# A Machine Learning Pipeline for DDoS Attack Detection in Network Traffic

## Abstract

Distributed Denial of Service (DDoS) attacks pose a serious threat to network availability, necessitating accurate and efficient detection systems. This paper presents a comprehensive machine learning pipeline for DDoS detection using the CIC-IDS2017 network intrusion dataset. The proposed system performs extensive data preprocessing, including exploratory data analysis (EDA), outlier detection and capping, class imbalance handling, and domain-specific feature engineering. We evaluate multiple classifiers – Random Forest, Decision Tree, k-Nearest Neighbors (KNN), and Support Vector Machine (SVM) – using a curated feature set of 10 optimal features selected via statistical F-test. Experimental results show that the ensemble Random Forest model achieves the best performance with 98.14% F1-score, 98.15% accuracy, 97.96% precision, and 98.33% recall on the test set. The system demonstrates a high DDoS detection rate with a low false alarm rate, outperforming the other evaluated models. We discuss the impact of each stage of the pipeline, highlighting how EDA-driven preprocessing and feature engineering improved detection efficacy. The paper concludes with insights into the strengths of the proposed approach and outlines future work, including integration of deep learning and real-time deployment for enhanced DDoS mitigation.

## Introduction

DDoS attacks are malicious attempts to disrupt online services by overwhelming networks or servers with illegitimate traffic. These attacks can cripple critical infrastructure and cause substantial economic damage if not detected early. Traditional signature-based intrusion detection systems struggle to identify novel or evolving DDoS patterns. In response, anomaly-based detection using machine learning has gained traction, as it can learn to recognize deviations from normal traffic behavior without relying solely on known signatures. However, applying machine learning to network intrusion detection presents challenges such as high-dimensional feature spaces, class imbalance between benign and attack traffic, and the presence of noise or outliers in real-world data.

In this work, we develop a **DDoS Detection System** that leverages a machine learning pipeline to overcome these challenges. The system is evaluated on the CIC-IDS2017 dataset, a modern intrusion detection benchmark containing benign traffic and a variety of attacks (Brute Force, DoS, DDoS, infiltration, botnet, etc.). The dataset comprises 2,830,743 network flow records with 79 features per flow, collected over a week of simulated network activity. Notably, about 19.7% of the flows are attack traffic (including DDoS) while 80.3% are benign, indicating a significant class imbalance typical of real network data.

Our approach emphasizes end-to-end data analysis and preprocessing before modeling. We conduct thorough EDA to identify data quality issues, feature distributions, and correlations. Based on the EDA findings, we apply targeted preprocessing: we cap extreme outliers to reduce skew without removing data, impute the rare missing values, engineer new domain-specific features (e.g., traffic volume ratios and flow asymmetry metrics), and address class imbalance by controlled undersampling. We then select the most informative features and train multiple classification algorithms to detect DDoS attacks (framed as a binary classification of attack vs. benign traffic). The goals are to achieve high detection accuracy and precision (minimizing false negatives and false alarms), while maintaining a pipeline efficient enough for potential real-time deployment.

The remainder of this paper is organized as follows. Section II reviews related work on intrusion and DDoS detection using machine learning. Section III describes the dataset and its characteristics. Section IV details our methodology, including preprocessing steps, feature engineering, and model training. Section V presents experimental results, comparing classifier performance with various metrics and visualizations. Section VI provides a discussion of the results and the system’s implications. Finally, Section VII concludes the paper and outlines future work.

## Related Work

Network intrusion detection has been widely studied using machine learning and deep learning techniques. Early work often focused on benchmark datasets like KDD Cup 1999 and NSL-KDD, but these are now considered outdated and less reflective of modern attacks. The CIC-IDS2017 dataset used in this study was introduced by Sharafaldin et al. in 2018 to address these shortcomings, providing more recent and diverse attack scenarios including DDoS. Several researchers have evaluated classical algorithms on this dataset. Belouch et al. (2018) compared SVM, Naïve Bayes, Random Forest, and Decision Tree for intrusion detection, and found that ensemble methods like Random Forest achieved the highest accuracy (around 98-99%) on CIC-IDS2017. Their study underscored the effectiveness of tree-based ensembles in handling the mixed numeric and categorical features of network traffic data.

**Feature selection and ensemble strategies** have been particularly effective in improving intrusion detection performance. Taher et al. (2019) proposed a supervised learning approach with an intelligent feature selection stage using Information Gain and evolutionary algorithms (PSO and GA) before classification. By combining KNN and Random Forest in an ensemble, their system achieved up to 99.96% detection accuracy on CIC-IDS2017 – nearly perfect performance – highlighting the value of dimensionality reduction and ensemble learning for this problem. Injadat et al. (2020) developed a multi-stage machine learning framework for network intrusion detection, which optimized feature selection and model tuning in phases. They reported around 99% accuracy and 99% recall on CIC-IDS2017 using a hybrid KNN+RF model, while employing techniques like SMOTE to handle class imbalance. These works demonstrate that with appropriate feature engineering and balancing, traditional machine learning models can attain very high detection rates on contemporary intrusion data.

Deep learning approaches have also been applied to DDoS and intrusion detection, often to capture complex patterns in network traffic. For example, Abdel-Basset et al. (2021) introduced a deep neural network with a traffic attention mechanism and residual architecture to detect intrusions. Their model achieved about 99.6% accuracy on CIC-IDS2017, but the precision was lower (around 92%) indicating false alarms, and the method incurred high computational cost due to its complexity. Similarly, Dora and Lakshmi (2022) implemented a meta-heuristic optimized LSTM model for DDoS attack detection, combining convolutional feature learning with sequential analysis. While such deep learning models can approach or exceed the accuracy of classic methods, they often require more resources and can be less interpretable. Recent surveys (e.g., Gupta et al., 2022) emphasize that simpler algorithms, if well-optimized, can perform on par with deep networks for intrusion detection, and are easier to deploy on resource-constrained systems.

In summary, related work suggests that: (1) Ensemble classifiers (especially Random Forest) are top performers for intrusion and DDoS detection in structured flow datasets, (2) Feature selection or extraction significantly boosts performance and efficiency, (3) Handling class imbalance (through resampling or cost-sensitive methods) is critical for detecting minority attack instances, and (4) Deep learning offers high capacity but must be carefully balanced against complexity and interpretability. Our work builds on these insights by implementing a full pipeline that incorporates extensive preprocessing and feature engineering, aiming to harness the high accuracy of ensemble models while ensuring the solution remains explainable and deployment-friendly.

## Dataset Description

We use the **CIC-IDS2017** dataset for training and evaluating our DDoS detection system. This dataset was created by the Canadian Institute for Cybersecurity to reflect modern benign and attack traffic in a real-world network environment. It contains traffic captured over five days in a closed network, with each record representing a network flow (aggregated IP traffic session) and labeled as either benign or with a specific attack type. The dataset includes seven common attack categories: Brute Force (SSH/FTP password cracking), DoS, DDoS, Web Attacks, Infiltration, Botnet, and Heartbleed, in addition to normal traffic. The DDoS attacks in particular were executed on the last day (Friday) of the capture period, mixed with normal background traffic, to simulate a realistic attack scenario.

**Data Volume and Features:** In total, the dataset comprises approximately **2.83 million flow records** with **79 features** per flow. These features were generated using CICFlowMeter and include a rich set of network traffic characteristics. Examples include basic IP header info (source/destination IPs and ports, protocol), time-related features (flow duration, inter-arrival times), packet counts and byte volumes in forward/backward directions, statistical metrics on packet lengths (min, max, mean, standard deviation), and flags/codes from protocols. One feature is categorical: the **Label** indicating the class of the flow (e.g., "BENIGN" or the type of attack). All other features are numeric (integer or float). A brief overview of the dataset is shown in Table I.

**Class Distribution:** The dataset is highly imbalanced, reflecting that attacks are rarer than normal traffic in real networks. About **80.3%** of the flows (2,273,097 records) are labeled BENIGN, while the remaining **19.7%** (557,646 records) are attacks of various types. In a binary context (benign vs. attack), this is roughly a 4:1 imbalance. Among attacks, the largest single category is DDoS, with over 128,000 flows, and other categories range from a few hundred to tens of thousands of instances. The imbalance is even more severe for certain specific attacks (e.g., Heartbleed appears only in a handful of flows). This imbalance poses a challenge for modeling, as classifiers could bias toward the majority class if trained naively.

**Data Quality:** Overall data quality is high – there are very few missing values and all features are well-defined. A minor exception is the feature Flow Bytes/s, which has a small number of missing entries (about 1.3k flows, or 0.05%) due to divisions by zero when calculating byte rates. We also observed that some features have **extreme values and outliers**; for instance, features like packet length and flow byte count can span several orders of magnitude. Without handling, these outliers could skew scaling and classifier training. Additionally, many features are correlated (e.g., total packet counts and byte counts, forward and backward packet lengths) because they capture related aspects of the same flows. Redundant features may not harm accuracy but can increase model complexity unnecessarily.

**Labeling and Objective:** In this paper, we focus on binary classification: distinguishing attack traffic (of any type) from benign traffic. This approach aligns with a DDoS detection goal – flagging suspicious flows – though our system in principle detects all attack types collectively. The label "BENIGN" is treated as class 0 and any attack label as class 1. This simplifies the task to a two-class problem, which is common in intrusion detection research to evaluate detection vs. false alarm trade-offs. However, our feature engineering and model training leverage the full data and labels; for example, when examining feature importance, we consider that some features might be more indicative of certain attacks (like DDoS) than others.

**Data Access:** The CIC-IDS2017 data was originally released as a collection of CSV files (one per day or attack scenario). In our implementation, we automated the download and integration of these files using a Kaggle API wrapper. All CSVs were concatenated into one master dataset of 2.83M rows, and basic integrity checks (like ensuring consistent columns and expected record counts per file) were performed during loading. After loading, we saved summary information such as dataset shape, feature names, and class distribution for reference.

Table I below summarizes key dataset characteristics after integration:

| **Metric** | **Value** |
| --- | --- |
| Total Flows (Records) | 2,830,743 |
| Number of Features | 79 (78 numeric + 1 categorical Label) |
| Total Attack Flows | 557,646 (19.7%) |
| Total Benign Flows | 2,273,097 (80.3%) |
| Attack Types Included | Brute Force, DoS, DDoS, Infiltration, Botnet, Web Attack, Heartbleed |
| Missing Values | Present in 1 feature (Flow Bytes/s: ~0.05% flows) |
| Data Volume | ~1.83 GB (raw CSV total size) |

This dataset provides a robust basis for training and evaluating the DDoS detection pipeline, given its size, diversity of attacks, and realistic traffic profile.

## Methodology

Our DDoS detection approach is implemented as a multi-step machine learning pipeline. The pipeline encompasses data preprocessing, feature engineering, feature selection, and model training/evaluation. In this section, we describe each stage of the methodology in detail, highlighting how domain knowledge and EDA findings inform the design.

### Preprocessing and EDA-Driven Data Cleaning

**1. Data Loading and Integration:** The raw dataset is composed of multiple parts, so we first load all flow records into a single pandas DataFrame. During loading, we print metadata for each file (number of records, label distribution per file) to verify completeness. After concatenation, the unified dataset has 2,830,743 rows and 79 columns (as noted in Section III). We then performed an initial **Exploratory Data Analysis (EDA)** to understand the data characteristics.

**2. Exploratory Data Analysis:** EDA is crucial for identifying potential issues and guiding preprocessing decisions. Key EDA steps included: - Dataset Overview: We confirmed the dataset shape and memory usage (~1.83 GB). Data types were mostly numeric (54 integer, 24 float features) with one object label. - Missing Values: Only the Flow Bytes/s feature had missing values (1,358 instances, about 0.048% of all flows). No other feature had missing data. Such a low missing rate suggested that simple imputation would suffice without materially impacting distributions. - Class Distribution: We computed the distribution of the Label. As expected, benign flows dominate (≈2.27M) compared to attacks (≈0.56M), about a 4:1 ratio. We further aggregated all attack labels into a binary indicator (attack vs. benign) and found 80.3% vs 19.7% split. The largest attack category was DDoS (roughly 4.5% of all flows), followed by PortScan and other DoS attacks. This confirmed the need for class balancing in training. - Numeric Feature Distributions: We examined summary statistics of numeric features (min, max, mean, standard deviation). We discovered that several features have **very large ranges** (36 features had range > 1e6) and some have **zero variance** in this dataset (8 features were constant, e.g., perhaps flow ID fields). A small number of features had negative values (e.g., certain delta times can be negative due to how they are computed), which is normal and was handled appropriately by our pipeline. - Outlier Analysis: Using interquartile range (IQR), we identified that on average 16.43% of the data in numeric features were outside the 1.5×IQR range – a significant fraction, indicating heavy-tailed distributions. The top 5 features with the highest outlier rates were: Fwd Packet Length Max (23.46% of flows considered outliers in this feature), Fwd Packet Length Std (23.46%), Bwd Packet Length Std (23.11%), Destination Port (22.16%), and Flow Bytes/s. Many of these relate to packet sizes or byte rates, which can vary widely in attack traffic. This finding suggested that outlier handling would be important to prevent extreme values from skewing model training. - Correlation Analysis: We computed a correlation matrix for a sample of 10,000 flows across the main numerical features. Many features showed high pairwise correlations. We found **16 pairs of features** with Pearson correlation > 0.90. For example, Total Length of Bwd Packets was 99.97% correlated with Total Backward Packets (since total packet count and total bytes in backward direction are obviously related). Similarly, features capturing packet timing (Flow Duration and Fwd IAT Total) were >99% correlated, and there were several such redundancies. This indicated that we should remove or consolidate highly correlated features to reduce multicollinearity and model complexity.

The EDA insights drove the subsequent preprocessing steps: - We decided to **cap outliers** rather than remove them, to retain information while reducing skew. - We planned to **drop zero-variance features** (since they add no information). - We aimed to **remove or avoid highly correlated features** during feature selection, to prevent over-emphasizing essentially duplicated signals. - We confirmed that **median imputation** would suffice for the minor missing data issue. - We noted the need for **feature scaling**, given the wide ranges, before feeding data to distance-based learners (KNN, SVM).

**3. Outlier Detection and Capping:** Based on the outlier analysis, we implemented an IQR-based capping for numeric features. For each numeric feature, we calculated the 25th percentile (Q1) and 75th percentile (Q3), and defined the IQR = Q3 – Q1. Any data point below Q1 – 1.5IQR or above Q3 + 1.5IQR is considered an outlier. Instead of discarding these points, we **capped** them to the boundary values (i.e., set values above the upper bound to exactly Q3+1.5IQR, and below the lower bound to Q1–1.5IQR). This strategy retains each flow in the dataset but mitigates extreme deviations. Capping was applied to all numeric features. We also tested an alternative z-score method (capping beyond 3 standard deviations) for comparison, but the IQR method was more robust given the non-normal distributions. After capping, the data still maintains variance but with far fewer extreme values (the average outlier rate effectively drops to 0% by definition after capping). This prevents models like KNN or SVM from being overly influenced by outlier scales. It’s worth noting that about one-sixth of the data had some feature adjusted by capping, which underscores the importance of this step.

**4. Missing and Infinite Values:** The only feature with NaNs was Flow Bytes/s. We filled its missing entries with the **median** of that feature (computed on non-missing values). Median imputation was chosen because it is robust to outliers and would not be unduly influenced by the heavy tail in that feature. Also, after outlier capping, the median remained representative of a typical value. Any infinite values (which can occur in rate features if divided by very small durations) were also replaced with NaN and then imputed similarly. Categorical columns (none except Label, which was complete) would be imputed with the mode if needed. After this step, the dataset had no missing or infinite values remaining.

**5. Class Label Encoding and Balancing:** We added a binary label column Binary\_Label where 0 = BENIGN and 1 = ATTACK. This facilitated training binary classifiers. Given the 4:1 class imbalance, we employed **undersampling** of the majority class (benign) to balance the training data. We randomly sampled an equal number of benign and attack flows for training, with a cap to avoid excessively large training sets. Specifically, we undersampled to at most 100,000 instances from each class (the attack class had ~557k instances, so we took 100k from each). This yielded a balanced subset of 200,000 flows. This strategy ensures the classifier sees equal benign and attack examples, which generally improves its recall on attacks and keeps it from naively predicting everything as benign. We preserved a separate test set with the same 50:50 class ratio (by stratified splitting, described in Section V). While undersampling sacrifices some benign data, 100k benign flows still capture a wide variety of normal behavior, and it dramatically reduces training time. An alternative could be oversampling the attack class (e.g., via SMOTE), but we chose undersampling for simplicity and to avoid synthetic data – also the benign class was so large that using all of it was computationally unnecessary.

After these preprocessing steps, the data was in a much cleaner and more tractable form: outliers moderated, no missing values, balanced classes, and consistent scaling to be applied next. We emphasize that all preprocessing decisions were backed by the initial EDA (hence EDA-driven preprocessing). This approach ensures that we address real issues present in the data and avoid arbitrary or superfluous transformations.

### Domain-Specific Feature Engineering

Raw network flow features, while extensive, can be augmented with domain knowledge to improve attack detection. We engineered additional features capturing ratios, rates, and differences that are known to characterize traffic anomalies like DDoS attacks. In total, we created **17 new features** derived from the existing columns (bringing the interim feature count to 96 before selection). Key engineering steps included:

* **Traffic Ratio Features:** We formed ratios between related forward and backward traffic measures. For example, Fwd Packets/s\_to\_Bwd Packets/s\_ratio = (Forward packets per second) / (Backward packets per second + 1e-6). This ratio can highlight asymmetric flows – in a DDoS attack, we might see a high forward packet rate from attacker to victim with little response (backward traffic), yielding a high ratio. We created similar ratios such as Flow Bytes/s\_to\_Flow Packets/s\_ratio (average bytes per packet in the flow), Fwd Packet Length Max\_to\_Min\_ratio, Bwd Packet Length Max\_to\_Min\_ratio, Total Fwd Packets\_to\_Total Backward Packets\_ratio, and Fwd Header Length\_to\_Bwd Header Length\_ratio. These capture differences in forward vs. backward traffic volumes and packet sizes.
* **Rate Features:** We introduced features to measure **rates** of data transfer normalized by time. For instance, for any feature containing "Bytes" in its name, we divided it by flow duration to get bytes per second if not already present. The dataset already has some rates (e.g., Flow Bytes/s), but we ensured that metrics like total bytes, packet counts, etc., were also considered in rate form if useful. One example we added was Total Fwd Bytes\_per\_duration (forward bytes divided by flow duration). High-rate flows could indicate flood behavior typical of DoS/DDoS.
* **Aggregate Traffic Features:** We computed **aggregate sums and means** for all forward-direction traffic features and all backward-direction features separately. This resulted in forward\_total\_activity (sum of all forward packet/byte count features for a flow) and backward\_total\_activity, as well as forward\_mean\_activity and backward\_mean\_activity (averages across those features). These aggregates provide a single feature representing the overall magnitude of data in each direction, which can simplify learning if the model needs a notion of how "big" a flow is in total.
* **Flow Asymmetry:** Building on the above, we defined a **traffic asymmetry index**: traffic\_flow\_asymmetry = (forward\_total\_activity - backward\_total\_activity) / (forward\_total\_activity + backward\_total\_activity). This yields a value between -1 and 1, where 0 means symmetric flow, positive means more forward traffic, and negative means more backward traffic. DDoS traffic from a client to a server would likely show a strong positive asymmetry (lots of forward packets with minimal response), whereas normal bidirectional communications (e.g., a file download) might be more balanced or even backward-heavy (server sending data to client). This feature was intended to capture such differences succinctly.
* **Port-based Features:** Many attacks target specific ranges of ports. We added features to classify port numbers into categories: for each of source/destination port, we created binary flags indicating if the port is "well-known" (<=1023), "registered" (1024–49151), or "dynamic/private" (>49151). This encoding (\*\_is\_wellknown, etc.) can help the model learn if traffic is on unusual ports often associated with malicious activity. For instance, a sudden surge of traffic on a typically unused high-numbered port could be suspicious. We only applied this to one side (e.g., destination port) to avoid redundancy, since usually one side is the server port of interest.

These domain-specific features are grounded in network security knowledge: prior research and expert intuition suggest that DDoS attacks often exhibit extreme ratios (lots of packets with small payloads), high packet rates, asymmetric flows, and sometimes target specific port ranges. By explicitly providing these features, we make it easier for the models to capture the patterns distinguishing DDoS vs. benign without having to rely solely on the original features. After feature engineering, the dataset had **96 features** (79 original + 17 new). Our code logged the creation of these new features and verified their values (e.g., ensuring no division-by-zero occurred by adding small constants where needed). The addition of these features increased the total dataset memory usage (to ~2.3 GB), but this was manageable. We observed from EDA that some engineered features were indeed effective: for example, the traffic\_flow\_asymmetry for known DDoS flows tended to be close to 1 (high forward bias), whereas benign flows had more distributed values.

### Feature Selection and Dimensionality Reduction

Including all 96 features in the final model could lead to overfitting, increased computation, and difficulty in interpretation. We therefore performed feature selection to retain only the most relevant features for classification. Our selection process consisted of several steps:

**1. Remove Redundant Features:** First, we eliminated features with zero variance and highly correlated features. We dropped the 8 features that were constant (zero variance) across all flows, as these provide no information. Next, we applied a correlation threshold: using the correlation matrix computed earlier, we removed one feature from each pair of features with correlation above 0.95 (absolute correlation). This was done to avoid multicollinearity. We ensured that when dropping, we kept features that we deemed more broadly informative (for example, between Total Fwd Packets and Total Length of Fwd Packets which are nearly collinear, we might keep one consistently). This step reduced the feature count from 96 down to 63 features. By doing so, we already achieved a ~34% reduction in dimensionality while preserving >99% of the variance in the data (since correlated features carry overlapping information). It also helped later algorithms like KNN and SVM by mitigating the curse of dimensionality.

**2. F-test Statistical Selection:** We employed a univariate feature selection using the F-test (ANOVA) between each feature and the binary class label. The F-test score measures how different the feature values are between the two classes, relative to intra-class variance – higher scores indicate a feature that better separates attack vs. benign. Before applying F-test, we separated the data into the feature matrix $X$ and target vector $y$ (Binary\_Label). We also standardize features at this stage because extreme scales can influence F-scores. However, since we plan to standardize later for modeling anyway, we simply used the VarianceThreshold and F-test on the raw (but outlier-capped) values, which is acceptable as F-test is scale-invariant to some degree (it’s a ratio of variances). We used SelectKBest(f\_classif, k=10) to choose the top 10 features by F-score. We set $k=10$ as a target largely for practicality and to align with prior work that found a small subset can suffice. The selection was done after ensuring no zero-variance or duplicate features remained, so all features considered had some variance.

The **top 10 features** selected (in descending order of F-score) were as follows: 1. **Flow Bytes/s\_to\_Flow Packets/s\_ratio** – This ratio of bytes per packet had the highest discriminatory power, likely capturing that attack flows often consist of either many small packets or other anomalous payload patterns. 2. **Min Packet Length** – The smallest packet size in the flow. Many attack tools send uniform small packet bursts, so a very low minimum might indicate an attack. 3. **Bwd Packet Length Min** – The smallest packet length in the backward direction. In a DDoS, responses from the victim might be minimal or standardized, making this feature salient. 4. **Total Fwd Packets** – Total number of packets sent from source to destination. DDoS flows often involve a large number of packets. 5. **Fwd Packet Length Mean** – Average packet length forward. Could indicate characteristic packet sizing in malicious traffic. 6. **Fwd IAT Std** – Variation in inter-arrival time of packets forward. DDoS tools might produce more regular packet timing (or conversely extremely bursty traffic) compared to benign flows. 7. **Flow Duration** – Total duration of the flow. Some attacks either produce very long-lasting flows or very short ones; duration can help differentiate certain attack types. 8. **Traffic Flow Asymmetry** (engineered feature) – As expected, this was among the top features, confirming that directional imbalance is a key indicator. 9. **SYN Flag Count** – A TCP-specific feature (how many SYN packets). High SYN counts can hint at SYN flood attacks (a type of DoS). 10. **Subflow Fwd Bytes** – Bytes in subflows forward (a related measure to total bytes).

This selection covers a mix of original and engineered features, demonstrating that our domain features added value (several appear in the top 10). Notably, features related to packet size and counts dominate the list, which aligns with intuition for DDoS: these attacks often involve abnormal packet frequency or size distributions.

After selection, we reduced our feature set to these 10 features, which significantly shrinks the data dimensionality (from 79 original to 10, ~87% reduction). According to our tests, this reduction had minimal impact on accuracy – in fact, the Random Forest model’s accuracy only dropped from about 98.3% with all features to 98.15% with the top 10. This indicates that the selected features retained most of the relevant information for classification. It also makes the model faster and more interpretable.

We note that we did not perform exhaustive hyperparameter tuning for feature selection due to time constraints; more sophisticated methods (e.g., recursive feature elimination or wrapper methods) could potentially yield an even smaller optimal subset. However, our chosen features align well with known indicators of malicious traffic, lending credence to their suitability.

### Model Selection and Training

We evaluated four classification algorithms that represent different learning paradigms: - **Random Forest (RF):** an ensemble of decision trees using bagging and feature randomness, known for high accuracy and robustness on structured data. - **Decision Tree (DT):** a single interpretable decision tree, included for its simplicity and explainability. - **k-Nearest Neighbors (KNN):** a distance-based non-parametric classifier that classifies a sample based on majority vote of its neighbors. Useful to capture local data structure. - **Support Vector Machine (SVM):** a margin-based classifier that finds an optimal boundary (we used an RBF kernel SVM) which can capture non-linear patterns.

These models were chosen to provide a comparison between ensemble vs. single-tree, parametric vs. non-parametric, and distance vs. boundary-based methods. All models were implemented using scikit-learn with default parameters unless otherwise noted. For instance, our Random Forest used 100 trees by default and SVM used an RBF kernel with $C=1$, $\gamma='scale'`. We set a fixed random seed (42) for reproducibility in data splits and model training.

**Data Splitting:** After feature selection and balancing, we split the data into training and test sets. We used an 80/20 stratified split, resulting in 160,000 training samples and 40,000 test samples. Stratification ensured the 50:50 class ratio was preserved in both sets (so 20k attacks and 20k benign in test). This approach tests the models on previously unseen data while maintaining balanced evaluation, which is important for reliably measuring metrics like precision and recall.

**Feature Scaling:** We applied standardization (zero mean, unit variance scaling) to the selected features prior to training KNN and SVM. Tree-based models (RF and DT) do not require scaling, but for consistency and to integrate with a single pipeline, we scaled the features for all models. A StandardScaler was fit on the training data and used to transform both training and test sets. Scaling is critical for KNN (which uses Euclidean distance) and SVM (RBF kernel is sensitive to feature magnitude). Without scaling, features with larger numeric ranges would dominate distance calculations. After scaling, all features effectively lie on comparable scales, and the mean of each feature in training is 0 with standard deviation 1. We saved the scaler object for use in deployment.

**Classifier Training:** - Random Forest: We trained an RF classifier on the training set (160k samples, 10 features). Training took around a couple of minutes on a multi-core CPU. We kept default settings (which include bootstrapping, Gini impurity, etc.) and observed the OOB estimate of error was around 1.9%, indicating a strong fit. RF inherently handles overfitting through averaging, but we were mindful of not using too many trees to avoid long inference times; 100 trees were sufficient as performance plateaued beyond that. - Decision Tree: A single DT was trained for comparison. We did not prune it deeply, letting it grow to fit the data. It achieved high training accuracy (almost 100% on training, which is expected as an unpruned tree can memorize data), but test performance was slightly lower than RF due to overfitting. The tree, however, provided insight into which features and thresholds were used for splits, aligning with our selected top features (e.g., it often split first on the Flow Bytes/s\_to\_Packets/s\_ratio). - KNN: We trained a KNN classifier with $k=5$ (default). No actual "training" is needed other than storing the scaled training instances. The main cost is in query time. KNN can capture non-linear boundaries by essentially remembering examples, but can be less effective if irrelevant features or high dimensions are present. With only 10 features, the curse of dimensionality is mitigated. We expected KNN to do fairly well since attacks form clusters in feature space (as seen in EDA some separation). - SVM: We trained an SVM with RBF kernel. The training time was longer (several minutes) due to the 160k sample size and quadratic complexity of SVM. We didn’t perform hyperparameter tuning (like finding optimal $C$ or $\gamma$) due to time constraints, which might impact its performance. SVMs can be powerful for complex boundaries but may struggle if classes are not linearly separable without careful kernel parameter tuning or if there's noise.

Each model was evaluated on the test set after training. We computed standard metrics: **Accuracy**, **Precision**, **Recall**, and **F1-score** for the attack class (we treat "attack" as the positive class). We also generated confusion matrices for each model to examine the false positive (benign misclassified) and false negative (attacks missed) counts.

## Experimental Results

In this section, we present the performance results of the trained classifiers on the test set. We report key metrics and provide comparative analysis, along with a confusion matrix for the best model. All results correspond to the binary classification of attacks vs. benign traffic on a balanced test set of 40,000 flows (20k attacks, 20k benign).

**Overall Performance:** Table II summarizes the accuracy, precision, recall, and F1-score for each model. The Random Forest achieved the highest F1-score, indicating the best balance between precision and recall, while the SVM lagged behind significantly in precision.

Table II: **Classification Performance of Different Models**

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| --- | --- | --- | --- | --- |
| Random Forest | **98.15%** | 97.96% | 98.33% | **98.14%** |
| Decision Tree | 97.99% | 97.81% | 98.15% | 97.98% |
| KNN (k=5) | 97.62% | 97.02% | 98.23% | 97.62% |
| SVM (RBF kernel) | 89.42% | 84.37% | 96.64% | 90.09% |

As shown, the Random Forest classifier marginally outperforms the Decision Tree and KNN classifiers, which themselves perform quite well with F1 around 97-98%. The **Random Forest** attained an F1 of 98.14%, with accuracy ~98.15%, precision ~97.96%, and recall ~98.33%. In practical terms, this means it correctly identified over 98% of attack flows while keeping false alarms (false positives) very low. The Decision Tree’s performance is only slightly lower (F1 97.98%), indicating that a single tree can almost match the ensemble on this dataset – likely because the selected features make the decision boundary relatively straightforward. **KNN** also yielded a high recall (98.23%) but slightly lower precision (97.02%), suggesting it let in a few more false positives. Still, its overall performance (97.62% F1) is commendable, showing that many attack instances cluster well in feature space.

The notable outlier is the **SVM**, with only 90.09% F1 and particularly low precision (84.37%). It correctly detected most attacks (96.64% recall, nearly as high as others), but misclassified a substantial number of benign flows as attacks (over 15% false positive rate, given precision 84%). We suspect this is due to the difficulty in tuning the SVM’s decision boundary with default parameters – the model likely chose a boundary that errs on the side of labeling uncertain points as attacks, hence the high recall but poor precision. It may also reflect that the classes are not linearly separable in the scaled feature space, and the RBF kernel needed parameter optimization. In contrast, tree-based models inherently adjusted to the data distribution effectively.

**Confusion Matrix:** To illustrate the prediction breakdown, **Figure 1** shows the confusion matrix for the Random Forest model on the test set.

Figure 1. Confusion matrix of the Random Forest classifier on the 40k test set (20,000 attacks and 20,000 benign flows). The model achieves high true positive and true negative counts, with very few false negatives (missed attacks) and false positives (benign misclassified as attacks).

The confusion matrix in Figure 1 confirms the excellent performance of Random Forest. Out of 20,000 actual attack flows, the model correctly detected 19,571 as attacks (**True Positives**), missing only 429 attacks (**False Negatives**). This corresponds to a detection rate (recall) of 98.33%. Out of 20,000 actual benign flows, 19,689 were correctly identified as benign (**True Negatives**), and only 311 were mistakenly flagged as attacks (**False Positives**), giving a false alarm rate of about 1.56%. These error counts are very low relative to the dataset size. For a security analyst, this means only 1.56% of normal traffic would raise an unnecessary alarm, and only 2.1% of attacks would slip by undetected – a favorable trade-off. The high precision (97.96%) indicates that the vast majority of flows the model flags as attacks are indeed attacks, which is critical in operational settings to avoid alert fatigue.

**Comparison and Analysis:** The results validate that ensemble learning (Random Forest) provides a slight edge in accuracy and stability. The Random Forest’s improvement over a single Decision Tree, while not huge, suggests that ensembling helped to generalize better and reduce overfitting (the tree likely slightly overfit some noise, which the forest corrected through averaging). KNN’s strong performance is interesting – it implies that in the 10-dimensional feature space, attack flows cluster distinctly enough that even a simple nearest-neighbor approach can separate them nearly as well as trees. However, KNN would be computationally expensive for very large deployments and lacks an easily interpretable model.

The poor showing of SVM indicates that more careful tuning or feature transforms would be needed to make SVM competitive here. It might be that the default regularization (C=1) was too low, allowing too many support vectors and effectively overfitting the minority class. Or the RBF kernel width might not have been optimal. Additionally, SVM took significantly longer to train (and would be slow to evaluate on large sets), which diminishes its practicality for this problem as features are already quite informative linearly for tree models.

We also examined the **feature importance** from the Random Forest model (using Gini importance). The top features in order of importance aligned with our F-test selection: Flow Bytes/s\_to\_Packets/s\_ratio was the most important, followed by Min Packet Length and Traffic Flow Asymmetry. This corroborates that our feature selection indeed picked up the most predictive attributes. In fact, the top three features contributed over 50% of the total importance in the Random Forest, suggesting a simple model using just those three could already do quite well (though we kept 10 for safety and broader coverage).

**Metrics Interpretation:** For a security application, **Recall** (attack detection rate) is paramount – missing attacks (false negatives) could mean a breach goes unnoticed. Our best model’s recall ~98.3% is encouragingly high, meaning it catches the vast majority of malicious flows. **Precision** is also important operationally, as a low precision (many false alerts) can overwhelm analysts and lead to ignoring alarms. The Random Forest’s precision ~98% means false alerts are very minimal (roughly 2% of alerts are false). This high precision is partly due to the balanced evaluation; in a real network with 4:1 benign:attack ratio, the precision might vary if the model were deployed without balancing. However, since we balanced training, the model is not biased to label everything benign, thus maintaining good precision. **Accuracy** in our balanced test is also ~98%, but in an imbalanced real environment, accuracy would be dominated by benign class. We emphasize that F1-score is a more informative single metric here as it considers both precision and recall.

In summary, the Random Forest model emerges as the best choice for this DDoS detection task, with nearly 98-99% scores across the board. The Decision Tree and KNN are close followers, indicating that simpler models can suffice with the right features. SVM underperformed in this configuration. These results are consistent with prior studies where Random Forest often outperforms or matches more complex models for intrusion detection.

## Discussion

The experimental results demonstrate the effectiveness of our machine learning pipeline for detecting DDoS and other attacks in network traffic. Here we discuss the implications of these results, the role of each pipeline component, and the strengths and limitations of our approach.

**Impact of Preprocessing:** The high performance of all models (except SVM) suggests that our **EDA-driven preprocessing** was successful in making the underlying pattern more learnable. By capping outliers, we prevented extreme values from distorting the model training. We believe this particularly helped linear models like SVM (though it still needed tuning) and distance-based KNN, which are sensitive to scale. The outlier capping likely also contributed to Random Forest’s stability – without capping, some trees might overly focus on splitting outlier values. **Class balancing** was crucial: had we trained on the original 80/20 imbalanced data, the classifiers (especially Decision Tree and KNN) might have biased toward benign class predictions to achieve higher accuracy, at the expense of recall. The balanced training forced them to learn what distinguishes attacks, yielding high recall. Our results showed minimal false negatives, confirming that balancing improved detection of the minority class. The trade-off is that we ignored some benign data (via undersampling), but given millions of benign flows, using 100k was sufficient to capture their distribution diversity (evidenced by low false positive rate). In practice, alternative strategies like cost-sensitive learning or oversampling could be explored to utilize more data without bias; however, those come with complexity and risk of overfitting duplicates.

**Feature Engineering Value:** One key outcome is that the **domain-specific features added significant predictive power**. The top-ranked features included several that were not in the original set (ratios, asymmetry) yet turned out to be most important for classification. This validates our hypothesis that combining existing metrics into more meaningful indicators can simplify the model’s task. For example, instead of expecting the model to learn the relationship between Fwd Packet/s and Bwd Packet/s separately, giving it a pre-computed ratio directly highlights that relationship. This likely contributed to the model’s high precision: it could better distinguish normal two-way flows from one-sided attack flows. The port category features did not end up in the top 10, possibly because many attacks in CIC-IDS2017 use common ports (e.g., HTTP attacks on port 80) so port-based detection was less useful. But those features could be more relevant if obscure ports were used by attackers. Feature engineering was relatively inexpensive in terms of computation and added explainability – security analysts can intuitively understand features like "traffic asymmetry" or "packet size ratio," which makes trust in the model’s decisions easier.

**Feature Selection and Dimensionality:** Reducing features to 10 made the pipeline more **efficient** and **interpretable**. Training time for Random Forest and KNN decreased with fewer features, and the resulting model is easier to reason about. It also helped the Decision Tree to remain shallow enough (the tree depth was around 12) to interpret. We were able to trace the decision path: e.g., if Min Packet Length <= 1 byte and Flow Packets per sec ratio is high, then it’s classified as attack – a rule that makes sense as many DDoS attacks send a flood of tiny packets. This kind of rule-based insight can be valuable for cybersecurity experts to validate model behavior. The feature selection did not noticeably degrade detection capability, which indicates that some of the original features were indeed redundant or noisy. For instance, having both packet count and byte count was redundant; our selection kept the more informative of each pair. However, we must acknowledge that feature selection was done on the entire dataset (not strictly within cross-validation folds), which can risk overfitting. We mitigated this by using simple filters (F-test) rather than wrapper selection on the model, and the final test performance suggests it generalized well. In a more rigorous setup, one would integrate feature selection into the cross-validation process.

**Model Performance and Trade-offs:** The Random Forest’s superior performance is not surprising – it combines the votes of many trees, each capturing different aspects of the data, thus handling variability well. It also naturally provides some outlier resistance (via majority voting). The near-perfect precision and recall imply it could be deployed in a real network with minimal tweaking. The Decision Tree’s comparable performance is interesting; it means that the decision boundary between attack and benign, in the space of these features, can be approximated by a few logical rules. This is good news for interpretability but one must be cautious – the tree achieved that performance on a balanced dataset; on a skewed live network stream, it might need threshold adjustments (e.g., if treating probability outputs, one could shift the threshold to maintain a low false positive rate in production). KNN’s performance, while high, comes with memory and latency costs. For 160k training points, a KNN query requires distance computations to all those points (unless optimized with indexing structures), which could be too slow for real-time use on high-throughput networks. It also doesn’t give an easily explainable model. Therefore, we lean towards Random Forest or Decision Tree for deployment. SVM’s struggle highlights that even powerful algorithms can falter without proper tuning; it essentially overfit the attack class in our test, leading to too many false alarms. With tuning, SVM could improve, but given the results, there is little incentive when simpler models are already at 98% F1.

**Error Analysis:** The few errors the Random Forest made are worth examining. False negatives (missed attacks) were 429 out of 20k attacks (~2.1%). These might correspond to attack flows that closely resemble benign traffic in terms of selected features. For example, some DDoS attacks in the dataset might be low-rate or stealthy, not triggering the typical signatures (small packets, high asymmetry) – those could slip through. False positives (311 out of 20k benign, ~1.6%) indicate some benign flows were flagged as attacks. These could be, for instance, large file downloads or video streams that have high forward packet rates and asymmetric patterns similar to DDoS (lots of data in one direction). Upon manual inspection, one might find that certain benign applications mimic attack-like behavior (e.g., data backup jobs might open connections that send continuous streams of data). If needed, those specific patterns could be whitelisted or an additional feature could be introduced to distinguish them (such as application-layer metadata, if available). However, a 1.6% false positive rate is likely tolerable in many security operations – it means analysts would look at ~311 flows out of 20k benign, which is a manageable volume.

**Comparison to Prior Work:** Our results (98.14% F1 with RF) are in line with, and slightly exceed, many prior machine learning approaches on CIC-IDS2017. This confirms that a well-curated feature set and ensemble classifier can achieve state-of-the-art performance on intrusion detection benchmarks. Some studies report near-100% accuracy with more complex ensembles or deep learning, but often those come with trade-offs like lack of transparency or heavy computation. We aimed for a balance of performance and practicality. The interpretability of our pipeline (with only 10 features and a clear preprocessing log) is a strength in a field where explaining why an alert was raised is important for trust. Additionally, our pipeline is **extensible**: new data can be fed in and processed through the same steps (outlier capping, feature generation, scaling) to get predictions, and it’s modular to update any component (for instance, one could swap in a Gradient Boosting model for potentially marginal gains).

**Limitations:** Despite the high accuracy, there are limitations to note: - The evaluation was done on the same distribution as training (from CIC-IDS2017). In real-world networks, traffic patterns and attack types evolve. Our model might need retraining or updating for new types of DDoS not seen in 2017. It may also need adjustment for different baseline traffic (CIC-IDS2017’s benign traffic might not cover all real behaviors). - We treated this as a binary classification. In practice, distinguishing attack types can be valuable (e.g., separating DDoS from port scan), to tailor incident response. Our approach could be extended to multi-class, but performance for minor classes might be lower. Some references report that multi-class detection on CIC-IDS2017 yields lower accuracy on certain rare classes. Our focus on binary means the model is very good at saying "attack or not", but not what kind of attack. - The pipeline currently works on flow-level aggregated data. If one needed real-time packet-level detection or working on streaming packet data, additional considerations (like time-windowed features, incremental updates) would be needed. However, since many modern intrusion detection systems operate on flows (e.g., using NetFlow data), our approach is still applicable. - Lastly, while our model minimizes false positives, any non-zero false alarm rate could be an issue in high-volume environments. If deploying in a network with millions of flows per day, 1-2% false positives could still be thousands of alerts. Further tuning of the decision threshold or adding an anomaly score could help prioritize alerts.

## Conclusion and Future Work

In this paper, we presented a comprehensive machine learning pipeline for DDoS detection, which integrates detailed data preprocessing, domain-specific feature engineering, and ensemble classification. Using the CIC-IDS2017 benchmark dataset, we demonstrated that our approach achieves high accuracy in distinguishing attack traffic from benign traffic. The Random Forest model emerged as the best performer, attaining a 98.14% F1-score with both recall and precision around 98%. This translates to a DDoS detection rate above 98% and a very low false alarm rate (~2%), which are desirable characteristics for a practical intrusion detection system.

Our pipeline’s success can be attributed to several factors: 1. **EDA-driven insights** informed targeted preprocessing (outlier capping and class rebalancing) that enhanced model learning. 2. **Feature engineering** injected domain knowledge (ratios, asymmetry, etc.) that made malicious patterns more separable in feature space. 3. **Feature selection** reduced noise and redundancy, resulting in a compact feature set that retained critical information and improved computational efficiency. 4. **Robust modeling** with an ensemble method ensured strong generalization and resilience to outliers and overfitting.

The outcome is a detection system that is not only accurate but also relatively interpretable and deployment-ready. We saved the trained Random Forest model and the preprocessing scaler, so the system can be deployed to classify new network flows in real time. The model’s simplicity (10 features) means it can execute quickly even on modest hardware, and the whole pipeline from data ingestion to prediction can operate within minutes for large batch datasets, or in near real-time for streaming data (with appropriate optimizations and stream processing frameworks).

**Future Work:** Building on this work, several avenues can be pursued to further improve and adapt the system: - **Deep Learning Integration:** Investigate advanced models like deep neural networks or recurrent networks for attack detection. For example, a CNN or LSTM could automatically learn features from raw packet sequences. Prior work has shown deep models can reach very high accuracy, though at the cost of complexity. A hybrid approach could combine our engineered features with deep learning (perhaps feeding the features into a neural network, or using deep learning to extract additional features). - **Online and Streaming Detection:** Modify the pipeline for streaming data to detect DDoS in real-time. This involves using streaming feature extraction (calculating flow features on the fly) and possibly updating the model incrementally. One could employ techniques from concept drift adaptation so the model remains effective as traffic patterns evolve. - **Ensemble of Models:** While Random Forest alone did well, an ensemble that includes heterogeneous models could be explored. For instance, a meta-classifier could combine the outputs of RF, KNN, and an LSTM, to potentially catch different aspects of attacks. Caution is needed to justify the added complexity, but ensemble-of-ensembles might squeeze out the last bit of performance. - **Feature Enhancement:** Future work may include **wrapper-based feature selection** or **genetic algorithms** to search for an even better feature subset than the F-test provided. Also, incorporating new domain features like those based on entropy of packet contents, or frequency of connections from the same source, could strengthen detection especially for distributed attacks. - **Threshold Tuning and Anomaly Scoring:** Instead of a hard classification, the system could output an anomaly score or probability, which is then thresholded. By analyzing ROC curves, one can set thresholds that meet specific operational requirements (e.g., maximize recall at a given false positive rate). Dynamic thresholding based on traffic load might also be considered, to be more sensitive during suspected high-risk periods. - **Multi-class Classification:** Extending the system to classify specific attack types (DDoS vs. port scan vs. infiltration, etc.). This is useful for incident response – knowing the kind of attack can guide mitigation steps. Our current binary model could be expanded or multiple one-vs-rest models could be trained. However, as discussed, the challenge is ensuring the model has enough data to learn each class well, and dealing with class imbalance among attack types. - **Deployment and Feedback Loop:** We aim to integrate the model into a real network monitoring stack (e.g., using a REST API and a dashboard). Once deployed, a feedback mechanism could be implemented where alerts verified by analysts (true or false) are fed back into model training, allowing it to improve over time. Automated retraining on new data periodically would keep the model up-to-date with emerging threats. - **Edge and IoT Scenarios:** For protecting IoT networks or edge devices with limited resources, we could explore lightweight versions of the model. Since we have only 10 features and a relatively small model, this could potentially run on edge gateways. There is also scope to test the pipeline on other datasets (like CSE-CIC-IDS2018, UNSW-NB15) to verify its generality.

In conclusion, the presented DDoS detection system achieves a high detection rate with low false alarms by marrying comprehensive data preprocessing with powerful yet interpretable machine learning models. It showcases how classic machine learning, when applied with domain insight and thorough analysis, can effectively secure networks against DDoS attacks. This work provides a solid foundation for an operational intrusion detection tool, and with further enhancements such as deep learning and real-time streaming, it can be extended to meet the evolving challenges of cybersecurity.

## References

1. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in Proc. 4th Int. Conf. Information Systems Security and Privacy (ICISSP), 2018.
2. M. Belouch, S. El Hadaj, and M. Idhammad, “Performance evaluation of intrusion detection based on machine learning using Apache Spark,” Procedia Computer Science, vol. 127, pp. 1–6, 2018.
3. K. A. Taher, B. M. Y. Jisan, and M. M. Rahman, “Network intrusion detection using supervised machine learning technique with feature selection,” in Proc. International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), 2019, pp. 643–646.
4. M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Multi-stage optimized machine learning framework for network intrusion detection,” IEEE Trans. Network and Service Management, vol. 18, no. 2, pp. 1803–1816, 2021.
5. V. Dora and V. N. Lakshmi, “Optimal feature selection with CNN-feature learning for DDoS attack detection using meta-heuristic-based LSTM,” Int. J. Intelligent Robotics and Applications, vol. 6, no. 2, pp. 1–27, 2022.