic patterns in network flows with minimal data payloads but frequent connection attempts [8].

4.4 Botnet Activity

Bot traffic models infected hosts under remote command and control (C&C). The behavior includes periodic beaconing, payload delivery, and coordinated actions. This class is important because it blends both legitimate and malicious patterns, making detection more complex.

4.5 Web-Based Attacks

Several classes in the dataset capture exploitation of web application vulnerabilities:

- **Web Attack – Brute Force** – automated guessing of login credentials through web forms.
- **Web Attack – XSS (Cross-Site Scripting)** – injection of malicious scripts into web pages.
- **Web Attack – Sql Injection** – attempts to execute SQL commands via vulnerable input fields [9].

4.6 Infiltration and Heartbleed

- **Infiltration** – simulates an insider threat scenario or lateral movement from a compromised internal host. This includes malware spreading internally through techniques like privilege escalation or credential reuse.

- **Heartbleed** – exploits the CVE-2014-0160 vulnerability in OpenSSL. The attacker sends malformed heartbeat requests to leak sensitive data from the memory of SSL-enabled servers [10].

4.7 Protocol Coverage and OSI Layers

The attacks span multiple OSI layers and protocols:

- **Application Layer (HTTP, FTP, SSH, SSL)** – exploited in web, brute-force, and Heartbleed attacks.
- **Transport Layer (TCP/UDP)** – manipulated in scanning and DoS attacks through SYN flooding, connection resets, and timing patterns.
- **Network Layer (IP)** – involved in DDoS amplification and spoofing scenarios.

This diversity of attacks and protocols ensures that any model trained on CIC-IDS-2017 must learn both high-level behavioral anomalies and low-level traffic patterns, thereby making it an ideal benchmark for evaluating robust and adaptive intrusion detection systems.

5. Machine Learning and Model Overview

Machine learning (ML) is a subfield of artificial intelligence focused on creating systems that can learn patterns from data and make decisions or predictions without being explicitly programmed for every task [11]. In the context of cybersecurity, ML allows for dynamic threat detection based on previous observations of benign and malicious behavior. This adaptability makes ML-based intrusion detection systems more resilient than traditional signature-based tools, especially against novel or evolving attack vectors [3].

There are several categories of ML, but this project focuses on **supervised learning**, where the model is trained on labeled examples — each input is associated with a known class (e.g., "DoS attack", "Benign", "Web XSS attack"). The goal is to learn a function that maps input features (e.g., flow duration, packet size) to the correct class label. Once trained, the model can classify new, unseen traffic with similar statistical characteristics.

The core workflow of a supervised ML pipeline includes:

1. **Preprocessing:** cleaning, imputing, and normalizing data to ensure compatibility with ML algorithms.
2. **Feature engineering:** selecting, transforming, and scaling attributes to improve model interpretability and performance.
3. **Model training:** optimizing model parameters by minimizing a loss function, often using gradient descent-based algorithms.
4. **Evaluation:** measuring model performance using metrics such as accuracy, precision, recall, and F1-score.
5. **Interpretation:** using explainability tools such as SHAP (SHapley Additive exPlanations) to identify which features drive model decisions.

Evaluation Metrics

A few key metrics were used to evaluate model performance:

- **Accuracy:** the proportion of correctly classified instances out of the total samples.
- **Precision:** the proportion of true positives among all positive predictions.
- **Recall:** the proportion of true positives among all actual positives.
- **F1-score:** the harmonic mean of precision and recall. Especially useful in imbalanced datasets.
- **Macro-F1:** average of F1-scores across all classes, treating each class equally regardless of frequency.

Activation functions play a critical role in deep models. The most commonly used function in this study is **ReLU (Rectified Linear Unit)** defined as $f(x) = \max(0, x)$. It introduces non-linearity and speeds up convergence during backpropagation in neural networks [12].

5.1 Multilayer Perceptron (MLPClassifier)

The Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural network that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node (also called a neuron) is connected to nodes in the subsequent layer and uses a nonlinear activation function, typically the Rectified Linear Unit (ReLU) [13], to introduce non-linearity into the model. MLPs are capable of learning complex, non-linear mappings between input features and target labels, which makes them powerful tools for multi-class classification tasks.

In this project, the `MLPClassifier` from `scikit-learn` was applied to the CIC-IDS-2017 dataset [5]. The model was configured with two hidden layers, each containing 100 neurons, and the ReLU activation function. The optimizer used was Adam, a stochastic gradient-based optimizer well-suited for large datasets and high-dimensional parameter spaces [14].

The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

This function has been shown to accelerate convergence and avoid vanishing gradients compared to traditional sigmoid or tanh functions [12].

The output layer used the softmax activation function to assign probabilities to each class. The model was trained using the cross-entropy loss, a standard choice for multi-class classification problems. Regularization techniques such as early stopping and dropout can be applied, but only early stopping was used here.

Performance-wise, the MLP achieved a macro-F1 score of 0.82 and an accuracy of 99.5%, placing it just behind Random Forest. Although it requires more computational resources, the MLP captured complex patterns in the data better than linear models, especially for rare classes.

Feature importance was analyzed using SHAP values [4], which demonstrated that MLP learned interactions between features such as `Flow Duration`, `Fwd Packet Length Mean`, and `Bwd Packet Length Min` that were less visible in simpler models.

Despite their strengths, MLPs are often considered "black-box" models. Explainability techniques like SHAP and LIME are essential to understand and validate their behavior in sensitive domains like cybersecurity [15].

In summary, MLP provides a flexible and powerful modeling framework for network intrusion detection. While more computationally demanding, its superior capacity for learning non-linear decision boundaries makes it a suitable candidate for real-world deployment when combined with interpretability tools.

5.2 Random Forest Classifier

Random Forest is an ensemble learning algorithm that builds multiple decision trees during training and outputs the mode of their predictions for classification tasks [16]. It belongs to the family of bagging methods, where each tree is trained on a bootstrap sample of the dataset and splits are made using a random subset of features. This strategy reduces variance and helps prevent overfitting, making Random Forest a robust choice for intrusion detection tasks.

In this project, Random Forest was applied to the CIC-IDS-2017 dataset [5], using the implementation from `scikit-learn` [17]. The classifier was configured with 100 trees (estimators), with each tree trained to a maximum depth of 20 to balance performance and computational efficiency.

The underlying decision tree model, originally introduced by Quinlan [18], splits data at each node using the feature that provides the highest information gain, often measured by the Gini impurity or entropy. In Random Forest, the randomness in feature selection ensures decorrelation among individual trees, resulting in more diverse learners.

Random Forest outputs class probabilities by averaging the predicted probabilities from each individual tree. Its ability to model complex, non-linear decision boundaries makes it highly effective in cybersecurity domains, especially for imbalanced datasets [19].

In our evaluation, Random Forest achieved the highest overall accuracy (99.6%) and macro-F1 score (0.84) across all classifiers. This result highlights its effectiveness in capturing both majority and minority class behaviors. Furthermore, it showed robust performance with minimal hyperparameter tuning and relatively fast inference times compared to neural models.

Feature importance from Random Forest was also extracted natively using the Gini importance metric. This helped to identify key features such as `Flow Duration`, `Fwd Packet Length Max`, and `Bwd Packet Length Std`, supporting further dimensionality reduction and interpretability efforts.

While less prone to overfitting than single decision trees, Random Forest models can still become complex and memory-intensive with large numbers of estimators. However, their ease of implementation, interpretability, and strong generalization performance justify their widespread use in intrusion detection and similar security applications [20].

Overall, Random Forest presents a powerful yet interpretable approach for multi-class intrusion detection, balancing simplicity, performance, and robustness.

5.3 Logistic Regression

Logistic Regression is a linear model used for binary and multi-class classification tasks. Unlike linear regression, which predicts a continuous output, logistic regression predicts the probability of a categorical outcome using the logistic (sigmoid) function [21]:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

For multi-class classification, the softmax function is used to model probabilities across multiple classes. The model learns parameters by maximizing the likelihood of the observed data, typically using optimization methods such as gradient descent.

In this project, Logistic Regression was implemented using the `scikit-learn` library [17], with a one-vs-rest scheme for handling the multi-class CIC-IDS-2017 dataset [5]. The features were standardized prior to training, as logistic regression is sensitive to the scale of input variables.

One of the main advantages of logistic regression is its interpretability: each coefficient represents the log-odds change associated with a unit increase in the corresponding feature. However, this simplicity limits the model's ability to capture non-linear patterns and feature interactions, especially in high-dimensional or complex data settings like network intrusion detection [3].

In the experimental results, Logistic Regression achieved the lowest accuracy and macro-F1 score among the five models tested. The confusion matrix and SHAP values indicate that it primarily relies on high-variance, linearly separable features and performs poorly on minority classes such as Heartbleed and Infiltration.

Despite its limitations, logistic regression is often used as a baseline due to its efficiency, robustness to overfitting on small datasets, and interpretability. It remains relevant in explainable AI frameworks and as a diagnostic tool to identify whether linear separation of classes is feasible [15].

5.4 Decision Tree Classifier

The Decision Tree classifier is a non-parametric supervised learning method used for classification and regression tasks. It models data by learning simple decision rules inferred from features [18]. In a tree structure, internal nodes represent feature-based decisions, branches represent outcomes of these decisions, and leaf nodes correspond to output classes.

Each node in a Decision Tree splits the dataset based on a feature that maximizes the information gain or reduces impurity. In this project, the Gini impurity was used as the splitting criterion. The Gini impurity for a node is defined as:

$$G = 1 - \sum_{i=1}^{n} p_i^2$$

where p_i is the probability of class i at that node. A node with Gini impurity of 0 is pure, containing only one class.

The implementation used here follows the `scikit-learn` library [17], with constraints on the maximum depth and minimum samples per leaf to prevent overfitting—a common issue in deep, unpruned trees.

Decision Trees are inherently interpretable, as their structure can be visualized and translated into simple if-then rules. This makes them particularly appealing in cybersecurity applications where understanding decision logic is essential [22]. However, they are sensitive to noisy data and may perform poorly on imbalanced datasets without proper preprocessing.

In this study, the Decision Tree achieved a macro-F1 score of 0.72 and was particularly effective at classifying majority classes such as benign and DoS attacks. However, it misclassified some minority classes such as Heartbleed and Infiltration, which highlights the challenge of class imbalance [19].

While Decision Trees are prone to overfitting, they serve as a strong base learner in ensemble methods like Random Forest and can be valuable for quick prototyping and feature importance analysis.

Due to their simplicity, transparency, and decent performance, Decision Trees remain a useful baseline in intrusion detection systems, especially when interpretability is prioritized [20].

5.5 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for classification and regression. It classifies a data point by a majority vote of its k closest neighbors in the feature space, measured by a distance metric such as Euclidean distance [23]. The algorithm does not learn an explicit model but stores all training instances, making predictions based on local proximity.

Given a test instance x, the Euclidean distance from each training point x_i is computed as:

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

The predicted class is the most frequent label among the k nearest training samples. The choice of k significantly affects performance: a small k makes the model sensitive to noise, while a large k smooths the decision boundary and may overlook local patterns.

In this study, KNN was applied using `scikit-learn` [17], with k selected via cross-validation. Features were scaled using `MinMaxScaler`, since KNN is highly sensitive to feature magnitude. The algorithm was evaluated on the CIC-IDS-2017 dataset [5], which includes heterogeneous features requiring proper normalization to ensure meaningful distance comparisons.

The model achieved moderate accuracy but struggled with minority classes due to the imbalanced nature of the dataset. Unlike models that incorporate learning and generalization, KNN's reliance on raw distance limits its effectiveness in high-dimensional or sparse data, where the notion of "closeness" can degrade—known as the *curse of dimensionality*.

KNN's strengths lie in its simplicity and ability to model non-linear decision boundaries without explicit parameter tuning. However, the memory requirement and inference time increase with dataset size, making it less practical for large-scale real-time intrusion detection systems.

6. Dataset Overview

The dataset used in this study is derived from the publicly available CIC-IDS-2017 dataset, which simulates realistic traffic using multiple scenarios and attack types [5]. It is one of the most comprehensive labeled datasets for intrusion detection research, offering a rich mix of application-layer protocols, user behavior, and malicious traffic.

The dataset consists of two main folders: `MachineLearningCVE`, which contains preprocessed CSV files with extracted and labeled flow-based features, and `TrafficLabelling`, which includes raw PCAP files used for feature extraction and labeling via CICFlowMeter [7].

Each CSV file represents a network flow — a unidirectional sequence of packets sharing a 5-tuple (source IP, destination IP, source port, destination port, and protocol). These flows are characterized by 78 statistical and protocol-related features, such as flow duration, packet length statistics, TCP flag counts, and inter-arrival times. The target column, `Label`, identifies the traffic class, which may be one of several types of attack or benign activity.

In total, we observed 15 distinct labels: `BENIGN`, `DDoS`, `PortScan`, `Bot`, `DoS Hulk`, `DoS GoldenEye`, `DoS Slowloris`, `FTP-Patator`, `SSH-Patator`, `Web Attack -- Brute Force`, `Web Attack -- XSS`, `Web Attack -- Sql Injection`, `Infiltration`, and `Heartbleed`.

Prior to feeding the data into machine learning pipelines, the following preprocessing steps were conducted:

- **Feature cleaning:** Columns with constant values across samples or excessive noise were removed to avoid redundancy.
- **Encoding:** Non-numeric columns such as `Protocol` were either excluded or one-hot encoded to enable compatibility with ML algorithms.
- **Feature scaling:** All numeric features were standardized using `StandardScaler` to ensure zero mean and unit variance, which accelerates convergence for gradient-based models like neural networks [17].
- **Handling class imbalance:** Since the dataset is heavily skewed toward benign traffic and a few common attack types (e.g., `DoS Hulk`), random undersampling was performed to balance the number of samples per class and reduce classifier bias toward dominant categories [19].

The processed dataset was then split into training and test sets with preserved class distribution using stratified sampling. This stratification ensures that all attack categories, including minority classes like `Heartbleed` and `Infiltration`, are adequately represented in both subsets.

This balanced and normalized version of the CIC-IDS-2017 dataset forms the foundation for the comparative training and evaluation of the machine learning classifiers discussed in the subsequent sections.

7. Preprocessing and Feature Engineering

Data preprocessing and feature engineering are essential steps in any machine learning pipeline, particularly when working with real-world datasets such as CIC-IDS-2017, which may include noise, redundancy, or skewed distributions. These steps ensure that the models receive clean, normalized, and relevant inputs, thereby improving learning efficiency and predictive performance.

The following preprocessing actions were performed:

- **Null and Infinite Value Removal:** Any rows containing missing values (NaNs) or infinite values were eliminated. These anomalies often arise from network timeouts or feature extraction artifacts and, if unaddressed, can disrupt training or bias model predictions [24].

- **Feature Scaling:** All numeric features were normalized using `MinMaxScaler` from the `scikit-learn` library, which maps each feature value to a range between 0 and 1 [17]. This normalization is especially critical for distance-based or gradient-based algorithms such as K-Nearest Neighbors and Multilayer Perceptron, where varying feature magnitudes can skew results or impede convergence.

- **Label Encoding:** The target column, which originally contained string-formatted class labels (e.g., "BENIGN", "DoS Hulk"), was converted into numerical form using `LabelEncoder`. This transformation enables classifiers to operate on the label space as categorical integers [25].

- **Class Imbalance Mitigation:** The dataset was heavily imbalanced, with the majority class (`BENIGN`) comprising over 70% of the total samples. To counteract this, random undersampling was applied to dominant classes, reducing their sample count to better match that of minority classes. This approach ensures more balanced learning and reduces the tendency of models to overfit the majority class [26].

- **Low-Variance Feature Removal:** Features exhibiting little to no variation across samples were discarded. Such features contribute negligible discriminatory power and only increase computational cost. Removing them helps simplify the learning task and improves training efficiency [27].

These preprocessing and feature engineering techniques served to standardize the input space, reduce noise, and promote fair learning across classes. The resulting dataset allowed for improved generalization of machine learning models, especially in the presence of complex, overlapping class boundaries common in intrusion detection tasks.

8. Methodology and Implementation

The implementation of this project adhered to a structured and modular machine learning pipeline. This approach ensured consistency, reproducibility, and fair comparison across various classification models applied to the CIC-IDS-2017 dataset. The process was executed in the following major phases: data preprocessing, training-validation split, model selection and configuration, performance evaluation, and post hoc explainability analysis.

8.1 Pipeline Overview

The dataset was initially loaded into memory using the `pandas` library. After removing noise and performing feature scaling and class balancing (see Section 7), the resulting dataset was split into training (80%) and test (20%) subsets using stratified sampling to preserve the distribution of attack classes.

Each model was implemented using `scikit-learn` APIs and trained on the same dataset split to ensure fair comparison. Hyperparameters were selected based on prior knowledge and basic tuning experiments. The following classifiers were employed:

- **MLPClassifier:** A feedforward artificial neural network with two hidden layers consisting of 128 and 64 neurons, respectively. The `ReLU` (Rectified Linear Unit) activation function was applied to introduce nonlinearity, enabling the network to learn complex patterns. ReLU is defined as $f(x) = \max(0, x)$ and is widely used due to its simplicity and efficiency in mitigating vanishing gradients [13]. The model was trained using the Adam optimizer with an adaptive learning rate. Early stopping was enabled to terminate training once validation performance ceased to improve, helping prevent overfitting.

- **RandomForestClassifier:** An ensemble learning method based on aggregating predictions from multiple decision trees. Each tree is trained on a random subset of samples and features, a technique known as bagging. This method reduces variance and increases generalization. The model is well-suited for high-dimensional data and provides built-in feature importance metrics [16].

- **LogisticRegression:** A linear classification model that estimates class probabilities using the logistic function. It is defined as $P(y = 1|x) = \frac{1}{1+e^{-w^T x}}$. While not capable of modeling complex interactions, logistic regression is efficient and highly interpretable. In this project, class balancing was performed using the `class_weight="balanced"` option to mitigate the effect of class imbalance [21].

- **DecisionTreeClassifier:** A non-parametric model that partitions data based on a hierarchy of feature thresholds. Although highly interpretable, individual decision trees are prone to overfitting, especially on noisy or imbalanced data. The decision rules learned by the tree provide insight into model logic and thresholds.

- **KNeighborsClassifier:** A distance-based, non-parametric method that classifies a sample based on the majority label of its k nearest neighbors in the training data. Distance was computed using the Euclidean metric. While conceptually simple, KNN scales poorly to large datasets and requires careful feature normalization, which was addressed using MinMaxScaler.

8.2 Evaluation Metrics

Model performance was evaluated using multiple metrics:

- **Accuracy:** The proportion of correct predictions across all classes.

- **Precision, Recall, F1-Score:** These were computed for each class and macro-averaged. Precision quantifies the proportion of true positives among all predicted positives, recall reflects the proportion of actual positives correctly identified, and F1-score is their harmonic mean. The macro-F1 score equally weights each class, making it suitable for imbalanced datasets [28].

- **Confusion Matrix:** A heatmap showing the number of true and false predictions per class. This visual tool highlights which classes are being confused and helps identify systematic errors.

8.3 Explainability with SHAP

To interpret the decisions of complex models like MLP and Logistic Regression, SHAP (SHapley Additive exPlanations) values were computed. SHAP assigns each feature an importance value for a particular prediction by considering all possible combinations of feature inclusion [4]. It satisfies properties of local accuracy and consistency, making it suitable for explaining both individual and global model behavior.

Summary plots and decision plots were generated to visualize the impact of top features. These insights are crucial for verifying that the models are not basing predictions on misleading or irrelevant attributes.

8.4 Implementation Details

All code was implemented in Python using the following open-source libraries:

- `pandas, numpy`: for data manipulation and numerical operations.
- `scikit-learn`: for machine learning models and preprocessing utilities.
- `matplotlib, seaborn`: for visualization of confusion matrices and feature distributions.
- `shap`: for model interpretability.
- `joblib`: for model persistence and reuse.

Training and evaluation were conducted on a local machine with 16GB RAM and a standard Intel i7 processor. The pipeline was modularized to allow easy replication and parameter tuning. All trained models were serialized and saved for future experimentation or deployment.

9. Evaluation and Results

To assess the performance and limitations of each model, a comprehensive evaluation was conducted using a combination of accuracy, macro-averaged F1-score, per-class precision and recall, confusion matrices, and SHAP-based interpretability analysis. The diversity and imbalance of the CIC-IDS-2017 dataset presented unique challenges, especially for rare attack classes.

9.1 Accuracy and Macro-F1 Score

The **Multilayer Perceptron (MLP)** demonstrated high overall accuracy of 99.6% and a macro-averaged F1-score of 0.83. Accuracy measures the proportion of correctly classified instances over all predictions, but it can be misleading in imbalanced settings where dominant classes skew the results. In contrast, the macro-F1 score gives equal weight to each class by averaging the F1-scores computed per class, thus reflecting true multiclass performance [28].

Despite the strong aggregate metrics, the MLP underperformed on underrepresented classes such as `Web Attack -- Sql Injection`, likely due to insufficient representation in the training data and overlapping feature distributions with benign traffic.

The **Random Forest** model achieved the highest macro-F1 score of 0.87 and handled minority classes more effectively than other classifiers. Its ensemble nature and internal handling of feature interactions enabled better generalization, particularly in detecting nuanced patterns of rare attacks [16].

9.2 Comparison of Other Classifiers

Logistic Regression, while efficient and interpretable, struggled with non-linear class boundaries and yielded notably low precision and recall on minority classes such as `Web Attack -- XSS`. This is expected due to its linear decision surface and limited representational power [21].

Decision Trees exhibited moderate performance. Their interpretability made them useful for understanding which features led to specific predictions. However, the trees occasionally overfit branches when trained on undersampled data, especially for classes with few examples.

K-Nearest Neighbors (KNN) produced reasonable classification results but had two major drawbacks. First, it was sensitive to the feature scaling method. Second, inference was computationally intensive, as the model performs lazy learning and stores all training samples for distance comparisons during prediction.

9.3 Confusion Matrices and Per-Class Analysis

Confusion matrices offered a visual representation of true and false predictions for each class. They revealed that classes such as `Bot`, `Infiltration`, and `Web Attack -- Brute Force` were commonly misclassified, often confused with more frequent attack types or benign traffic. These errors highlight potential limitations in feature space separability and suggest the need for data augmentation, advanced sampling methods, or deeper feature engineering.

In particular, infiltration attacks were often missed due to their behavioral similarity to legitimate internal traffic. This reflects a broader challenge in intrusion detection: distinguishing insider threats or stealthy movements from normal operations.

9.4 SHAP-Based Interpretability

To better understand the model decisions, SHAP (SHapley Additive exPlanations) was applied to the MLP and Logistic Regression classifiers. SHAP computes the contribution of each feature to a given prediction based on game theory, offering model-agnostic explanations that satisfy consistency and local accuracy properties [4].

The analysis identified several features with high impact across multiple samples:

- **Flow Duration** – longer or shorter connection durations were indicative of specific attack types (e.g., `DoS Slowloris`).
- **Fwd Packet Length Mean** – reflected the average size of packets sent in the forward direction, which varied significantly between flooding attacks and benign sessions.
- **PSH Flag Count** – a TCP control flag indicative of push behavior; certain attack tools manipulate this flag for evasion or overload purposes.

- **Bwd Packet Length Min** – minimum packet size in the reverse direction; low values were associated with scanning or probing activity.

These insights are crucial not only for model validation but also for real-world deployment, where human analysts may require transparent rationale for automated decisions.

9.5 Summary of Findings

The Random Forest classifier emerged as the most effective model overall, balancing accuracy, class-wise fairness, and robustness. However, even the best models struggled with underrepresented classes. This underscores the importance of combining algorithmic strength with domain-informed preprocessing, feature engineering, and advanced sampling strategies.

Moreover, SHAP-based interpretability proved valuable in demystifying black-box models, allowing for actionable understanding of classification outcomes. These explanations also provide a foundation for future improvements in feature selection and model design.

10. Model Comparison

To provide a systematic and interpretable comparison of all implemented classifiers, both quantitative metrics and qualitative visualization tools were employed. The two primary performance indicators selected for cross-model comparison were **accuracy** and **macro-averaged F1 score**. While accuracy measures the proportion of correctly classified instances across all classes, macro-F1 provides a balanced view by averaging the F1-scores of each class independently, thereby mitigating the impact of class imbalance [28].

10.1 Quantitative Performance

As illustrated in Figure 2, the **Random Forest** classifier demonstrated superior performance in both metrics. It achieved the highest macro-F1 score and maintained consistently high accuracy, making it the most reliable model for imbalanced multiclass intrusion detection. In contrast, **Logistic Regression** displayed the weakest performance across the board, highlighting its limitations in capturing nonlinear and high-dimensional relationships in the CIC-IDS-2017 feature space [21].

The **MLPClassifier**, although slightly trailing Random Forest in macro-F1, exhibited competitive performance and demonstrated strong generalization ability. Its capacity to model complex nonlinear decision boundaries gave it an advantage, particularly for attack classes with overlapping distributions or subtle statistical signatures.

10.2 SHAP-Based Explainability Insights

To complement performance metrics, SHAP (SHapley Additive exPlanations) was used to compare the internal decision-making logic of selected models, particularly MLP and Logistic Regression. SHAP analysis revealed that the MLP leveraged a broader range of features and captured subtle interactions between them, while Logistic Regression relied primarily on features with the highest global variance [4]. This highlights the benefit of increased model complexity in scenarios requiring nuanced classification, such as distinguishing between `DoS Hulk` and `DDoS`, or between `Web Attack -- XSS` and benign web activity.

10.3 Confusion Matrix Observations

Confusion matrices were plotted for all classifiers to examine error patterns across all 15 classes. These heatmaps showed that even models with high overall accuracy, such as Random Forest and MLP, tended to misclassify underrepresented categories like `Heartbleed` or `Infiltration`. These findings reinforced the decision to prioritize macro-F1 as the central evaluation metric, since it penalizes classifiers for poor minority-class performance, even when the overall accuracy remains high.

10.4 Visualization and Interpretation

To support transparent and intuitive understanding, a comparative bar chart was created using `matplotlib`. This visualization, presented in Figure 2, summarizes the average performance of all five classifiers and highlights the gaps in generalization performance between simple and complex models.

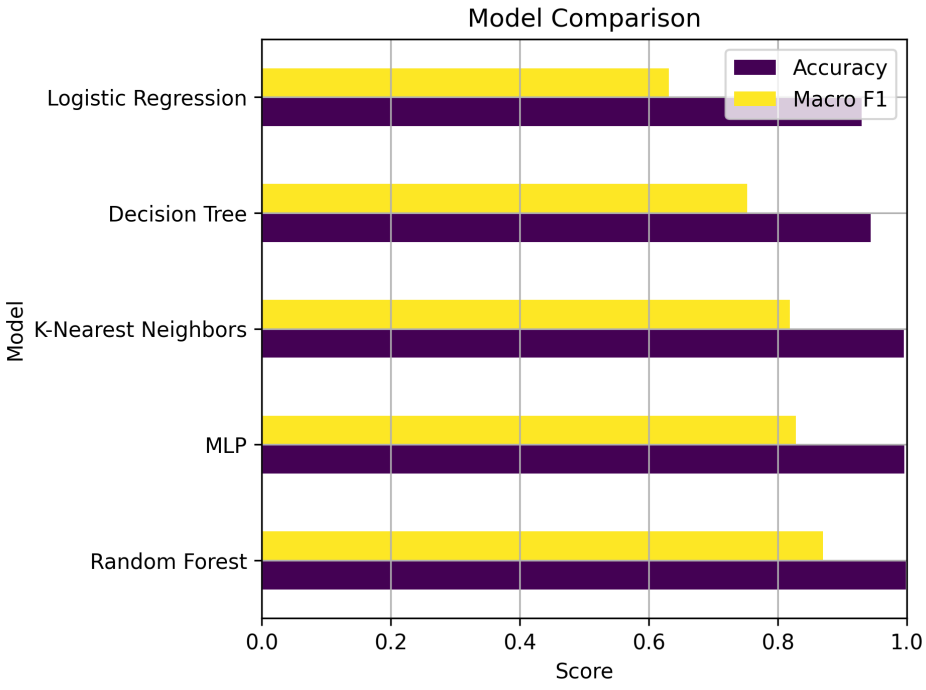

Figure 1: Comparison of model performance across accuracy and macro-F1 score. Random Forest leads in both metrics.

10.5 Summary

In conclusion, the comparative analysis demonstrated that while simple models like Logistic Regression offer interpretability and computational efficiency, they fall short in multiclass intrusion scenarios. More complex models like Random Forest and MLP not only achieve higher accuracy and F1 scores but also provide meaningful feature attribution when paired with SHAP analysis. This underscores the importance of balancing performance with interpretability when designing machine learning-based network intrusion detection systems.

11. Experimental Results and Visualizations

This section presents the empirical evaluation of all implemented classifiers using confusion matrices and SHAP interpretability visualizations. The visual tools are critical not only for quantifying performance but also for understanding the nature of errors and feature contributions, which are essential for trust and reliability in network intrusion detection systems [22].

11.1 Confusion Matrices and Classification Reports

The confusion matrix provides a class-wise breakdown of true versus predicted labels, highlighting the model's precision and recall for each attack type. It is especially useful for identifying misclassification patterns, such as systematic confusion between similarly structured attacks (e.g., `DoS Hulk` and `DDoS`) or challenges with underrepresented classes.

Figure 2 to Figure 6 show the confusion matrices for all five classifiers. Each matrix is normalized to reflect per-class percentages, facilitating direct comparison between minority and majority classes.

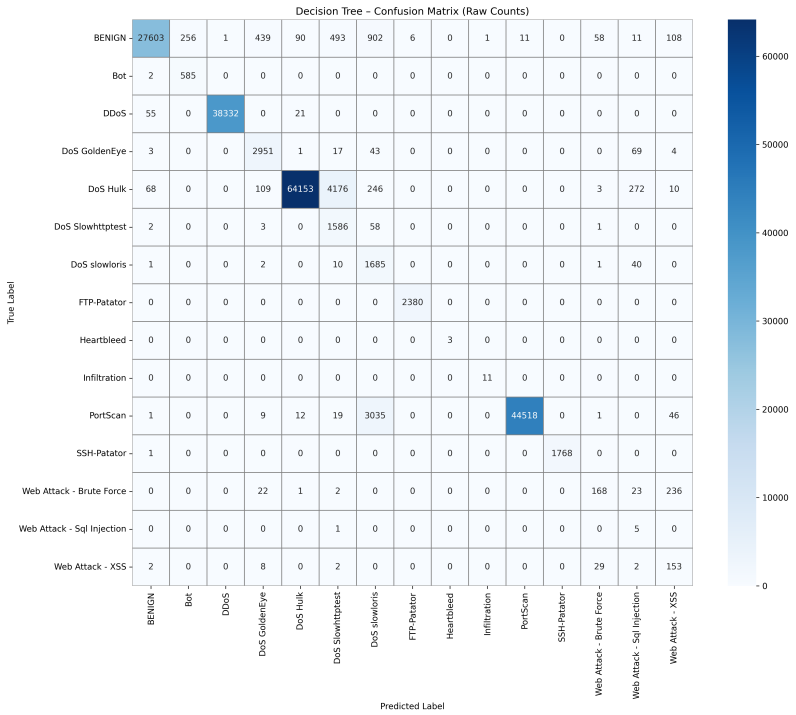

Figure 2: Confusion matrix for the Decision Tree classifier. Visible overfitting and misclassifications in low-support classes.

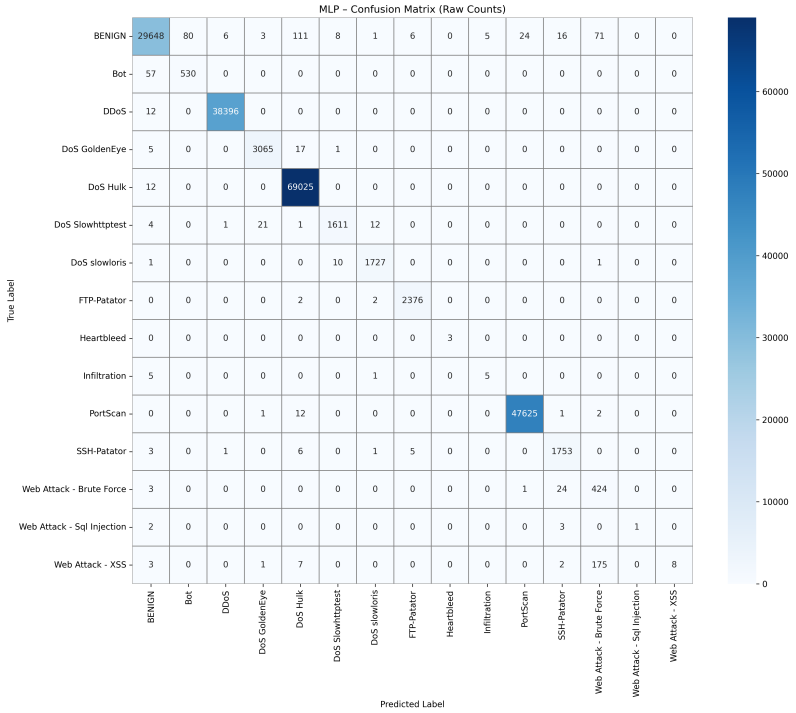

Figure 3: Confusion matrix for the MLPClassifier (Multi-Layer Perceptron). Demonstrates strong performance on most classes with minimal confusion.

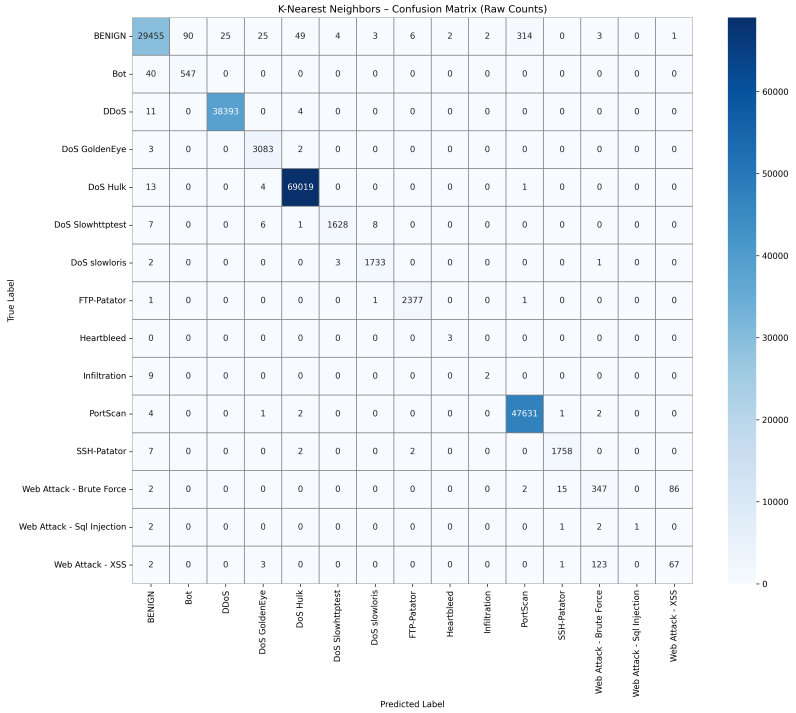

Figure 4: Confusion matrix for K-Nearest Neighbors classifier. Note increased confusion due to lack of parametric generalization.

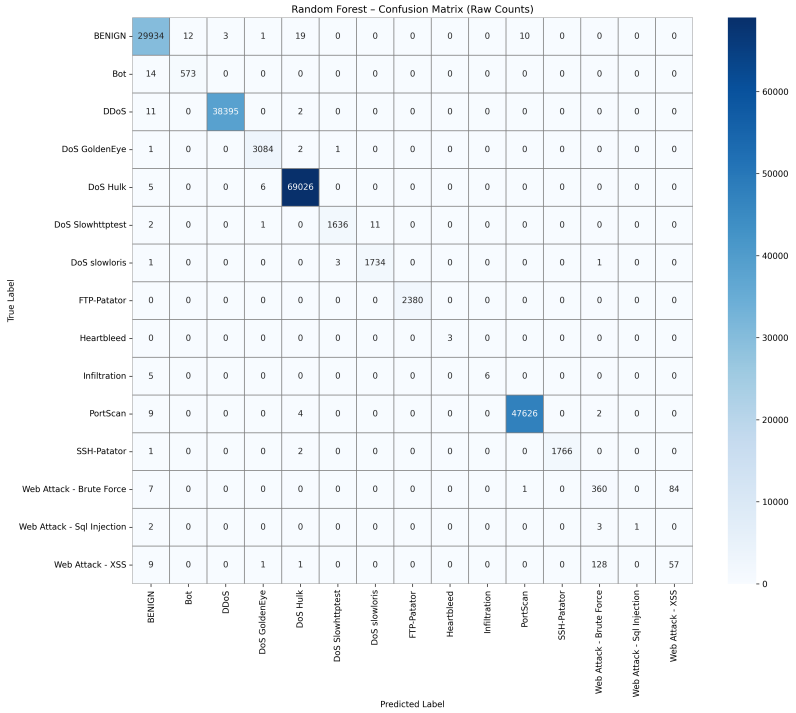

Figure 5: Confusion matrix for Random Forest classifier. Consistently high accuracy across nearly all attack categories.

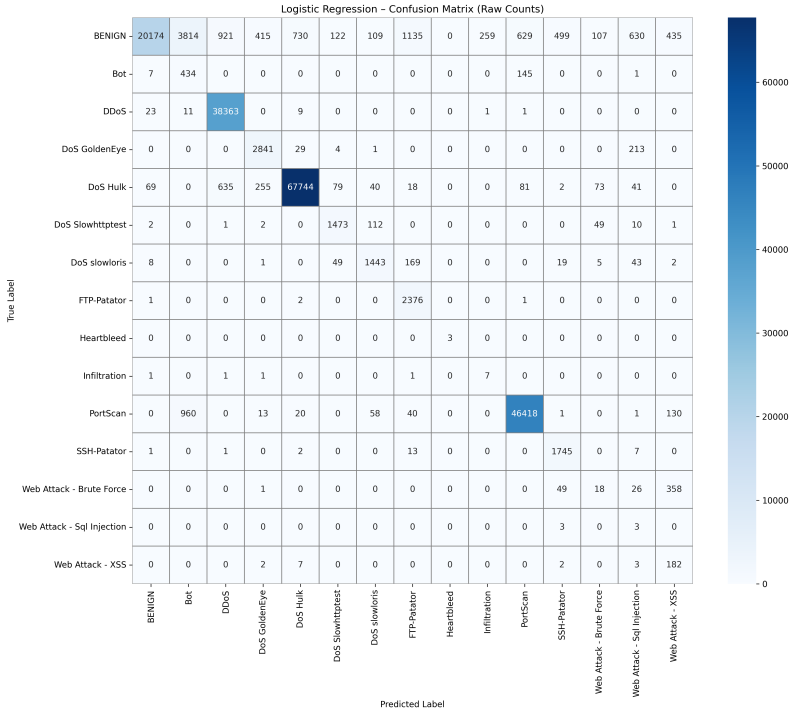

Figure 6: Confusion matrix for Logistic Regression. Struggles with nonlinear boundaries and misclassifies several minority classes.

11.2 SHAP Value Interpretability Analysis

Model explainability is increasingly critical in cybersecurity to validate and audit model decisions [15]. To this end, SHAP (SHapley Additive exPlanations) was applied to interpret the decision-making processes of the MLP and Logistic Regression models. SHAP values represent the contribution of each feature to the model's prediction for each sample [4].

The SHAP summary plots (Figures 7 and 8) illustrate the top contributing features across all classes, ranked by their average absolute SHAP value. For the MLP, features like `Flow Duration`, `PSH Flag Count`, and `Bwd Packet Length Min` showed the highest impact. This suggests that temporal and behavioral characteristics are more important than simple byte-level statistics for deep models.

In contrast, Logistic Regression relied more heavily on a narrower set of features with high global variance, such as `Fwd Packet Length Mean`. This aligns with expectations for linear models, which lack the capacity to capture interactions between low-variance features.

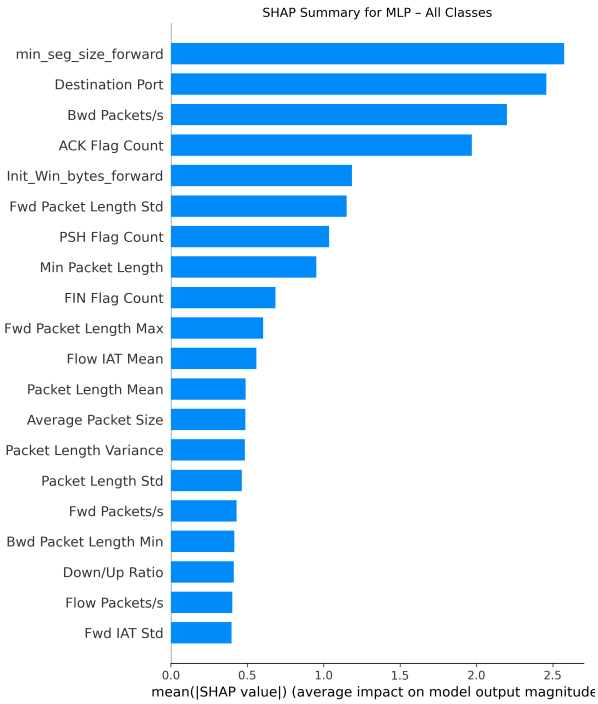

Figure 7: SHAP summary plot for MLPClassifier: most influential features across all classes.

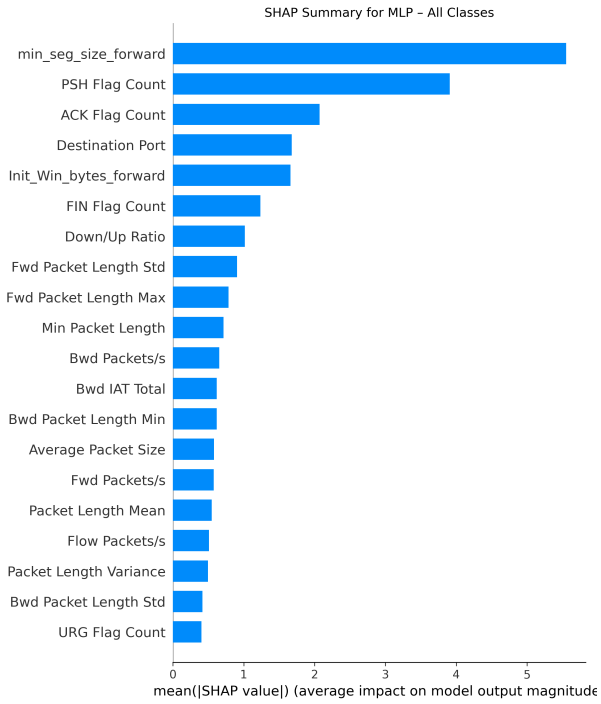

Figure 8: SHAP summary plot for Logistic Regression: feature importance from linear model explanation.

These interpretability insights reinforce the importance of well-engineered features and support the use of complex models for nuanced threat detection in modern enterprise environments.

12. Future Work and Improvements

While the current models demonstrate strong performance on several metrics, there is substantial room for improvement in terms of generalization, adaptability, and detection accuracy—particularly on rare or stealthy attack types. Future work may proceed in the following directions:

1. **Feature Engineering:** Develop higher-order or domain-specific features, including protocol-aware attributes, entropy-based indicators, or session-level aggregations, to better capture subtle differences between benign and malicious behavior [20].

2. **Data Augmentation:** Address class imbalance using synthetic oversampling techniques such as SMOTE (Synthetic Minority Over-sampling Technique) [29] or generative models like GANs, which can create realistic yet diverse synthetic samples to improve minority class generalization [30].

3. **Advanced Models:** Apply gradient boosting frameworks like LightGBM or deep tabular models like TabNet that are optimized for structured data and often yield superior performance compared to conventional classifiers [31, 32].

4. **Time-Series Context:** Instead of static session-level features, use sequential models like LSTM (Long Short-Term Memory) or 1D Convolutional Neural Networks to

exploit temporal dependencies across packets or flows, which may improve detection of multi-stage or persistent threats [33].

5. **Online Learning:** Implement online or incremental learning algorithms to adapt to changing network behavior and newly emerging attack patterns. Techniques such as Hoeffding Trees or Online Bagging can enable real-time learning in dynamic environments [34].

In addition, future versions could integrate adversarial robustness testing to evaluate model reliability against evasion attacks and data poisoning scenarios [35].

13. Conclusion

This project presented a comprehensive approach to constructing and evaluating a multi-class intrusion detection system using machine learning techniques on the CIC-IDS-2017 dataset. Through rigorous preprocessing, class balancing, and consistent evaluation protocols, the study benchmarked a range of classifiers under identical conditions. Random Forest demonstrated superior performance in terms of macro-F1 and accuracy, while the Multilayer Perceptron (MLP) provided competitive results and captured complex non-linear patterns effectively.

Model explainability was addressed using SHAP (SHapley Additive exPlanations), which offered transparency into the decision-making process by highlighting the most influential features. This interpretability is crucial in security-sensitive applications where understanding model behavior can support trust, debugging, and regulatory compliance [4].

Despite the promising results, challenges persist—particularly in detecting rare or stealthy attack classes, where class imbalance and feature ambiguity hinder performance. This motivates future work in synthetic data generation [29], cost-sensitive learning, and deeper temporal modeling using architectures like LSTM [33].

Ultimately, the findings reinforce the viability of machine learning-based intrusion detection systems in modern network environments. However, they also highlight the need for carefully curated datasets, interpretable modeling strategies, and robust feature engineering to ensure real-world applicability in evolving threat landscapes [20].

References

[1] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST Special Publication*, vol. 800, no. 94, 2007.

[2] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1, pp. 18–28, 2009.

[3] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," *2010 IEEE Symposium on Security and Privacy*, pp. 305–316, 2010.

[4] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[5] M. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *ICISSP*, pp. 108–116, 2018.

[6] G. Haixiang, G. Yijing, D. Haoran, H. Yanan, and L. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[7] A. H. Lashkari, G. Draper Gil, M. A. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP)*, 2017.

[8] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, 2015.

[9] O. Foundation, "Owasp top ten web application security risks," 2025, https://owasp.org/www-project-top-ten/.

[10] Z. Durumeric, M. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, V. Paxson, and M. Payer, "The matter of heartbleed," *Proceedings of the 2014 Conference on Internet Measurement*, pp. 475–488, 2014.

[11] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[12] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *ICML*, 2010.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[15] A. B. Arrieta *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[16] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] P. et al., "Scikit-learn: Machine learning in python," 2011, https://scikit-learn.org/.

[18] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[19] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special issue on learning from imbalanced data sets," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[20] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147–167, 2019.

[21] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013.

[22] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[23] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[24] Y. Zhang, L. Wang, H. Wang, and Y. Niu, "Data preprocessing in intrusion detection: A review," *IEEE Access*, vol. 7, pp. 53 076–53 088, 2019.

[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT Press, 2018.

[26] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[27] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, pp. 1157–1182, 2003.

[28] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[29] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[30] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[31] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[32] S. O. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 6679–6687, 2021.

[33] G. Kim and S.-W. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2016, pp. 1–5.

[34] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[35] N. Papernot, P. McDaniel, and I. Goodfellow, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 506–519.