Flow-Based Intrusion Detection Using Ensemble Machine Learning

Devesh Senthilraja

Abstract

This paper presents a flow-based intrusion detection approach designed for modern encrypted network traffic. The proposed method uses an ensemble of machine learning classifiers in a two-tier stacking architecture to detect malicious flows using only flow-level metadata, without inspecting any packet payloads. The approach is evaluated on a large-scale benchmark dataset (CIC-IDS2018) containing diverse attack types mixed with extensive encrypted traffic. Results indicate that the stacking ensemble outperforms individual classifiers and traditional voting ensembles in detecting a wide range of attacks, while preserving privacy by avoiding decryption. The study establishes baseline performance for various algorithms on encrypted flows, demonstrates the advantages of learned ensemble fusion, and provides insights through ablation and feature augmentation experiments. These contributions illustrate a practical solution for intrusion detection in fully encrypted network environments, combining high detection effectiveness with privacy preservation.

I. Introduction

The widespread adoption of encryption in Internet communications has created significant challenges for traditional intrusion detection systems. Over 80% of Internet traffic is now encrypted, which means that legacy defenses relying on packet payload inspection are increasingly operating blind. Techniques like deep packet inspection and signature-based pattern matching (e.g., Snort rules) become ineffective when packet contents are hidden. Attackers have capitalized on this blind spot by tunneling malicious activities through encrypted channels (for instance, disguising command-and-control traffic or data exfiltration as normal TLS flows), rendering many conventional detection methods obsolete. This trend underscores the need for payload-agnostic detection strategies that can identify threats without reading packet contents. Few existing IDS solutions handle fully encrypted traffic effectively, which motivates the present work to fill that gap using flow-level information only.

Another core challenge is that encrypted malicious traffic often blends in with legitimate traffic at the metadata level. Without payload signatures, an IDS must distinguish benign and malicious flows using only statistical features such as packet lengths, rates, timing intervals, and protocol headers. Sophisticated attacks can mimic normal user behavior

to evade detection, leading to high false-positive rates if detection models are not sufficiently robust and adaptive. Moreover, modern attack campaigns are diverse (ranging from high-volume denial-of-service floods to low-and-slow infiltrations), requiring detection methods capable of handling a wide spectrum of patterns. Identifying all these attack types from flow-level observations alone demands machine learning techniques capable of modeling subtle statistical differences without relying on decrypted content. In summary, conventional IDS approaches struggle in encrypted environments and few alternatives exist; this gap underscores the need for an intrusion detection approach based on flow metadata.

To address these challenges, this paper proposes a privacy-preserving intrusion detection method grounded in ensemble machine learning, with a focus on a stacking-based meta-classifier architecture. In contrast to single-model classifiers or simplistic majority-vote ensembles, a stacking ensemble combines the strengths of multiple learning algorithms by training a meta-classifier to fuse their outputs. This design enables the detection system to capture complex non-linear relationships in flow feature space and to compensate for individual model weaknesses. The approach relies exclusively on statistical features extracted from each network flow (e.g., packet counts, byte rates, inter-arrival times, and protocol flags) rather than any packet payload content, making it naturally suited for monitoring fully encrypted traffic. The resulting two-tier model adapts to diverse attack patterns and provides a practical, privacy-compliant anomaly detection solution for modern networks.

Research Objectives and Contributions: This study develops a flow-based stacking ensemble IDS that achieves high attack detection capability using only network flow metadata. The key contributions of this work are summarized as follows:

- Payload-Agnostic Stacking Ensemble: Introduces a two-tier stacking ensemble IDS using heterogeneous base learners on flow-level features, attaining high detection efficacy without inspecting packet payloads. This demonstrates an effective solution for encrypted traffic where deep packet inspection fails.
- Baseline Algorithm Performance: Establishes robust baseline results by evaluating ten standalone machine learning classifiers across multiple feature subsets, benchmarking the limits of single-model performance on encrypted traffic.
- Learned vs. Fixed Fusion: Demonstrates the advantage of a learned fusion approach by comparing the stacking meta-model to traditional ensemble voting methods (hard/soft voting). The stacking ensemble with an optimized

meta-classifier outperforms equal-weight voting, highlighting the value of trainable combination rules.

- Base Learner Importance Analysis: Quantifies the contribution of each base learner through an ablation study. Removing individual base models reveals their influence on ensemble performance, providing insight into model diversity and complementarity.
- Feature Augmentation for Minority Attacks: Explores augmenting the meta-classifier's input with select raw flow features. This extension evaluates whether simple additional features can further boost detection performance.

II. Literature Review

2.1 Traditional Anomaly Detection Techniques and Their Limitations

Early network intrusion detection systems largely fell into two categories: signature-based detectors and anomaly-based detectors. Signature-based IDS (exemplified by tools like Snort) use known patterns of malicious payloads to identify attacks, whereas anomaly-based systems establish a baseline of normal behavior and flag deviations from that baseline. Both approaches face serious limitations in today's environment. Signature-based techniques require visibility into packet contents and thus fail outright on encrypted traffic. Anomaly-based methods, on the other hand, may detect novel attacks but often at the expense of high false positives, since any unusual but benign behavior can be misclassified as malicious. This fundamental trade-off has highlighted the need for detection approaches that are both sensitive (to catch new attacks) and specific (to avoid false alarms).

Research has underscored the difficulty of achieving this balance. Guo et al. (2023) [5] observe that entirely new zero-day attack patterns often evade purely supervised learning models, which are inherently limited by the scope of their training data. In response, some works have explored unsupervised or one-class anomaly detection techniques, such as clustering network flows or training one-class models on normal traffic, to detect deviations without prior attack labels. While these unsupervised methods can in theory identify novel threats, they tend to suffer from unstable behavior and high false alarm rates if not carefully tuned. Likewise, attempts to use deep learning for anomaly detection (e.g., autoencoders or recurrent neural networks) have encountered challenges: these models can automatically learn features from raw flows, but they demand very large training sets and significant computation, and their decisions are often opaque. In summary, traditional IDS techniques (whether

signature-based or basic anomaly detectors) are ill-equipped to handle the dual challenge of fully encrypted traffic and continuously evolving attacks. These limitations drive the need for new approaches that operate on metadata and leverage more powerful machine learning techniques to achieve both high detection rates and low false positives in encrypted environments.

2.2 Machine Learning for Encrypted Traffic Analysis

With the majority of network traffic now encrypted, intrusion detection research has shifted toward analyzing features that remain observable despite encryption. Kang et al. (2017) [1] emphasize that the sheer volume of encrypted data in enterprise networks makes it difficult to spot sophisticated attacks, as traditional content-inspection methods can no longer “see” inside the traffic. In response, recent work focuses on statistical patterns and side-channel information gleaned from flow metadata rather than packet payloads. Flow-level features, such as packet sizes and counts, inter-packet timing intervals, connection durations, and header flags, have become a cornerstone of anomaly detection for encrypted traffic. Shiravi et al. (2012) [2] introduced comprehensive flow-based feature sets as content-agnostic proxies for suspicious behavior, demonstrating that many attack types can be identified through metadata alone. Tools like CICFlowMeter (from the Canadian Institute for Cybersecurity) have further standardized the extraction of dozens of such features from packet captures, enabling researchers to reproducibly evaluate IDS models on encrypted traffic across different datasets.

Multiple studies have validated that machine learning models trained on flow features can achieve high accuracy in detecting attacks without any packet payload data. For instance, Ibraheem et al. (2022) [3] showed that classifiers leveraging inter-packet timing and size patterns can successfully surface anomalies in HTTPS (TLS-encrypted) flows. Similarly, Singh et al. (2025) [4] applied explainable machine learning techniques (using SHAP values) to TLS-encrypted traffic, confirming that even though encryption hides content, there are still statistical fingerprints of malicious behavior in the metadata that models can exploit. These works illustrate an important point: encryption blinds straightforward payload inspection, but it does not render traffic analysis impossible; the communication patterns themselves often betray an attack.

Despite this progress, reliably detecting intrusions in encrypted traffic remains challenging. Attackers continuously adapt their tactics, and advanced malware can generate traffic patterns that closely mimic benign usage. For example, an infected host

might deliberately behave like a normal web browser to avoid suspicion. As a result, purely flow-based detectors risk either high false positives (flagging unusual but benign behaviors) or missed detections (if the malicious pattern is too subtle) when they are not carefully designed. Achieving both high sensitivity and high specificity in encrypted traffic analysis is difficult. As noted earlier, supervised learning models can struggle with zero-day attacks that fall outside the training distribution. One strategy to improve generalization has been to incorporate unsupervised anomaly detection components: for example, clustering algorithms or autoencoders that detect outlier flows without needing known attack labels. However, such approaches have their own drawbacks: they can be unstable or trigger excessive false alarms if they pick up on noise. Another strategy has been to use deeper learning architectures (e.g., recurrent neural networks or deep autoencoders) to automatically learn feature representations from flows, which has shown promise but comes at the cost of high computational complexity. Overall, the literature indicates that detecting intrusions in fully encrypted environments is feasible using flow metadata alone, but doing so demands robust and adaptive models capable of handling a diverse range of attack patterns and adjusting to new threats. This insight has driven research toward ensemble and other advanced machine learning techniques that can boost detection performance while maintaining the generality needed for encrypted traffic scenarios.

2.3 Intrusion Detection Datasets and CIC-IDS2018

Evaluating intrusion detection methods requires representative datasets that capture realistic benign traffic and a variety of attack behaviors. Historically, researchers relied on benchmarks like the KDD Cup 1999 dataset and its improved version NSL-KDD, which provided labeled examples of simulated attacks. While useful in their time, these older datasets are now considered outdated; they lack the diversity of modern malware and do not include encrypted traffic. More recent corpora such as UNSW-NB15 (2015) and the Canadian Institute for Cybersecurity IDS datasets (2017 and 2018 editions) were developed to address these gaps by incorporating contemporary attack techniques, updated network protocols, and, in the case of CIC-IDS2018, a significant proportion of encrypted traffic.

CIC-IDS2018 (also known as CSE-CIC-IDS2018) is a prime example of a modern intrusion detection dataset and is used as the basis for this study. Introduced by Sharafaldin et al. (2018) [7], CIC-IDS2018 was collected in a controlled cyber-range environment that closely mimics a large enterprise network. It encompasses 10 days of traffic captures, each day simulating specific attack scenarios blended with normal

background traffic. In total, the dataset contains over a dozen attack types covering a broad spectrum of threat categories including brute-force authentication attacks, various denial-of-service (DoS and distributed DoS) floods, botnet command-and-control traffic, web application exploits (such as SQL injection and XSS), internal network infiltration, and data exfiltration. All these attacks are interspersed with legitimate user activity (e.g., web browsing, email, chat, voice-over-IP), yielding a complex mix that reflects real-world network conditions. Crucially, a large portion of the benign traffic in CIC-IDS2018 is encrypted (for example, HTTPS web browsing, SSH sessions, and VPN connections), which makes this dataset especially well-suited for evaluating flow-based IDS approaches that cannot rely on packet payload inspection.

Each network connection (flow) in CIC-IDS2018 is described by an extensive set of features generated by CICFlowMeter, amounting to approximately 80 attributes per flow. These features capture a wide range of behavior indicators: for example, basic counters (total packets and bytes sent/received in each direction), flow duration and byte rates, packet length statistics (minimum, maximum, mean, standard deviation), inter-arrival time metrics, counts of TCP flag occurrences, connection status indicators, and more. All features are numeric or boolean and represent purely metadata about the flow; no application-layer payload content is included, preserving privacy. Each flow record is labeled as Benign or as a specific attack type, allowing for both binary classification and more fine-grained analysis by attack category (though models in many studies, including this one, focus on the binary distinction between benign and malicious). The rich feature set in CIC-IDS2018 provides a strong foundation for machine learning algorithms; prior work has shown that carefully chosen subsets of these features can yield detection accuracies above 95% for many attack classes on this dataset. For instance, features related to traffic volume and timing are highly effective for catching high-rate attacks like DDoS, whereas more subtle features such as certain unusual TCP flag patterns or idle time distributions can help identify stealthy intrusions.

However, the high dimensionality of CIC-IDS2018's feature space also poses challenges. Many of the ~80 recorded features are inter-correlated or even redundant (for example, "Total Fwd Packets" and "Total Fwd Bytes" are closely related, as are various packet length statistics). Including all of these features in a model can confuse learners and increase computational cost without improving accuracy. To tackle this, researchers have applied feature selection and dimensionality reduction techniques on CIC-IDS2018, such as correlation analysis, information gain ranking, and Principal Component Analysis (PCA), to identify the most informative features and eliminate noise. In this study, rather than permanently filtering out features, the features were

organized into semantically related groups and ensemble methods were used to manage complexity. Nonetheless, the general lesson from the literature is that some form of feature reduction or careful feature handling is beneficial when using this dataset. Overall, CIC-IDS2018 represents a comprehensive, up-to-date benchmark for evaluating intrusion detection systems. It provides a rigorous test bed for flow-based IDS approaches, especially those targeting encrypted traffic, because it contains realistic encrypted benign flows alongside a wide array of attack types. By using this dataset, the evaluation in this research covers the challenges of imbalanced classes, diverse attack behaviors, and encryption: key factors that any practical IDS must be able to handle.

2.4 Classification Algorithms for Flow-Based IDS

A wide variety of classification algorithms have been explored for flow-based intrusion detection, ranging from simple linear models to complex ensemble and deep learning methods. Each approach brings its own advantages and inductive biases when modeling network traffic. Key categories of algorithms include:

- Tree-Based Models: Decision trees and tree-based ensembles (e.g., Random Forests, ExtraTrees, and gradient-boosted trees like XGBoost or LightGBM) are well-suited to tabular flow data. They can capture non-linear interactions between features and effectively learn threshold rules (for example, “if packet rate > X and SYN flag count = Y, then classify as attack”). A single decision tree can overfit if grown too deep, but ensemble methods like Random Forest mitigate this by averaging many trees trained on bootstrap samples, improving generalization. Boosting algorithms (like AdaBoost or XGBoost) sequentially add trees that focus on previous errors, often achieving state-of-the-art accuracy on structured data. Overall, tree-based classifiers have been top performers in many IDS studies due to their ability to automatically handle heterogeneous features and discover important split conditions.
- Linear Models: Linear classifiers such as logistic regression and linear support vector machines model a flow’s label as a weighted sum of its feature values. These models are computationally efficient and tend to generalize well when the classes are roughly linearly separable in the feature space. In the context of network flows, linear models can pick up broad distinctions (for instance, a high overall packet count or byte rate might strongly indicate an attack). However, they lack the capacity to capture complex non-linear feature interactions (e.g., a subtle combination of timing and size patterns), so they may underperform on problems where attacks manifest through intricate multi-feature signatures.

Regularization techniques (like L2 penalties) are often applied to linear models in IDS tasks to prevent overfitting, given the high-dimensional input.

- Instance-Based Models: The k-Nearest Neighbors (kNN) classifier is a non-parametric approach that classifies a new flow by examining the labels of the most similar flows (neighbors) in the training set. Instance-based methods can detect attacks that form tight clusters in feature space; for example, a repeated attack pattern might produce very similar flow records that kNN can group together. The challenge with kNN for intrusion detection is that distance metrics become less meaningful in high-dimensional feature spaces, and the method can be sensitive to noisy or irrelevant features. It is also computationally expensive at prediction time, since classifying a new flow requires computing distances to many training examples. In practice, kNN (with a small k) has been tested for catching localized anomalies, but its performance can degrade on large or noisy datasets.
- Neural Network Models: Neural networks, such as multi-layer perceptrons (MLPs), can learn complex decision boundaries by composing multiple non-linear transformations of the input features. In intrusion detection, a moderately sized feed-forward neural network can combine flow features in ways that might detect subtle patterns; for instance, an MLP could learn a hidden neuron that activates for a specific combination of packet timing and size statistics indicative of a particular attack. Prior work has shown that neural nets can achieve high accuracy on IDS benchmarks, but they require careful regularization (e.g., dropout, early stopping) to avoid overfitting, especially when the feature set is large relative to the number of training examples. Simpler neural architectures (one or two hidden layers) have been used as base learners in recent studies to balance complexity and generalization.

By evaluating diverse model families, researchers have found that no single algorithm is universally best for flow-based intrusion detection; each has certain strengths depending on the attack type or feature characteristics. That said, there is a general trend that more complex non-linear models (e.g., boosted tree ensembles and neural networks) tend to outperform simpler ones (linear models or single decision trees) on rich flow datasets. Tree ensembles in particular (such as Random Forests and gradient boosting) have repeatedly been top performers on benchmarks like CIC-IDS2018, likely because they handle the mixture of categorical-like patterns (flags) and continuous patterns (timing or size metrics) very well. Simpler models, however, are not without value; they often provide faster computation and can serve as diverse components in an ensemble. In fact, the diversity among classifiers (linear vs. non-linear, parametric vs.

instance-based, etc.) is beneficial for ensemble methods, as it ensures that different models contribute complementary views of the data. This forms the rationale for using a heterogeneous ensemble of classifiers in the proposed approach, as discussed next.

2.5 Ensemble Learning: Voting vs. Stacking Approaches

Ensemble learning has become a prominent strategy in IDS research due to its ability to improve robustness and accuracy by combining multiple models. Classic bagging ensembles like Random Forest (which aggregates many decision trees) and boosting methods like AdaBoost or XGBoost have long been used to reduce variance and capture diverse attack patterns in intrusion detection. More recently, simpler ensemble schemes based on classifier voting have shown success on benchmark datasets. For example, Demir et al. (2023) [6] introduced a voting-based ensemble system (VEL-IDS) that achieved superior detection rates compared to individual classifiers on CIC-IDS2018. In a voting ensemble, several base classifiers are trained independently and their outputs are combined by a fixed rule: hard voting predicts an attack if the majority of base models vote “malicious,” whereas soft voting averages the base models’ predicted probabilities and chooses the class with the highest average. Voting is straightforward to implement and often boosts accuracy by smoothing out the individual errors of classifiers. Indeed, prior IDS studies have found that even a simple majority vote can outperform the best single model in some cases.

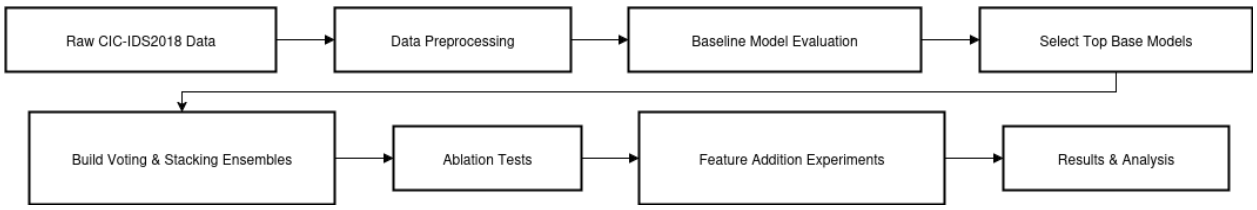
However, a limitation of equal-weight voting is that it treats all base learners as equally important. In practice, if one classifier is significantly more accurate than the others, a naïve majority vote does not fully exploit that strength; a strong model’s vote can still be outvoted by a group of weaker models. Likewise, if multiple models make correlated errors on a certain attack type, a voting ensemble will not correct those errors since it has no mechanism to dynamically adjust weights. This is where stacking ensembles (also known as stacked generalization) offer a more adaptive approach. In a stacking ensemble, the outputs of the base classifiers (either their predicted classes or probability scores) are used as features to train a meta-classifier, which learns how to best combine the base learners’ decisions. Unlike voting, which uses a fixed combination rule, stacking learns from data which models to trust more in different scenarios. For example, a stacking meta-model could learn that if models A and B predict “benign” but model C predicts “malicious,” and model C has historically been more reliable for that type of flow, then the meta-classifier should output “malicious.” The meta-classifier (sometimes called a level-2 model) is typically trained on a separate validation set or via cross-validation to avoid overfitting, ensuring that it generalizes well.

Stacking has been explored in prior IDS studies as well. Early works often used a simple meta-learner like logistic regression to combine diverse base classifiers, essentially learning a weighted voting scheme where the weights are optimized during training. Recent investigations have gone further by using more complex meta-learners, including gradient boosting machines or neural networks, to maximize the ensemble's accuracy. The appeal of stacking, especially for encrypted traffic analysis, is its flexibility; it can adaptively emphasize the strengths of each base model on different types of traffic. By training the meta-classifier on outputs from the base models (using held-out data to prevent overfitting), the system effectively learns an optimal fusion of decision criteria tailored to the problem. This data-driven fusion often outperforms any static combination rule. For instance, one study reported that using a lightweight neural network as the meta-learner provided a small but consistent boost in detection performance over conventional voting on an IDS dataset. Such results align with the expectation that a meta-model can intelligently correct the biases or blind spots of individual detectors. In other words, stacking can sometimes achieve higher detection rates than even the best individual base classifier, and also higher than an equal-weight or hand-tuned ensemble.

In summary, ensemble learning offers clear benefits for intrusion detection. Voting ensembles are simple and have been proven to enhance baseline detection by aggregating multiple models' judgments, but they lack flexibility in weighting each model's contribution. Stacking ensembles take the idea a step further by learning how to combine model outputs, typically leading to superior performance at the cost of a more complex training process. Given the heterogeneity of encrypted network traffic and the variety of attack vectors, a stacking approach is well-suited to leverage different models' strengths. This insight motivates the use of a stacking ensemble in the present research for flow-based IDS. By allowing the meta-learner to dynamically weight the decisions of the base classifiers, the system can often outperform any single model or fixed-rule ensemble, providing a powerful and adaptive defense against attacks in fully encrypted network environments.

III. Methodology

This section describes the methodology used to develop and evaluate the flow-based intrusion detection system. An overview of the key steps taken in this study is illustrated in the pipeline below:



3.1 Dataset and Feature Selection

The experiments used the CIC-IDS2018 dataset, which contains labeled network flow records across multiple days with approximately 20% of flows being malicious and the remainder benign. Three days of data were excluded: 02-20-2018.csv due to its large file size, and 02-28-2018.csv and 03-01-2018.csv due to incomplete flows that could have introduced bias or inaccuracies. The remaining seven days formed the basis for all analyses. From each CSV file, only numeric flow features were retained: non-numeric columns were dropped and all entries converted to numeric values. Any rows containing invalid values (NaN or infinities) were removed. This process yielded 75 valid numeric features per flow, plus the original Attack Type label. Each flow was assigned a binary label for classification: benign flows were encoded as 0 and all attack flows as 1. (The original string “Attack Type” was preserved in its own column but not used as a feature.)

The 75 numeric features were organized into six conceptual groups for analysis. Table 1 presents these feature groups along with their corresponding features. For instance, the flow metrics group includes features like Flow Duration and Flow IAT Mean; the packet size statistics group includes features such as Tot Fwd Pkts and Pkt Len Mean; the timing/IAT group captures inter-arrival statistics (e.g. Fwd IAT Tot); the flags and protocol group covers TCP flag counts and protocol identifiers (e.g. ACK Flag Cnt, Protocol); the rates and ratios group includes traffic rate measures (e.g. Down/Up Ratio, Fwd Pkts/s); and the connection activity group comprises stateful metrics (e.g. Active Mean, Idle Max).

Feature Group	Features
Flow_metrics (7)	"Flow Duration", "Flow Byts/s", "Flow Pkts/s", "Flow IAT Mean", "Flow IAT Std", "Flow IAT Max", "Flow IAT Min"
Packet_size_stats (20)	"Tot Fwd Pkts", "Tot Bwd Pkts", "TotLen Fwd Pkts", "TotLen Bwd Pkts", "Fwd Pkt Len Max", "Fwd Pkt Len Min", "Fwd Pkt Len Mean", "Fwd Pkt Len Std", "Bwd Pkt Len Max", "Bwd Pkt Len Min", "Bwd Pkt Len Mean", "Bwd Pkt Len Std", "Pkt Len

	Min", "Pkt Len Max", "Pkt Len Mean", "Pkt Len Std", "Pkt Len Var", "Pkt Size Avg", "Fwd Seg Size Avg", "Bwd Seg Size Avg"
Timing_iat (10)	"Fwd IAT Tot", "Fwd IAT Mean", "Fwd IAT Std", "Fwd IAT Max", "Fwd IAT Min", "Bwd IAT Tot", "Bwd IAT Mean", "Bwd IAT Std", "Bwd IAT Max", "Bwd IAT Min"
Flags_and_protocol (16)	"FIN Flag Cnt", "SYN Flag Cnt", "RST Flag Cnt", "PSH Flag Cnt", "ACK Flag Cnt", "URG Flag Cnt", "CWE Flag Count", "ECE Flag Cnt", "Fwd PSH Flags", "Bwd PSH Flags", "Fwd URG Flags", "Bwd URG Flags", "Protocol", "Dst Port", "Init Fwd Win Byts", "Init Bwd Win Byts"
Rates_and_ratios (9)	"Down/Up Ratio", "Fwd Pkts/s", "Bwd Pkts/s", "Fwd Byts/b Avg", "Fwd Pkts/b Avg", "Fwd Blk Rate Avg", "Bwd Byts/b Avg", "Bwd Pkts/b Avg", "Bwd Blk Rate Avg"
Connection_activity (13)	"Subflow Fwd Pkts", "Subflow Fwd Byts", "Subflow Bwd Pkts", "Subflow Bwd Byts", "Fwd Act Data Pkts", "Active Mean", "Active Std", "Active Max", "Active Min", "Idle Mean", "Idle Std", "Idle Max", "Idle Min"

Table 1: Feature Groups and Corresponding Features.

Each feature was used in its original scale (no additional normalization was applied beyond the aforementioned type conversion). The features were grouped based on logical similarity to facilitate structured experimentation across subsets; however, no model was ever trained on all 75 features at once (each model operated on only one feature group at a time).

3.2 Experimental Protocols

Two evaluation protocols were employed: same-day and cross-day. Under the same-day protocol, each day's cleaned data was split into a 70% training set and a 30% hold-out test set (stratified by label to preserve class proportions). From the training set, a 50% random downsampling was applied to reduce data volume due to hardware constraints. In code, this procedure was implemented as:

```
X_tr, X_te, y_tr, y_te = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
```

```
)
tmp = apply_downsample(pd.concat([X_tr, y_tr], axis=1), frac=0.5)
X_train = tmp.drop('Label', axis=1)
y_train = tmp['Label']
X_test, y_test = X_te, y_te
```

Here, train_test_split with a fixed random seed (42) ensured reproducibility, and apply_downsample performed uniform random sampling of half the rows. No further stratification was applied during downsampling.

Under the cross-day protocol, a leave-one-day-out strategy was used: for each day held out as the test set, the data from the other six days were combined to form the training pool. This combined training set was then randomly downsampled to 20% of its original size to manage computational load. In particular, all other days' CSVs were concatenated, and 20% of the rows were randomly sampled:

```
train_dfs = [load_cleaned_csv(fp) for fp in all_files if fp !=
file_fp]
df_tr = pd.concat(train_dfs, ignore_index=True)
tmp = apply_downsample(df_tr, frac=0.2)
X_train = tmp[feats];  y_train = tmp['Label']
df_te = load_cleaned_csv(file_fp)
X_test = df_te[feats];  y_test = df_te['Label']
```

The held-out day's data was used as the test set with no downsampling, ensuring a strict evaluation on entirely unseen data.

In both protocols, class weighting was applied during model training to counter residual class imbalance. This technique increased the penalty for misclassifying minority-class examples by assigning higher weight to their loss during optimization. The weights were computed inversely proportional to class frequencies using the formula:

$$\text{class\_weight}[c] = (N_{total} / 2N_c)$$

for class $c \in \{0,1\}$, where N_c is the number of training samples of class c. In code, for example, after the training labels y_train were prepared, the following was set:

```
neg, pos = (y_train==0).sum(), (y_train==1).sum()
total = neg + pos
```

```
class_weight = {0: total/(2*neg), 1: total/(2*pos)}
```

Given the naturally skewed class distribution, class weighting encouraged the models to focus more on accurately classifying the less-frequent attack flows.

3.3 Model Training and Evaluation

3.3.1 Base Models

Ten classifiers were evaluated as base models across both evaluation protocols: Logistic Regression, Decision Tree, Random Forest, Extra Trees, XGBoost, LightGBM, CatBoost, k-Nearest Neighbors, Linear SVM, and Multi-Layer Perceptron. Each model was trained on each of the six feature subsets and evaluated across all 14 train-test scenarios, with seven distinct days evaluated under both the same-day and cross-day protocols. This setup ensured a thorough examination of how each model performed across different data splits and feature perspectives.

Hyperparameter tuning was conducted using grid search with three-fold cross-validation. Each trial was executed using the following procedure:

```
gs = GridSearchCV(model, param_grid, cv=3, scoring='f1', n_jobs=1)
gs.fit(X_train, y_train)
best = gs.best_estimator_
y_pred = best.predict(X_test)
```

The optimization objective was the F1 score, selected to balance precision and recall, particularly in light of the malicious class being the minority in most cases. Prioritizing the F1 score helped the models remain sensitive to both false positives and false negatives, improving their effectiveness in this imbalanced detection task.

The hyperparameter search grids used for each model are shown in table 2 below:

Classifier	Fixed Parameters	Tuned Parameters (Grid)
Logistic Regression	solver='saga', max_iter=1000, random_state=42	C: [0.01, 0.1, 1]
Decision Tree	random_state=42	max_depth: [5, 10, 20]

Random Forest	random_state=42, n_jobs=1	n_estimators: [100, 200]; max_depth: [None, 10, 20]
Extra Trees	random_state=42, n_jobs=1	n_estimators: [100, 200]; max_depth: [None, 10, 20]
XGBoost	use_label_encoder=False eval_metric='logloss', random_state=42, n_jobs=1	learning_rate: [0.1, 0.01]; n_estimators: [100, 200]; max_depth: [3, 5]
LightGBM	random_state=42, n_jobs=1, verbose=-1	learning_rate: [0.1, 0.01]; n_estimators: [100, 200]; num_leaves: [31, 50]
CatBoost	verbose=0, random_state=42	depth: [4, 6]; iterations: [100, 200]; learning_rate: [0.1, 0.01]
k-Nearest Neighbors	—	n_neighbors: [3, 5, 7]
Linear SVM	max_iter=10000, random_state=42	C: [0.1, 1, 10]
MLP	max_iter=500, random_state=42	hidden_layer_sizes: [(50,), (100,)]; alpha: [1e-4, 1e-3]

Table 2: Base classifier hyperparameter search grids

Each trained model was evaluated on the hold-out test set, and a range of metrics was recorded, including accuracy, precision, recall, F1 score, ROC-AUC, and PR-AUC. This comprehensive baseline study was conducted to identify the four most effective base-model and feature-group combinations to serve as candidates for the stacked meta-ensemble classifier.

3.3.2 Stacked Meta-Model

Following the baseline evaluation, four base classifiers were selected along with their top-performing feature subsets. The selection criteria were based on achieving the highest average F1 score across all evaluation scenarios while maintaining relatively efficient training time. These base models served as the foundation for the subsequent meta-ensemble experiments.

Two ensemble strategies were implemented: voting and stacking. For voting, both hard and soft variants were evaluated. In hard voting, the predicted class label was determined by a majority vote across the base models. In soft voting, the predicted probabilities from each base model were averaged, and the final prediction was obtained by thresholding this average at 0.5. The implementation of both strategies is shown below:

```
# Hard voting: majority vote
votes = df_test[[f"label_{m}" for m in BASE_MODELS]].values
hard_pred = (votes.sum(axis=1) >= (len(BASE_MODELS) / 2)).astype(int)

# Soft voting: average probability
probs = df_test[[f"pred_{m}" for m in BASE_MODELS]].values
soft_score = probs.mean(axis=1)
soft_pred = (soft_score >= 0.5).astype(int)
```

In the stacking configuration, each meta-classifier was trained using the predicted scores of the selected base models on the training split. Evaluation was performed on the same held-out test set used for the base-model evaluation, using the corresponding base-model predictions as input features. This setup ensured a consistent comparison between stacking and individual base models.

The meta-classifiers included ten candidates: Logistic Regression, Decision Tree, Random Forest, Extra Trees, XGBoost, LightGBM, CatBoost, k-Nearest Neighbors, Linear SVM, and Multi-Layer Perceptron. Each was trained independently on the meta-feature set for every day and protocol. An example of the training and evaluation procedure is shown below:

```
X_tr_meta = df_train_preds[['pred_Model1', 'pred_Model2', ...]]
X_te_meta = df_test_preds[['pred_Model1', 'pred_Model2', ...]]
meta_model = META_MODELS[name]
meta_model.fit(X_tr_meta, y_tr)
y_pred = meta_model.predict(X_te_meta)
```

Each configuration was evaluated using the same metrics recorded in the base model study: accuracy, precision, recall, F1 score, ROC-AUC, and PR-AUC. This experiment was conducted both to assess whether stacking outperforms simpler ensemble techniques such as hard or soft voting, and to identify the most effective meta-classifier architecture for use in the subsequent ablation and feature augmentation experiments.

3.3.3 Meta-Model Ablation Studies

An ablation study was conducted to evaluate the contribution of each base classifier to the overall performance of the stacked ensemble. For this experiment, the meta-classifier was held fixed while the input feature set, comprising the predicted scores from the base models, was selectively modified. Specifically, each base model was removed one at a time from the meta-feature set, and performance was compared against a baseline configuration in which all base models were included.

Concretely, for a given ablation, only the prediction scores from the remaining base models (e.g., pred_Model1, pred_Model2, pred_Model3) were passed to the meta-classifier during training and evaluation. This was implemented as follows:

```
cols = [f"pred_{m}" for m in BASE_CLASSIFIERS if m != drop]
X_tr = train_df[cols].values
y_tr = train_df["y_true"].values
X_te = test_df[cols].values
y_te = test_df["y_true"].values
meta = META_MODEL
meta.fit(X_tr, y_tr)
y_pred = meta.predict(X_te)
```

Each configuration was evaluated under both same-day and cross-day protocols, across all available days. The evaluation included the same set of performance metrics used throughout the study: accuracy, precision, recall, F1 score, ROC-AUC, and PR-AUC. This ablation study was designed to determine the individual influence of each base model on the ensemble's overall performance, and to assess whether any single model was particularly redundant or indispensable in the context of stacked learning.

3.3.4 Meta-Feature Addition Experiments

This experiment examined whether augmenting the stacked ensemble with additional raw features could improve classification performance. Each trial involved adding one of the 75 available numeric features to the set of meta-inputs (consisting of the base model prediction scores) and evaluating its effect across each day under both evaluation protocols.

Implementation details varied slightly by protocol. For the same-day setup, a train-test split was performed, and the selected raw feature was appended to the meta-feature matrix using index alignment with the original flow data:

```
raw_df = pd.read_csv(f"{day}.csv")
X_tr[feat] = raw_df[feat].iloc[train_df.index].values
X_te[feat] = raw_df[feat].iloc[test_df.index].values
```

For the cross-day setup, a leave-one-day-out protocol was followed. The training feature values were drawn from all days except the one held out, and the test feature values were extracted from the current day:

```
raw_full = pd.concat([pd.read_csv(f"{d}.csv") for d in other_days],
ignore_index=True)
raw_tr = raw_full.reset_index(drop=True)
X_tr[feat] = raw_tr[feat].values

raw_te = pd.read_csv(f"{day}.csv")
X_te[feat] = raw_te[feat].values
```

Each configuration was evaluated using the same set of metrics as the previous experiments: accuracy, precision, recall, F1 score, ROC-AUC, and PR-AUC. The goal of this experiment was to determine whether incorporating individual raw features could enhance the meta-classifier's ability to distinguish between benign and malicious flows beyond what was already captured by the base model outputs.

In summary, the methodology strictly followed the outlined protocols: data selection and cleaning were handled in prepare_datasets.py, and the experimental studies (base models, stacking, ablation, and feature-addition) were conducted by their corresponding Python scripts. The experimental design parameters (splits, downsampling rates, class weights, model lists, etc.) are explicitly implemented in the code and cited above. All code snippets shown are taken directly from the implementation to illustrate critical steps. This ensures that the reported methodology exactly matches the computations performed in the experiments.

IV. Results

This section reports model performance under the same-day and cross-day protocols, focusing on metrics including Accuracy, Precision, Recall, F1, ROC-AUC, and PR-AUC.

We first summarize the base models, then evaluate the meta-model, ablations, and feature additions.

4.1 Base Model Study

Ten classifiers were evaluated as base learners on six distinct feature groups across both same-day and cross-day protocols. Table 3 summarizes the top 10 model–feature combinations for each protocol, ranked by F1 score.

Classifier	Feature Group	Accuracy	Precision	Recall	F1
CatBoost	flags_and_protocol	0.9997	1	0.9634	0.979
LightGBM	flags_and_protocol	0.9997	0.9985	0.9637	0.9787
DecisionTree	flags_and_protocol	0.9997	0.9978	0.9634	0.9782
XGBoost	flags_and_protocol	0.9997	0.9967	0.9634	0.9778
RandomForest	flags_and_protocol	0.9997	0.9978	0.9621	0.9776
ExtraTrees	flags_and_protocol	0.9997	0.9967	0.9612	0.9766
KNN	flags_and_protocol	0.9997	0.9987	0.9521	0.9723
ExtraTrees	flow_metrics	0.9853	0.9677	0.8277	0.8746
XGBoost	flow_metrics	0.9855	0.9821	0.8233	0.8727
LightGBM	flow_metrics	0.9857	0.9744	0.8194	0.8679

Table 3.1 Top 10 Base Model Results (Same Day)

Classifier	Feature Group	Accuracy	Precision	Recall	F1
DecisionTree	flags_and_protocol	0.8341	0.6942	0.5543	0.6075
ExtraTrees	flags_and_protocol	0.8385	0.6759	0.5626	0.6054
KNN	flags_and_protocol	0.8395	0.73	0.5727	0.6024
RandomForest	flags_and_protocol	0.8391	0.674	0.5594	0.6007

XGBoost	flags_and_protocol	0.841	0.6438	0.5877	0.6004
LightGBM	flags_and_protocol	0.8338	0.6445	0.513	0.5631
LightGBM	rates_and_ratios	0.928	0.5524	0.5332	0.5132
CatBoost	rates_and_ratios	0.9231	0.553	0.5209	0.5088
XGBoost	rates_and_ratios	0.9226	0.552	0.5192	0.5071
CatBoost	flags_and_protocol	0.8623	0.6569	0.438	0.4949

Table 3.2 Top 10 Base Model Results (Cross Day)

Overall, ensemble tree-based models achieved the highest classification performance. CatBoost, LightGBM, and XGBoost reached high same-day F1 scores between 0.977 and 0.979 using the Flags and Protocols feature group, outperforming simpler classifiers such as k-NN (which peaked at 0.972 with the same feature group). In contrast, cross-day performance declined notably: models like Decision Tree only reached an F1 score of about 0.607, and CatBoost dropped to 0.494, despite being trained on the Flags and Protocols feature group. These results indicate that more complex learners, especially tree-based methods, are better equipped to model network flow data patterns under same-day evaluation, while their generalization across days remains more limited.

Another clear trend is the variability of performance across feature groups. Certain subsets proved far more discriminative than others, as illustrated by Figure 1.

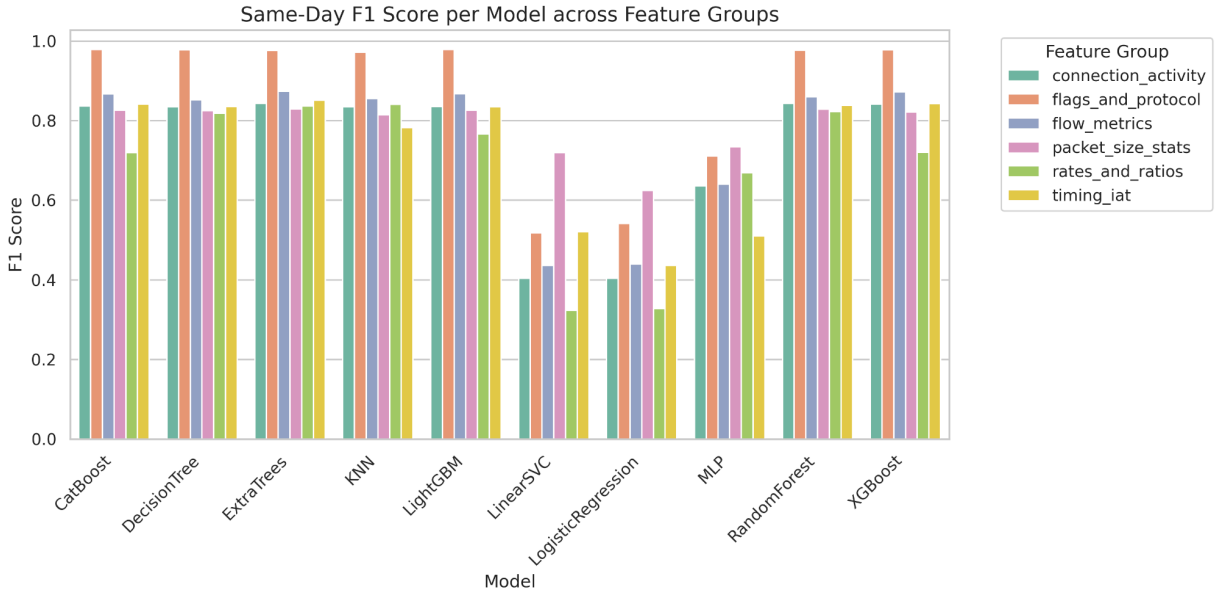


Figure 1.1 F1 Score Per Model Across Feature Groups (Same Day)

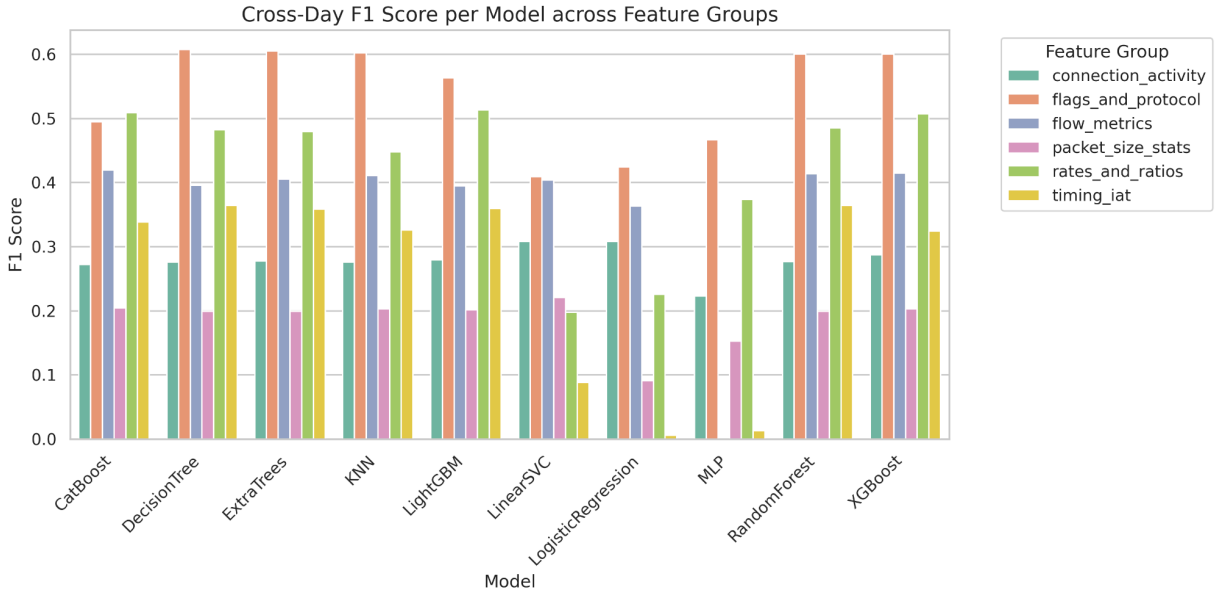


Figure 1.2 F1 Score Per Model Across Feature Groups (Cross Day)

Most classifiers (7 out of 10 in the same-day tests and 9 out of 10 in the cross-day tests) achieved their highest F1 scores using the Flags and Protocols feature group. This feature group consistently yielded the highest average F1 for a majority of models in both same-day and cross-day settings. Conversely, groups like Timing_IAT and Packet

Size Stats resulted in noticeably lower performance. Another observation is that overall F1 scores declined in the cross-day setting due to a temporal distribution shift, as shown in Figure 2.

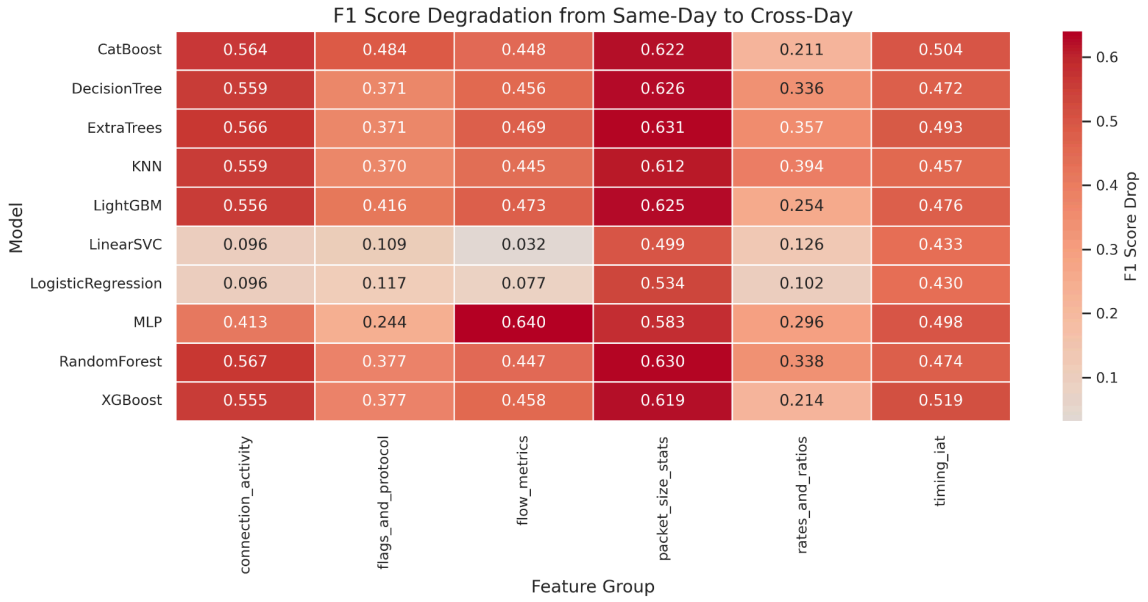


Figure 2. F1 Score Degradation between Same-Day and Cross-Day Protocols

The Flags and Protocols and Rates and Ratios groups exhibited the least degradation in F1 score, demonstrating stronger cross-day generalization compared to more volatile feature groups such as Packet Size Stats and Connection Activity. Taken together, these trends highlight that tree-based models such as LightGBM, XGBoost, Random Forest, and Decision Tree, when trained on stable feature groups like Flags and Protocols, performed well across both same-day and cross-day settings. Their strong and reliable F1 scores led to these models being selected as the base learners for subsequent ensemble experiments.

4.2 Meta Model Study

Using the four top base learners (Decision Tree, Random Forest, XGBoost, LightGBM) as ensemble constituents, stacking-based meta-models were compared against traditional voting ensembles. Table 4 summarizes the overall metrics for each meta-classifier and voting method under both protocols.

Model Name	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
KNN	0.9997	0.9997	0.9637	0.9791	0.9819	0.9816
MLP	0.9997	0.9997	0.9637	0.9791	0.9999	0.9837
XGBoost	0.9997	0.9978	0.9634	0.9782	0.9991	0.9832
LinearSVC	0.9997	0.9967	0.9634	0.9778	0.9999	0.9832
LightGBM	0.9997	0.9942	0.9637	0.9771	0.9995	0.9829
CatBoost	0.9997	0.9946	0.9634	0.977	0.9999	0.9831
DecisionTree	0.9997	0.9946	0.9634	0.977	0.9995	0.9811
ExtraTrees	0.9997	0.9946	0.9634	0.977	0.9995	0.9822
RandomForest	0.9997	0.9946	0.9634	0.977	0.9995	0.9828
Soft Voting	0.9994	0.8882	0.9928	0.9075	0.9999	0.9837
Hard Voting	0.9966	0.8606	0.9987	0.8636	0.9978	0.9297
LogisticRegression	0.9996	0.7143	0.7138	0.714	0.9999	0.9837

Table 4.1 Overall Metrics for each meta classifier and voting scheme (Same Day)

Model Name	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
MLP	0.8578	0.8109	0.6101	0.6728	0.7649	0.6941
ExtraTrees	0.8607	0.8375	0.5954	0.6676	0.7576	0.7252
LightGBM	0.8599	0.8379	0.5823	0.6592	0.7855	0.765
LinearSVC	0.8356	0.6936	0.5784	0.6219	0.8256	0.771
LogisticRegression	0.8361	0.6845	0.5854	0.6218	0.8255	0.7297
Soft Voting	0.8362	0.6838	0.5856	0.6215	0.8258	0.7233
RandomForest	0.8392	0.6951	0.5649	0.6147	0.7578	0.7248

CatBoost	0.8341	0.6947	0.5616	0.6129	0.8133	0.762
XGBoost	0.8385	0.695	0.5614	0.6129	0.7931	0.7201
KNN	0.8376	0.6941	0.5524	0.6063	0.7981	0.7674
DecisionTree	0.8386	0.695	0.5499	0.6052	0.7934	0.7347
Hard Voting	0.8497	0.59	0.6303	0.598	0.8087	0.6831

Table 4.2 Overall Metrics for each meta classifier and voting scheme (Cross Day)

Stacking with a learned meta-classifier yielded superior results compared to direct voting. The best-performing meta-learner was the Multi-Layer Perceptron (MLP), which achieved the highest average F1 score across all ensemble methods. Under the same-day evaluation, the stacked MLP ensemble attained an F1 of 0.9791, outperforming soft voting (0.9075) and hard voting (0.8636). A similar pattern held in the cross-day tests: MLP stacking reached an F1 of 0.6728, compared to 0.6215 with soft voting and 0.5980 with hard voting. This indicates that the meta-classifier effectively learned how to weight the base model outputs, resulting in an improved balance between precision and recall. Consistently, MLP stacking also achieved some of the highest ROC-AUC and PR-AUC values in both protocols (reaching 0.7649 and 0.6941 under cross-day evaluation, and 0.9999 and 0.9837 under same-day evaluation), exceeding all voting-based alternatives. Overall, stacking improved detection performance across all metrics, demonstrating a significant advantage over the simpler ensemble techniques.

Interestingly, not all meta-classifiers in the stacking framework performed equally well. Figure 3 illustrates the comparative F1 scores of different meta-models versus voting ensembles under both same-day and cross-day evaluations.

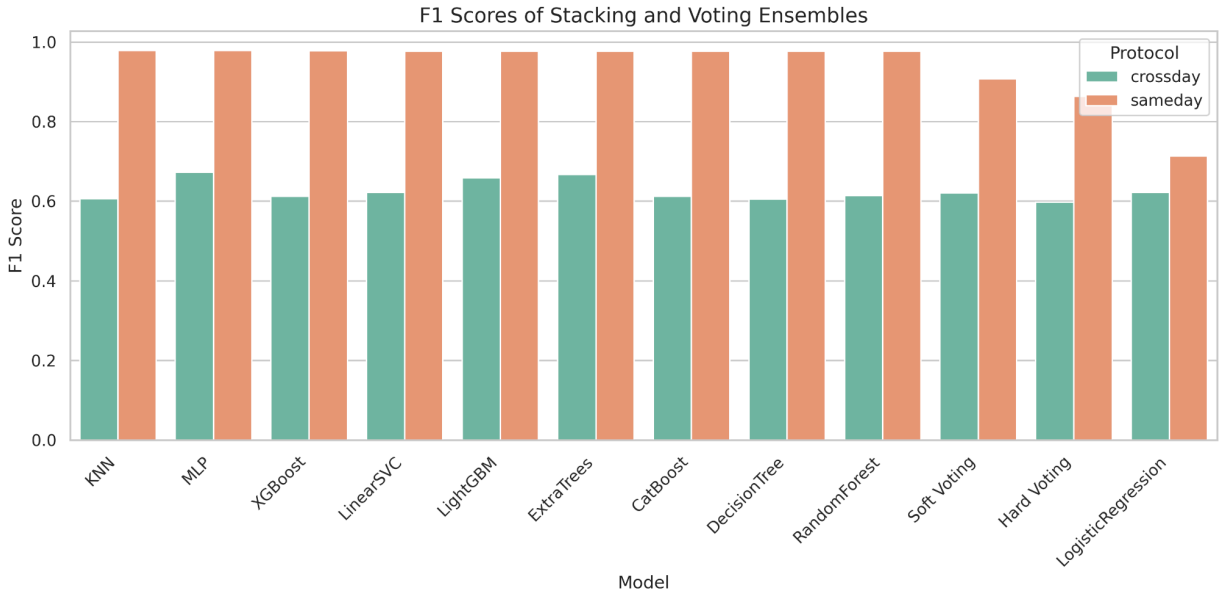


Figure 3. F1 Scores of Stacking and Voting Ensembles

While MLP delivered the strongest results overall, several other meta-models (such as LightGBM and ExtraTrees) also outperformed both hard and soft voting in terms of F1 score across both testing protocols, though by smaller margins. In contrast, simpler architectures like Logistic Regression tended to underperform, occasionally falling below even the hard-voting ensemble in the same-day setting. The MLP stack stands out at the top of both testing protocols, confirming it as the optimal meta-learner in this study.

These results validate that a learned stacking model (particularly the MLP meta-classifier) can better exploit the complementary strengths of base classifiers, thus delivering more robust detection performance than a straightforward majority-voting ensemble.

4.3 Ablation Study

To assess each base classifier's contribution to the ensemble, an ablation experiment was conducted by removing one base model at a time from the stacked MLP ensemble. Table 5 reports the resulting performance metrics for the full ensemble and each ablated variant, across both same-day and cross-day protocols.

Removed Model	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
None	0.9997	0.9997	0.9637	0.9791	0.9999	0.9837
LightGBM	0.9997	0.9997	0.9637	0.9791	0.9999	0.9837
RandomForest	0.9997	0.9996	0.9637	0.9791	0.9999	0.9834
DecisionTree	0.9997	0.9996	0.9624	0.9784	0.9999	0.9835
XGBoost	0.9997	0.8568	0.8557	0.8562	0.9995	0.9803

Table 5.1 Ablation results (Same Day), sorted by F1 score

Removed Model	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
None	0.8578	0.8109	0.6101	0.6728	0.7649	0.6941
XGBoost	0.8605	0.8371	0.5952	0.6675	0.7682	0.7336
LightGBM	0.8572	0.813	0.5965	0.6658	0.82	0.744
DecisionTree	0.853	0.7954	0.5962	0.6633	0.7598	0.693
RandomForest	0.8354	0.6836	0.5848	0.6211	0.8604	0.689

Table 5.2 Ablation results (Cross Day), sorted by F1 score

As expected, the full ensemble (with all four base learners) achieved the highest overall scores, reaching approximately 0.9791 F1, 0.9999 ROC-AUC, and 0.9837 PR-AUC on average for the same-day tests, and 0.6728 F1, 0.7649 ROC-AUC, and 0.6941 PR-AUC for the cross-day tests. Removing certain base learners caused clear performance drops. For instance, dropping XGBoost reduced the same-day F1 to 0.8562. Similarly, removing Random Forest reduced the cross-day F1 to 0.6211. Both of these removals also caused meaningful declines in AUC scores across the testing protocols. In contrast, removing LightGBM or Decision Tree led to much smaller changes. In particular, ablating LightGBM barely impacted the same-day F1 (staying at 0.9791) or the cross-day F1 (from 0.6728 to 0.6658), indicating that it contributed the

least to the ensemble. These results suggest that XGBoost and Random Forest are central to ensemble performance on both seen and unseen days. These trends are further illustrated in Figure 4, which visualizes the F1 score drop resulting from the removal of each base model.

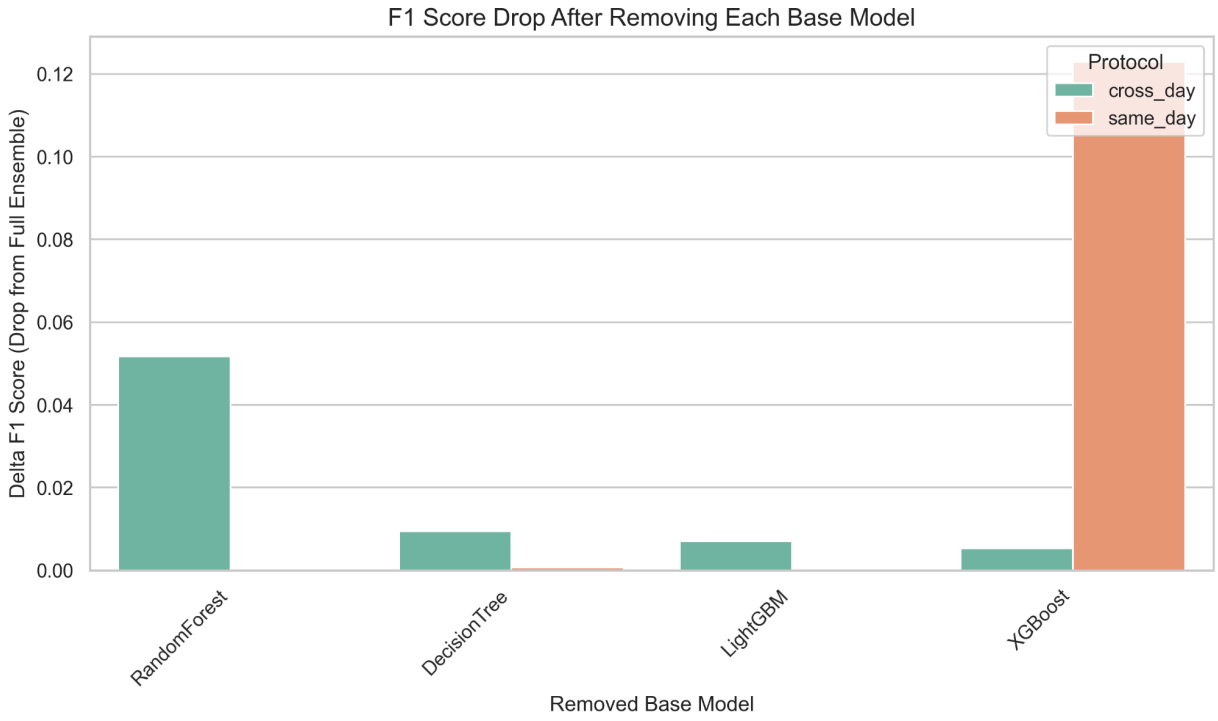


Figure 4. F1 score drop after removing each base model (bar chart)

In the same-day setting (where overall F1 scores were higher), XGBoost emerged as the key contributor: its removal led to a sharp 12% absolute drop in F1, while other models had minimal impact. In contrast, under cross-day evaluation, every base learner played a more pivotal role. Random Forest's removal resulted in about a 5% drop, and smaller yet still meaningful degradations followed from removing Decision Tree and LightGBM. This broader distribution of impact suggests that while same-day performance was largely carried by a single model, generalization across days required the combined strength of multiple diverse learners.

Together, these results reinforce the importance of XGBoost for maximizing same-day performance and highlight the collective value of all four base models, especially Random Forest, for ensuring robust cross-day generalization.

4.4 Feature Addition Study

Finally, a feature addition experiment was performed to explore whether adding individual raw features to the meta-classifier, beyond the base model predictions, could improve ensemble performance. Table 6 presents the top-performing features in this experiment, ranked by F1 score.

Feature	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
ACK Flag Cnt	0.9997	0.9998	0.9635	0.9791	0.9993	0.9828
Fwd PSH Flags	0.9997	1.0	0.9634	0.9791	0.9998	0.9824
CWE Flag Count	0.9997	1.0	0.9634	0.9791	0.9999	0.9836
Down/Up Ratio	0.9997	0.9998	0.9635	0.9791	0.9999	0.9803
ECE Flag Cnt	0.9997	0.9998	0.9635	0.9791	0.9999	0.9824
FIN Flag Cnt	0.9997	1.0	0.9634	0.9791	0.9999	0.9834
Fwd Blk Rate Avg	0.9997	1.0	0.9634	0.9791	0.9999	0.9836
Fwd Byts/b Avg	0.9997	1.0	0.9634	0.9791	0.9999	0.9836
Fwd Pkts/b Avg	0.9997	1.0	0.9634	0.9791	0.9999	0.9836
Bwd Pkts/b Avg	0.9997	1.0	0.9634	0.9791	0.9999	0.9836

Table 6.1 Top feature by F1 Score (Same Day)

Feature	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
ECE Flag Cnt	0.8585	0.8123	0.6218	0.6795	0.7401	0.6916
RST Flag Cnt	0.8585	0.8123	0.6218	0.6795	0.7401	0.6915
URG Flag Cnt	0.8584	0.8124	0.6199	0.6787	0.8087	0.7397
ACK Flag Cnt	0.8584	0.8113	0.6199	0.6782	0.7665	0.7005

Tot Bwd Pkts	0.8583	0.8115	0.6193	0.6780	0.7873	0.7069
Subflow Bwd Pkts	0.8583	0.8115	0.6193	0.6780	0.7873	0.7069
Down/Up Ratio	0.8583	0.8115	0.6192	0.6780	0.7805	0.7310
Fwd Pkt Len Std	0.8583	0.8019	0.6196	0.6738	0.9150	0.8080
None	0.8578	0.8109	0.6101	0.6728	0.7649	0.6941
Pkt Len Std	0.8580	0.7950	0.6202	0.6719	0.8279	0.7234

Table 6.2 Top feature by F1 Score (Cross Day)

While most features had minimal or no effect, a few produced small but consistent gains, particularly in cross-day evaluations. Notably, ACK Flag Cnt maintained the F1 score at 0.9791 in the same-day setting but increased it from 0.6728 to 0.6782 in the cross-day setting. Similarly, ECE Flag Cnt also maintained the same-day F1 at 0.9791 but increased the cross-day F1 to 0.6795. The Down/Up Ratio feature (from the Rates and Ratios group) also preserved performance (approximately 0.968 F1 on average in same-day tests) but raised it to about 0.6780 on average in cross-day tests. Though these gains were modest (generally under a 1% absolute increase), they were consistent across protocols, indicating that these features captured insights complementary to what the base classifiers were already providing. These results suggest that a small number of raw features, particularly those related to directional flow behavior and protocol-level flags, may provide additional value when refining ensemble decisions. To better understand the contribution of each feature, Figure 5 depicts how the F1 score of the stacked model changes as each of the 75 features is added one at a time.

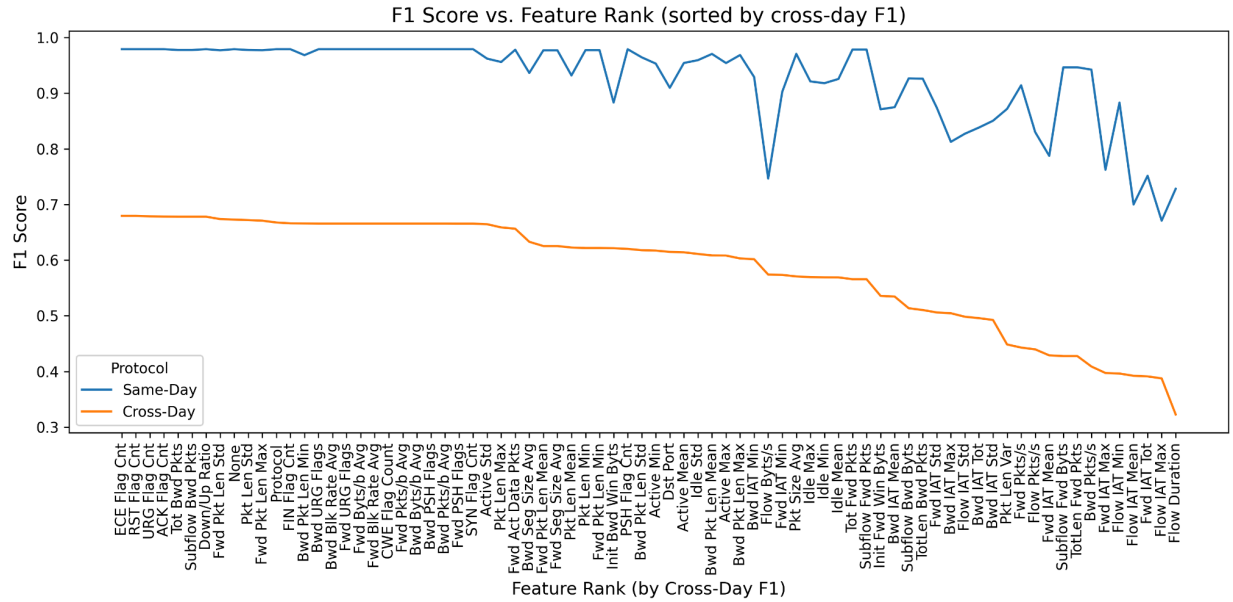


Figure 5. Change in stacked model F1 score as individual features are added

While a small number of features near the top clearly provide marginal improvements, the curve drops off sharply beyond the first ~20 additions, indicating that most features actually have a negative impact on performance. Rather than plateauing, this decline suggests that many raw inputs introduce noise or redundancy, ultimately degrading the predictive power of the ensemble. This observation reinforces the notion that only a narrow subset of well-aligned features, especially those tied to protocol-level signals or directional flow dynamics, offer meaningful benefit when augmenting the ensemble.

Together, these findings indicate that while feature augmentation can offer minor enhancements, the core effectiveness of the stacked model is already captured by the base classifiers and their selected feature groups.

V. Discussion

The experimental results indicate that flow-based detection using an ensemble is highly effective on modern benchmark data. The stacking ensemble achieved near-perfect detection on CIC-IDS2018 in same-day testing (F1 ≈ 0.979) and maintained solid performance across different days (cross-day F1 ≈ 0.673), outperforming individual classifiers and traditional voting schemes. This confirms that a learned meta-classifier can better balance precision and recall than fixed fusion rules, yielding more robust detection across diverse attack types. Compared to prior works on CIC-IDS2018 and

similar datasets, which report high in-sample accuracy with single models or basic ensembles, the proposed approach offers improved adaptability. For example, machine learning models have previously shown strong accuracy on encrypted traffic, but often without addressing temporal shifts. In our case, the stacking method not only matches state-of-the-art detection rates reported in literature but also sustains higher performance compared to individual classifiers when confronted with completely unseen daily traffic. This suggests the ensemble can generalize better to new conditions, an important advantage for real-world deployment.

The findings also highlight which components contributed most to success. Tree-based algorithms (especially XGBoost and Random Forest) were key drivers of ensemble performance, aligning with other studies that emphasize the strength of gradient boosting and forests in intrusion detection. The flags & protocol features proved most discriminative, consistent with reports that metadata like header flags carry significant attack signatures. Meanwhile, adding raw features to the meta-classifier yielded only marginal gains; a few features (e.g. ACK flag count, Down/Up ratio) slightly improved cross-day recall, but most features provided no benefit or introduced noise. This underscores that the core feature groups were already informative and that the ensemble was largely capturing the relevant information.

Despite its strong performance, the proposed approach has several practical limitations. Firstly, the dataset had to be downsampled due to hardware constraints, meaning the models were trained on only a fraction of available data. While necessary to manage memory and computation, this downsampling may omit some traffic patterns and limit the absolute performance. Minimal hyperparameter tuning was performed (using only small grid searches), so the results might be further improved with more extensive optimization. Additionally, no deep packet features or advanced feature engineering beyond basic grouping were applied; the approach relies purely on provided flow features without normalization, which could affect algorithms sensitive to feature scale. Another limitation is that the evaluation, although comprehensive on CIC-IDS2018, reflects one particular environment. The ensemble's effectiveness against completely novel (zero-day) attacks was not explicitly tested; like most supervised models, it may struggle with threats that differ significantly from the training data.

Future Work: Building on this study, several enhancements are planned to address the above limitations and further improve the system:

- Dimensionality Reduction: Apply PCA or other feature engineering techniques to reduce redundant features and improve model efficiency.

- Model Explainability: Integrate SHAP or similar explainable AI methods to interpret feature importance and explain the ensemble's decisions for better transparency.
- Handling Imbalance: Use techniques like SMOTE or targeted oversampling to bolster minority attack classes, complementing the class-weighting approach and improving recall for rare attacks.
- Feature Normalization: Investigate normalizing or scaling flow features, which may benefit algorithms like k-NN or SVM and lead to more consistent performance across features.
- Cross-Dataset Evaluation: Test the ensemble on new datasets (e.g., other recent IDS benchmarks) to verify its generalization to different network environments and attack scenarios.
- Zero-Day Detection: Explore hybrid models that combine the ensemble with anomaly detection or online learning to identify zero-day attacks, ensuring the IDS remains effective against emerging threats.

These future directions aim to enhance the model's accuracy, generality, and interpretability, moving the solution closer to real-world deployment in diverse settings.

IV. Conclusion

This work introduced a privacy-preserving, flow-based intrusion detection solution using an ensemble of machine learning classifiers in a two-tier stacking architecture. The proposed system demonstrated improved detection performance over individual models and traditional voting ensembles on a modern encrypted traffic dataset, achieving high recall and precision across a broad range of attack types. Key contributions include establishing baseline performance for various algorithms on encrypted flows, quantifying base learner importance via ablation, and showing the benefit of trainable fusion in the ensemble. The practical value of this approach is significant: by inspecting only flow metadata and not packet payloads, it operates without decrypting traffic, thereby preserving user privacy and simplifying compliance with data protection requirements. The results also show that the ensemble can adapt to temporal variations in network traffic, indicating its suitability for deployment in dynamic real-world networks.

References:

[1] Kang, M. et al. (2017). Encrypted Traffic Analysis: A New Blind Spot for Intrusion Detection. Proceedings of the IEEE Conference on Communications and Network Security (CNS), 415–418.

[2] Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A.A. (2012). Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection. Computers & Security, 31(3), 357–374.

[3] Ibraheem, H.R., Zaki, N.D., & Al-Mashhadani, M.I. (2022). Anomaly Detection in Encrypted HTTPS Traffic Using Machine Learning: A Comparative Analysis of Feature Selection Techniques. Mesopotamian Journal of Computer Science, 2022(1), 18–28.

[4] Singh, K., Kashyap, A., & Cherukuri, A.K. (2025). Interpretable Anomaly Detection in Encrypted Traffic Using SHAP with Machine Learning Models. arXiv preprint arXiv:2505.16261.

[5] Guo, Y. (2023). A Survey of Machine Learning-Based Zero-Day Attack Detection: Challenges and Future Directions. Computer Communications, 198, 50–66.

[6] Emanet, S., Karataş Baydoğan, G., & Demir, O. (2023). An Ensemble Learning Based IDS Using Voting Rule: VEL-IDS. PeerJ Computer Science, 9:e1553. https://doi.org/10.7717/peerj-cs.1553

[7] Sharafaldin, I., Lashkari, A.H., & Ghorbani, A.A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP), 108–116.