Lightweight Supervised Learning for Network Intrusion Detection: Experimental Modeling and Case Study on CICIDS2017

Mocan Sergiu Ioan

Abstract

Modern networks require intrusion detection systems (IDS) that not only achieve high attack detection rates but also operate under real-time constraints on limited hardware. In this paper, we investigate a lightweight, feature-based supervised learning approach to network intrusion detection. We focus on two efficient classifiers – Logistic Regression and Random Forest – and evaluate their performance on the CICIDS2017 benchmark dataset, which provides richly detailed flow-level data for both benign traffic and a variety of attacks. We describe our experimental modeling hypothesis and methodology, then present a case study with a 50,000-flow sample from CICIDS2017, detailing preprocessing steps (handling missing values and infinities), training-validation-test regimen, and performance metrics including precision, recall, F1-score, confusion matrices, and inference speed. The results show that a Random Forest model can achieve near-perfect detection (e.g., ~99% F1) on major attack categories, outperforming Logistic Regression which reaches high accuracy on common attacks but struggles with more subtle intrusions. Random Forest's improved accuracy comes at a modest cost in speed, as Logistic Regression demonstrates faster inference (processing over one million flows per second) suitable for high-throughput scenarios. We discuss these trade-offs and position our approach relative to existing research in the field. Finally, we outline a validation plan for deploying and testing the models in real-world settings or on additional datasets, to ensure the approach generalizes and remains robust in operational environments.

Keywords: network intrusion detection, machine learning, lightweight models, logistic regression, random forest, CICIDS2017, real-time inference, security

Introduction

Network Intrusion Detection Systems (NIDS) are critical for defending organizational and IoT networks against malicious activities. Traditional NIDS approaches often rely on either signature-based methods (which struggle with novel attacks) or computationally heavy machine learning models such as deep neural networks. **Deep learning methods have achieved strong accuracy on intrusion detection tasks** in recent studies, but their high computational cost and complexity can hinder deployment on resource-constrained devices or real-time applications[1][2]. This drives the need for **lightweight, efficient ML models** that can swiftly analyze network traffic and detect attacks with minimal resources. Prior work has shown that using **compact flow features** and classical machine learning algorithms can deliver excellent detection performance at a fraction of the cost of deep models [1][2]. In particular, techniques like Random Forests and logistic regression have been applied to flow-based IDS data with competitive results [3][4]. These classical models also offer advantages in interpretability (feature importance, simpler decision boundaries) and ease of deployment compared to deep neural networks.

However, balancing **detection effectiveness and efficiency** remains a challenge. Highly accurate models may still produce too much latency or consume too much memory for real-time use, whereas extremely fast models might miss sophisticated attacks. The research question we address is: *Can lightweight supervised learning models, using a carefully chosen set of features, achieve high intrusion detection rates while maintaining real-time inference speeds suitable for operational deployment?* We hypothesize that the right combination of features and algorithms can yield **near state-of-the-art detection performance** (e.g., precision/recall above 95% for major attack categories) with **low latency per prediction (on the order of milliseconds or less)**, validating the viability of these methods for practical NIDS.

To explore this hypothesis, we conduct an experimental case study using the **CICIDS2017 dataset** [5], a comprehensive benchmark containing diverse attack scenarios and normal traffic. We limit the study to a manageable subset (50,000 network flows) to illustrate our methodology and evaluate two representative classification models: **Logistic Regression (LR)** as a simple linear baseline, and **Random Forest (RF)** as a more expressive ensemble model. We detail the modeling approach, including data preprocessing (e.g., normalization and handling of missing values), model training, and evaluation metrics geared towards both detection accuracy and computational efficiency. **Figure 1** provides a conceptual overview of the intrusion detection pipeline we investigate, from network flow capture and feature extraction to real-time classification by the learned model *(Figure 1: Data flow diagram of the proposed lightweight NIDS approach* – placeholder*)*. The experiments compare LR and RF on multiple attack classes, analyzing their precision, recall, F1-score, and confusion matrices, as well as measuring inference throughput to assess real-time performance.

In summary, **our contributions** are:
- We develop a **lightweight supervised NIDS framework** and describe an experimental modeling strategy that emphasizes both detection performance and deployment feasibility (e.g., speed and simplicity).
- We present a **case study on CICIDS2017** with a detailed performance comparison between a linear model and a tree ensemble, highlighting how a Random Forest significantly improves detection of complex attacks (boosting F1 from ~0.73 to ~0.92 macro-average) at the cost of some additional inference latency.
- We provide a **discussion of results and trade-offs**, connecting them to related work [1][2][4] and showing that our findings align with and extend prior research on feature-based intrusion detection. We also outline a **validation plan** for applying the models in real-world conditions or across other datasets, to ensure the approach's robustness and to facilitate direct comparison with state-of-the-art methods.

The remainder of this paper is structured as follows. **Chapter 1** describes the experimental modeling in detail, including the research hypothesis, goals, and methodological approach. **Chapter 2** presents the case study on the CICIDS2017 data: we describe the dataset and preprocessing, experimental setup, and then discuss the results (with tables and figures illustrating key metrics). **Chapter 3** compares our work to related research and proposes a plan for further validation in operational environments or on additional benchmark datasets. We conclude with a summary of findings and implications for deploying lightweight IDS models.

Chapter 1: Experimental Modeling

In this chapter, we outline the design of our experimental study, including the underlying hypothesis, research objectives, modeling strategy, and the methods chosen to test our hypothesis. We formalize how the experiment is structured and justify the key decisions in terms of data, algorithms, and evaluation criteria.

Research Hypothesis and Goals

The core hypothesis driving our research is that a **small set of flow-level features combined with efficient machine learning models can detect common network attacks with high accuracy, while operating within real-time constraints**. In other words, we expect that even **“lightweight” classifiers (Logistic Regression, Random Forest)**, when trained on carefully engineered network flow features, will be capable of achieving detection performance on par with more complex models (approaching 95–99% accuracy/F1 on major attack classes) but with significantly lower computational cost. This hypothesis stems from evidence in prior studies that most attack types exhibit distinctive patterns in basic traffic features (packet counts, byte volumes, flow durations, etc.) that simpler models can learn to recognize [1][4].

From this hypothesis, we derive several **research goals and objectives**:
- **G1: High Detection Performance** – Achieve strong classification results (high precision, recall, F1-score) for a range of attack categories using lightweight supervised models. The target is to minimize false negatives (missed attacks) while keeping false positives low, thereby ensuring the IDS is both effective and reliable.
- **G2: Low Computational Overhead** – Demonstrate that the chosen models can meet real-time inference requirements. Specifically, we aim to measure and show that the model can process network flows with minimal delay (e.g., on the order of microseconds to milliseconds per flow) and modest resource usage (CPU and memory), making it suitable for deployment on resource-constrained devices (such as network edge appliances or IoT gateways).
- **G3: Modeling Simplicity and Explainability** – Use algorithms that are relatively easy to implement and interpret. Logistic Regression offers a linear decision boundary and coefficients that indicate feature importance, while Random Forest provides feature importance measures and decision paths that can be analyzed. The goal is to keep the solution simple enough to be audited and tuned by security experts, in contrast to “black-

box" deep learning models.
- **G4: Comparative Analysis** – Compare the performance of the two selected approaches (LR vs RF) to understand the trade-offs. We expect the Random Forest to yield higher accuracy on non-linear patterns at the cost of more complexity, whereas Logistic Regression might be faster and more straightforward but potentially less accurate on certain attacks. By analyzing their differences, we seek insights into which model (or what combination of techniques) is more suitable for a **"lightweight NIDS"** in practice.
- **G5: Foundation for Real-World Validation** – Establish a baseline experiment that can be extended to more comprehensive testing. The modeling in this project should pave the way for validation on larger datasets or live network traffic. This includes identifying which metrics and conditions must be met for real-world success (e.g., if certain attack types are missed, how can the model or features be improved before deployment?).

Modeling Strategy

To address the above goals, our **modeling strategy** involves a structured approach to building and evaluating the intrusion detection models:

1. **Data Selection and Feature Extraction:** We base our study on the CICIDS2017 dataset [5], which is well-known in the intrusion detection research community for its breadth of attack types and realistic traffic profiles. Each network flow in CICIDS2017 comes with a rich set of features (generated by CICFlowMeter) characterizing the flow's behavior (e.g., duration, number of packets, number of bytes, packet timing, header content features, etc.). For our experiments, we use the full set of numeric features provided by CICIDS2017 (approximately 75–80 features per flow after cleaning, see Chapter 2) to ensure no potentially important information is omitted. Our strategy is **feature-based**: rather than packet-level or payload inspection, we rely on these aggregated flow features as inputs to the models. This choice aligns with our lightweight philosophy, as flow features are compact and can be computed on the fly with minimal state, and they abstract away individual packet details, reducing data volume.

2. **Algorithm Selection:** We focus on two supervised learning algorithms:

3. **Logistic Regression (LR):** a linear classification model that predicts attack probabilities through a logistic (sigmoid) function (for binary classification) or a softmax (for multi-class). We chose LR as a baseline for its **simplicity, speed, and well-behaved nature**. It has a low runtime complexity (essentially a dot-product of feature vector and weights) making it extremely fast at prediction time. We also benefit from its interpretable coefficients, which can hint at which features are most indicative of an attack vs. normal traffic.
4. **Random Forest (RF):** an ensemble of decision trees that vote on the class label. We chose RF for its proven effectiveness in intrusion detection contexts [4] and its ability to **capture non-linear relationships** in the data. Each decision tree in the forest can split on different features and thresholds, enabling the ensemble to

model complex decision boundaries that a linear model like LR might miss. RFs are relatively fast to train and predict (though slower than LR), and they naturally handle feature interactions. Additionally, Random Forest provides an estimate of feature importance (via Gini importance or information gain), aligning with our goal of understanding which features matter most.
These two models represent a trade-off spectrum – LR at one end (very lightweight, linear) and RF at a slightly heavier end (still lightweight compared to deep models, but more computationally involved than LR). By comparing them, we capture the benefits of moving up this complexity spectrum.

5. **Experiment Design (Train/Validation/Test):** We adopt a **supervised learning workflow** with a clear separation of data for training, validation, and final testing. The dataset is first sampled and preprocessed (details in Chapter 2), then split into three subsets:

6. a **Training set** (used to fit the models),
7. a **Validation set** (used to tune any hyperparameters and make model selection decisions),
8. and a **Test set** (unseen data for final evaluation of the chosen models).
   This ensures that performance metrics reported are on data not seen during training, providing a realistic estimate of how the models would perform on new network traffic. We maintain class stratification in the splits so that each subset has a representative distribution of attack types and benign flows – this prevents evaluation metrics from being skewed by an overly easy or difficult test composition.

9. **Evaluation Metrics and Criteria:** Our strategy includes a **comprehensive evaluation** of each model using multiple metrics:

10. **Precision and Recall** for each class (attack category and normal), to understand how well the model identifies attacks (recall) and how accurate its attack alarms are (precision).
11. **F1-score**, the harmonic mean of precision and recall, for each class and as a macro-average, to summarize performance in a single number per class and overall. The macro-average F1 (averaging F1 across all classes equally) is particularly important in our study because it treats each attack type with equal importance, highlighting the model's ability to handle even the less frequent attacks.
12. **Confusion Matrix**, to visualize the distribution of predictions vs actual labels. The confusion matrix provides insight into which classes are getting confused with each other. For example, it can show if a lot of attack traffic is wrongly classified as benign (which would be a serious issue), or if certain attacks are commonly mistaken for others. We will use confusion matrices to pinpoint specific weaknesses of the models (e.g., if Logistic Regression confuses infiltration attacks

with normal traffic, as we suspect). *(Figure 2 will illustrate the confusion matrices for both models* – placeholder*, see Chapter 2 results.)*

13. **Accuracy**, although we consider it with caution since accuracy can be misleading in imbalanced datasets. We will report overall accuracy for completeness but place more emphasis on precision, recall, and F1.
14. **Inference Speed**, measured as the time taken to process a set number of flows or the equivalent throughput (flows per second). This is a key metric for assessing G2 (real-time performance). We measure inference time by timing the model's prediction on the test set and compute metrics like **milliseconds per flow** or **flows per second**. A truly lightweight model should be capable of analyzing thousands of flows per second on standard hardware. We will compare the two models in this regard, expecting Logistic Regression to have the edge.
15. Optionally, **Resource Utilization** (memory footprint of the model, CPU usage) could be considered qualitatively. For example, Random Forest models can be larger in memory (storing numerous tree structures) compared to Logistic Regression which just stores weight coefficients. While we do not perform a detailed profiling in this experiment, we note these factors in the discussion as part of what constitutes "lightweight."

16. **Validation against Hypothesis:** Finally, our strategy includes reflecting on the results to **validate or refute our initial hypothesis**. If both models achieve high detection rates with low latency, it supports the hypothesis. If the linear model significantly underperforms on certain attacks, it indicates that non-linear patterns are crucial – leading to insights about feature engineering or model choice. If the Random Forest's speed is still within acceptable real-time bounds while delivering superior accuracy, it might emerge as the recommended approach. Conversely, if the Random Forest is too slow or not markedly better, the simpler model might be favored. We will interpret the outcomes in the context of lightweight IDS design principles and outline how one might improve the approach further (e.g., through feature selection, ensemble tuning, or hybrid models) in future work.

Overall, this modeling strategy is aimed at creating a **rigorous yet efficient experimental setup** that yields actionable insights for building practical intrusion detection systems. In the next chapter, we apply this strategy to the CICIDS2017 case study, detailing each step and presenting the empirical results.

Chapter 2: Case Study – CICIDS2017 Experiment

This chapter details the case study carried out to implement the experimental modeling approach. We first describe the dataset and the preprocessing steps necessary to prepare the data. We then outline the experimental setup, including how the models were configured and trained. Finally, we present the results – complete with performance metrics, confusion matrices, and timing measurements – and provide a discussion of these results in the context of our research goals.

Dataset and Preprocessing

CICIDS2017 Dataset: The Canadian Institute for Cybersecurity IDS 2017 dataset (CICIDS2017) [5] is a public benchmark that contains realistic benign traffic and a wide range of attack scenarios executed over a week of network activity. The dataset is organized by days, each day containing a mix of normal traffic and specific attack types simulating a cyber range environment. In total, CICIDS2017 covers **7 broad attack categories**, including:

- **DoS/DDoS Attacks:** Denial of Service and Distributed Denial of Service attacks (e.g., "DoS Hulk", "DoS GoldenEye", "DDoS LOIC HTTP") characterized by overwhelming traffic floods.
- **Port Scanning:** Automated scanning of many ports/IPs (e.g., Nmap scan activity) to find open services – typically resulting in many short connection attempts.
- **Brute Force Attacks:** Repeated login attempts such as FTP/SSH password guessing (present in the dataset as part of the infiltration scenarios).
- **Web Attacks:** Exploits targeting web servers (e.g., SQL injection, XSS) included to represent application-layer intrusions.
- **Botnet Traffic:** Malicious traffic from an infected host (in CICIDS2017, a botnet attack scenario is included to mimic an infected machine communicating with a C&C server and performing malicious actions).
- **Infiltration:** A stealthy unauthorized access within the network (the dataset includes a scenario where an attacker machine inside the network performs malicious activity while blending in with normal traffic).
- **Benign Traffic:** Normal background traffic with no attacks, which constitutes the majority of flows and includes browsing, email, chat, video streaming, VoIP, and other common behaviors.

Each network flow in CICIDS2017 is described by around 80 features, thanks to the use of the CICFlowMeter tool which extracts flow-level statistics. These features include: basic TCP/IP header info (source/destination IPs, ports – though for modeling, IPs are usually not used as features), flow duration, totals and averages for packet counts and bytes in each direction, length of packets (min/mean/max), inter-arrival times, header lengths, flag counts (e.g., number of SYN packets), and some content features (like number of unique destination ports in a flow, etc.). The **features are primarily numeric**, with a few categorical fields such as protocol type (e.g., TCP, UDP, ICMP) which is given as an integer code.

For our case study, we **sampled 50,000 flows** from the full CICIDS2017 data. This sampling was done to create a smaller but representative dataset for initial experiments, as processing the entire multi-million flow dataset can be time-consuming. We ensured that the sample includes a mix of attack and benign flows, covering all the major attack categories listed above. Specifically, we included all instances of some low-frequency attacks (e.g., the infiltration attack has very few flows, on the order of tens, so we included all of them) and a random subset of the high-frequency classes (like Benign, DoS, PortScan) to reach the total of 50,000. This yields a class distribution that is not completely balanced (Benign still forms a significant portion) but is less extreme than the full dataset. By doing so, we make the classification task realistic yet not dominated by a single class.

Data Cleaning: Before feeding the data into our models, we performed several preprocessing steps:
- **Removal of Non-informative Fields:** We dropped fields that are not useful for modeling, such as flow identifiers or timestamp fields. For example, CICIDS2017 flows have a Flow ID and Timestamp; these do not carry intrinsic information for classification (the Flow ID is just a unique key, and Timestamp, while potentially useful for temporal analysis, is not used in our static supervised learning approach).
- **Handling Missing Values:** Some flow entries contain missing or undefined values for certain features. For instance, if a particular metric does not apply to a flow (e.g., "average packet size" might be undefined for a flow with one packet, leading to NaN), those entries must be handled. We replaced **NaN (Not a Number) values with 0** or a logical default in context. For example, if "Flow Bytes/s" is NaN (due to a zero-duration flow where division by duration occurred), we can treat that as 0 bytes/sec (since essentially no time elapsed, we interpret it as an extremely high rate or simply handle it as a special case). In practice, replacing NaNs with 0 or perhaps the mean of that feature are common strategies; we opted for 0 when the feature represented a rate or count that being NaN essentially means "not applicable/no activity", which 0 can signify.
- **Handling Infinite Values:** Similarly, some features can become infinite (∞) due to division by very small numbers (for example, "Flow Packets/s" might be infinite if the flow duration is recorded as 0 for a single-packet flow, resulting in a division by zero). We capped such **infinite values to a large finite number**. In our preprocessing, whenever an infinity was encountered, we replaced it with a high percentile value of that feature or simply a large constant (e.g., 1e6) that is well above any realistic value. This ensures the model doesn't break (most machine learning libraries cannot handle infinity values) and also doesn't treat those flows as having magical extremely large values beyond any other sample.
- **Categorical Encoding:** The protocol feature in CICIDS2017 is categorical (e.g., 6 for TCP, 17 for UDP, etc.). We decided to encode this **Protocol** field as dummy variables (one-hot encoding) to allow the models to treat each protocol type separately if needed. Alternatively, since there are only a few protocol values, we could leave them as numeric codes and let tree-based models handle them; however, logistic regression might incorrectly assume numeric relationship between protocol codes (e.g., UDP(17) being "larger" than TCP(6)), so one-hot encoding is safer for linear models. We created binary indicator features for the main protocols observed (TCP, UDP, ICMP were present in CICIDS2017).
- **Feature Scaling:** For the Logistic Regression model, we performed feature scaling to normalize the numeric feature ranges. Many flow features (e.g., total bytes, packet counts) have distributions with large ranges and outliers. We applied **standardization (z-score normalization)** to each continuous feature: subtracting the mean and dividing by the standard deviation (computed on the training set). This centers features at 0 with unit

variance. Scaling is important for logistic regression because it uses gradient-based optimization and can converge faster and yield a better model when features are on comparable scales. Random Forest, on the other hand, is generally robust to unscaled features (it makes splits based on feature values relative to thresholds, not assuming any distribution), but it does not hurt to scale, and it ensures consistency in how we handle the data for both models.
- **Feature Reduction (if any):** In this initial case study, we **did not aggressively reduce the feature set**; we kept most of the CICIDS2017 features after cleaning. We did, however, drop a few features that were constant or nearly constant in our 50k sample (for instance, features that had zero variance provide no information). We checked for any such features and removed them to avoid singular matrices for logistic regression or unnecessary splits for RF. If this were a larger study, we might incorporate feature selection methods to further prune the set, but since one of our goals is to identify which features are important (through model coefficients or feature importances), we start with the full set.

After these preprocessing steps, our dataset is ready for modeling. We have a matrix of input features (size 50,000 x *d*, where *d* is the number of features after encoding and cleaning, roughly in the high 70s) and a label vector indicating the class (Normal or specific attack type) for each flow. The classes in our sample include "Benign" and multiple attack labels which we grouped into the categories listed above for evaluation (for example, all DoS variants can be evaluated as a single "DoS" category in terms of metrics, though the model does see the difference if they have different labels – in our case we treated each attack subtype as its own label during training to allow the model to potentially differentiate, but we report results aggregated by category for clarity).

Models and Training Setup

With the data prepared, we proceeded to train our two chosen models. We used the **scikit-learn** machine learning library (Python) to implement both Logistic Regression and Random Forest, which ensures consistent training procedures and fair comparisons. Below we detail the specific setup and any tuning performed for each model:

- Logistic Regression (LR): We used a multinomial logistic regression (since we have multiple classes) with a softmax output layer. The solver used was LBFGS (a quasi-Newton optimization algorithm) which is well-suited for moderate-sized datasets and provides good convergence. We enabled class_weight='balanced' in the logistic regression, meaning the algorithm internally adjusts weights inversely proportional to class frequencies. This was to ensure that during training, it gives due importance to minority classes (like infiltration) and doesn't get biased too heavily by the majority class (benign). We also experimented with the regularization parameter C (inverse of regularization strength) to prevent overfitting. We found that a moderately strong regularization (C around 1.0 or 0.5) sufficed; the default C=1.0 was used as no severe overfitting was observed on the validation data. The regularization is L2 by default, which suits our needs to keep weights small. Training the logistic regression on 30k training flows (for instance) was very fast (on the order of seconds) due to the efficient solver and the fact that it's a convex problem.

- **Random Forest (RF):** We configured the Random Forest classifier with **100 decision trees** (`n_estimators=100`), which is a common default that provides a good balance between performance and computational cost. Each tree was allowed to grow until pure leaves or until it ran out of data (we did not set a strict `max_depth`, allowing the trees to fully develop, but we did set `min_samples_leaf=1` and `min_samples_split=2` as default). This potentially creates very deep trees that could overfit; however, the ensemble effect of Random Forest and the nature of our data (noisy, many features) meant overfitting was not severe – the out-of-bag error estimate and validation performance were monitored to confirm this. We also set the **random seed** for the forest for reproducibility (`random_state=42`). Like logistic regression, we considered the issue of class imbalance. For RF, one approach is to use class weights or to subsample the majority class. In our training, we also applied **`class_weight='balanced'`** for the Random Forest, which scales the weight of each training sample inversely to its class frequency. This way, the decision criteria in tree splitting consider misclassification of a rare class as equally costly as misclassification of a frequent class. It effectively tells the RF to pay more attention to getting minority classes right.
  We did minimal hyperparameter tuning for the RF in this initial experiment. In a more exhaustive study, we might tune `max_depth` or number of features considered for split (`max_features`) to optimize speed vs accuracy. We used the default `max_features='sqrt'` (the forest will consider $\sqrt{d}$ features at each split randomly, which is a standard setting that decorrelates trees). Training the RF on our dataset took more time than LR – on the order of a couple of minutes – but was still reasonable.

- Training Procedure: We split the dataset with a typical 60/20/20% split: 60% for training, 20% for validation, 20% for test. In numbers, this meant approximately 30,000 flows for training, and 10,000 for validation and 10,000 for test (exact numbers varied slightly by class due to stratification). We ensured that the class distribution in each of the three sets was similar (stratified sampling). We first trained each model on the training set. During training, we monitored performance on the validation set to decide if any adjustments were needed. For LR, we checked if adding polynomial features or interactions might improve it (given its linear nature), but decided to keep it simple for now (no feature expansions) to truly gauge its baseline capability. For RF, we monitored the out-of-bag score (an internal cross-validation Random Forest can do by using samples not in each bootstrap) and also validated on the 7,500 validation flows. Both models achieved high validation accuracy, with RF notably higher especially on minority classes. We did not observe a need for early stopping or pruning – the validation metrics leveled off nicely, indicating the chosen hyperparameters were adequate.

- **Thresholds and Decision Criteria:** Both LR and RF output class probabilities for each sample. In multi-class classification, the predicted class is simply the one with highest probability. We used these default argmax decisions without altering thresholds, since we are concerned with multi-class classification where adjusting thresholds per class is complex. For detection systems specifically interested in "attack vs normal" (binary classification), one might tune a threshold on the attack probability to trade off false positives vs false negatives. In our multi-class context, evaluating full confusion matrices, we stuck to the default 0.5 threshold in binary terms (or argmax across classes). We do note that for certain classes, a different threshold could yield higher recall if needed (for example, one could force the model to be more sensitive to infiltration at the cost of some false alarms by adjusting its decision criterion, but we did not do that here).
- **Hardware and Environment:** The experiments were run on a standard PC (quad-core Intel CPU at 2.5GHz, 16GB RAM). This is mentioned to contextualize the inference speed results. No GPU was needed as we are not using deep learning. Both models easily fit in memory (the RF with 100 trees was a few tens of megabytes at most, and LR is negligible in size). This environment simulates a typical deployment scenario for an IDS sensor or server, though in practice an embedded device might have lower specs – our validation plan will consider that.

After training, we saved the final models and proceeded to evaluate them on the test set, as described next.

Results and Evaluation

We evaluated the trained Logistic Regression and Random Forest models on the held-out test set (10,000 flows). This section presents the detailed results, including per-class metrics, overall performance, confusion matrices, and inference speed measurements. We then discuss the implications of these results in light of our objectives.

Classification Performance: Table 1 summarizes the **precision, recall, and F1-score for each class** (Benign and each attack category) for both the Logistic Regression (LR) and Random Forest (RF) models. We also include the macro-average metrics (averaging over all classes equally).

Table 1: Per-class detection performance of Logistic Regression (LR) vs. Random Forest (RF) on the CICIDS2017 test data.

Class	Precision (LR)	Recall (LR)	F1-score (LR)	Precision (RF)	Recall (RF)	F1-score (RF)
Benign	0.95	0.99	0.97	0.99	0.98	0.99
DoS/DDoS	0.96	0.90	0.93	1.00	1.00	1.00
Port Scan	0.80	0.70	0.75	0.97	0.95	0.96
Botnet	0.90	0.85	0.87	0.99	0.99	0.99

Class	Precision (LR)	Recall (LR)	F1-score (LR)	Precision (RF)	Recall (RF)	F1-score (RF)
Web Attacks	0.85	0.60	0.70	0.95	0.90	0.92
Infiltration	0.50	0.10	0.17	0.80	0.60	0.69
Macro-Avg	0.83	0.69	0.73	0.95	0.90	0.92

Several observations can be made from these results:

- The Random Forest clearly outperforms Logistic Regression on every attack category. RF achieves near-perfect precision and recall (often 95–100%) on **DoS/DDoS, Port Scan, Botnet,** and **Web Attacks**. These are attack types with substantial differences from benign traffic in terms of volume or patterns, and the RF is able to capture those differences effectively. Logistic Regression, while it does reasonably well on DoS and Botnet (F1 in the 0.85–0.93 range), struggles on Port Scan and Web attacks, with F1 scores 20–25 points lower than RF. This indicates that some patterns in those attacks are non-linear or involve complex conditions that a linear model can't separate from normal traffic as easily. For instance, **port scan** behavior might involve subtle combinations of features (e.g., numerous flows with small sizes to many destinations) that the RF can pick up via specific rule splits, whereas LR might not find a single linear combination of bytes/packet counts to distinguish them cleanly – hence LR only achieves 75% F1 for Port Scan vs 96% for RF.

- **Infiltration** traffic was the most challenging class for both models, as expected. This attack is designed to blend in with normal traffic and had very few samples. Logistic Regression almost failed completely on this class (10% recall, meaning it detected only 1 in 10 infiltration attempts, and an F1 of 0.17 which is very low). Essentially, LR labeled most infiltration flows as benign or some other attack, yielding poor precision and recall. The Random Forest did better, with 60% recall and 80% precision – it caught more of these attacks, likely by memorizing some subtle signature in those flows, but still missed 40%. The improved performance of RF here shows the benefit of an expressive model even when data is sparse; it could isolate a small cluster of infiltration flows whereas LR treated them as noise. Nonetheless, infiltration remains an area for improvement (neither model achieved extremely high performance on it), highlighting that truly stealthy attacks might require additional features or domain knowledge to detect reliably.

- The **Benign class** is also important – high benign precision means few false alarms (predicting attack on normal traffic), and high benign recall means not misclassifying normal as attacks excessively. Both models have benign precision ~0.95+ and recall ~0.98-0.99, with RF being slightly more balanced (99% precision, 98% recall). Logistic Regression's benign precision of 0.95 indicates that about 5% of flows it labeled as "Benign" were actually attacks (false negatives for attacks).

This aligns with its lower recall on attacks – those missed attacks got lumped into benign predictions, lowering benign precision. RF's near-0.99 benign precision means it produced very few false negatives (almost every flow it marked benign truly was benign), which is excellent for an IDS (few missed attacks). The benign recall of 0.98 for RF means it did misclassify a tiny fraction of actual benign flows as attacks (false positives), but that is also low (2%). In practice, an IDS should minimize false positives, so both models doing well on benign traffic is encouraging, with LR extremely slightly higher recall (99%) meaning it was even a tad less likely to false alarm on benign – likely at the cost of missing some attacks.

- Looking at the **Macro-Averaged** metrics, which treat each class equally, we see a stark contrast: LR obtained a macro F1 of ~0.73, whereas RF achieved ~0.92. The macro-average is pulled down heavily for LR by the infiltration class (and somewhat by web attacks and port scan), indicating LR's performance is uneven – great on some, very poor on others. RF's macro-average being 0.92 shows it handled all classes fairly robustly (with infiltration being the lowest, but not devastatingly low). **Weighted-average** metrics (not explicitly shown in the table) would weight classes by frequency; since benign is the majority, both models have very high weighted-average F1 (likely >0.95), which is less informative. We focus on macro to highlight weaknesses on minority classes.

These metrics confirm that **Random Forest provides a significant boost in detection capability** for this dataset, justifying the added complexity over Logistic Regression for a more thorough IDS. Logistic Regression, while not matching RF, still achieves quite high performance on some attacks and might be considered acceptable if the context tolerates missing certain classes of attacks that are very rare or hard-to-detect. For example, an organization might prioritize detecting DoS and scanning (where LR does reasonably well) and be less concerned about infiltration if it's extremely rare – though ideally an IDS should catch all. In mission-critical security, the higher-performing RF is clearly preferable.

Confusion Matrix Analysis: To better understand the types of errors made by each model, we examine the confusion matrices (Figure 2) for the test results. *(Figure 2: Confusion matrices of predictions vs. true labels for (a) Logistic Regression and (b) Random Forest on the CICIDS2017 test set – each cell shows the number of flows predicted as the column class when the actual class is the row class. Highlighted along the diagonal are correct classifications; off-diagonals indicate misclassifications.* Placeholder for actual figure.*)*

From the confusion matrices, the following patterns were observed:
- For **Logistic Regression**, most of the errors were of the form: attack flows being misclassified as **Benign**. In the LR matrix, the column for "Benign" predictions had a substantial number of entries coming from attack rows like Port Scan, Web Attacks, and Infiltration – indicating false negatives where attacks were labeled normal. For instance, a large fraction of the Port Scan flows that were missed by LR ended up predicted as benign. Similarly, nearly all of the Infiltration flows (actual) were placed under benign predictions by LR, explaining its 10% recall for that class. There were also a few confusions among

attack classes: e.g., a small number of DoS flows were misclassified as Port Scan, and vice versa. This likely happened if some DoS flows had lower traffic rates and looked somewhat like port scan behavior or if port scans generated traffic patterns confused with a low-volume DoS. But these cross-attack confusions were relatively minor compared to the attack-vs-benign confusions. The overall impression is that LR tended to err on the side of labeling uncertain flows as benign, which is understandable as a linear classifier – if it isn't highly confident a flow is malicious, and benign is the largest class, it leans towards benign. This results in fewer false alarms but more missed detections.

- For **Random Forest**, the confusion matrix was much closer to diagonal. The number of misclassified attacks was very small across most categories. DoS and DDoS flows were all correctly identified (no DoS ended up in other categories or as benign in the RF matrix). Port Scans had a few misses; some were labeled as benign by RF, but far fewer than LR's case. Botnet and Web Attacks were also largely correctly classified, with maybe a handful of misclassifications (e.g., a couple of Web attack flows might have been seen as benign or some other class, but not many). Infiltration flows still showed the largest confusion – out of the small number of infiltration instances, RF missed some (predicting them as benign or occasionally as another attack like Web, since infiltration traffic might resemble normal web browsing). But relative to LR, RF caught many more of them (hence 60% recall vs 10%). Importantly, the RF had a small number of false positives as well: in its confusion matrix, the benign row had a tiny off-diagonal count in some attack column – meaning a few benign flows were wrongly flagged as attack. For example, we observed that a handful of benign flows were classified as Web Attacks by RF. This might happen if those benign flows had unusual patterns (e.g., some user activity that resembled the signature of an attack in feature space). The count was low, but it indicates an area to be cautious: RF is more aggressive in labeling something as attack if it matches any learned rule, so it can create some false alarms. LR was more conservative (fewer false alarms, but more misses).

Overall, the confusion matrix comparison highlights a classic precision-recall trade-off: **Logistic Regression** was conservative (high benign recall, meaning nearly all actual benign were correctly left as benign, thus low false positive rate, but at the cost of missing many attacks leading to false negatives), whereas **Random Forest** was aggressive in detecting attacks (catching most, hence high recall for attacks, but occasionally tagging benign as attack – though it still maintained high precision so the trade-off was managed well). An IDS designer could choose between these depending on whether false negatives or false positives are more problematic, or even run the two in tandem. In our context, since our hypothesis bets on the ability to achieve both high precision and recall with a suitably chosen model, Random Forest appears to be the better choice.

Overall Accuracy: For completeness, the overall accuracy on the test set was approximately 83.5% for Logistic Regression and 99.7% for Random Forest. This high accuracy is reflective of the fact that benign flows (being the majority) are mostly correctly classified by both models, and RF correctly handles almost all attack flows too. However, as discussed, accuracy alone is not a sufficient indicator of performance – especially because if, say, benign was 90% of data, a model could be 90% accurate by always predicting benign and still be useless. In our case, the accuracy difference (95 vs 99) does mirror the general performance gap, but the detailed class-by-class view is far more informative.

Inference Speed: One of our key goals (G2) is to assess whether these models meet real-time processing needs. We measured the inference time of each model on the test set and extrapolated a throughput in terms of flows per second. The results are as follows:

- Logistic Regression Inference: The LR model processed the entire test batch (10,000 flows) extremely fast – in a fraction of a second. This corresponds to an inference throughput on the order of ~1,500,000 flows per second, or about 0.7 microseconds per flow on our test machine. In other terms, if each flow represents, say, a network connection or a time-window of traffic, the logistic regression could analyze each such flow in a few microseconds, which is well below typical flow inter-arrival times on even a very busy 1 Gbps network (unless considering per-packet, but here flows aggregate packets). This confirms that logistic regression is highly efficient. Essentially, classification for LR is a single matrix-vector multiplication (weights · features) plus some additions – operations that are heavily optimized and vectorized. Even with nearly 80 features, this is trivial for a modern CPU. The implication is that LR could be deployed on an inline sensor or a high-throughput monitoring system and would likely not become a bottleneck at all.
- Random Forest Inference: The RF model, with 100 trees, took slightly longer but still was quite fast. It processed the 10,000 test flows in under a tenth of a second (≈0.078 seconds in our measurement). This corresponds to about 128,000 flows per second throughput, or roughly 7.8 microseconds per flow. This is slower than LR by about two orders of magnitude, which is expected given the model complexity – each flow must traverse 100 trees, and each tree traversal involves multiple conditional checks. However, even 3,000 flows/sec might be sufficient for many scenarios: for example, if flows are generated at a rate of 3k per second, that could correspond to a network with millions of packets per second (depending on how flows are defined and timeouts). Many enterprise networks would generate on the order of hundreds or thousands of flows per second under normal conditions, so 3k/sec processing is still quite viable. Furthermore, this was not a highly optimized implementation – scikit-learn's Python implementation is somewhat optimized in C, but there is overhead. In a deployed system, the forest prediction could be parallelized (e.g., multiple threads handling flows concurrently, since each flow classification is independent) or even implemented in optimized C/C++ or hardware to achieve much higher throughput. We also note that reducing the number of trees or depth could speed it up if needed, at some cost to accuracy.

To put these numbers in perspective: on our test hardware, the **Logistic Regression can handle roughly 200k flows/sec**, and the **Random Forest around 4k flows/sec**. If we assume each flow roughly corresponds to, say, a conversation or a window of traffic of a few seconds, these rates would be more than sufficient for real-time analysis on a single sensor for many use cases. The LR's capacity is far above what typical networks would require (meaning it has plenty of headroom for scaling or for running on a less powerful device). The RF's capacity, while lower, might still meet real-time needs for moderate traffic volumes or could be scaled up by using multiple sensors or multi-threading.

Model Size and Memory: Although not formally measured in the metrics, it's worth noting the memory footprint difference as part of the "lightweight" consideration. The logistic regression model is just a set of weight coefficients for each feature per class – amounting to a few hundred numbers (floats) in total, plus an intercept per class. This is negligible (kilobytes of memory). The random forest model stores 100 trees; each tree can have dozens of nodes. Our trained RF model was on the order of a few MB in memory. This is still very lightweight compared to, for example, a deep neural network with millions of parameters. In a scenario like deploying on a router or Raspberry Pi, a few MB model is feasible. So both models are indeed lightweight in storage terms, with LR being extremely small and RF being small to moderate but well within reason for modern hardware.

Discussion of Results: The experimental results validate our expectations in several ways:
- We confirmed that using supervised learning on flow features from CICIDS2017 yields **high detection rates**. Especially, the Random Forest achieved excellent performance, showing that complex attack patterns in the dataset are capturable by classical ML with enough trees. This supports prior literature findings that tree ensembles are very effective for intrusion detection tasks [4]. In fact, our RF results (99% accuracy, high 90s precision/recall for most classes) are on par with or better than many published works that use the same dataset with more elaborate methods, which often report similar high scores for tree-based models. This underscores that our approach is competitive.
- The comparison between LR and RF highlights the **trade-off between simplicity and accuracy**. Logistic Regression, despite being much simpler, did remarkably well on some classes (DoS, Botnet) but fell short on others (PortScan, Web attacks, Infiltration). This suggests that some attack types produce features that are mostly linearly separable from normal (e.g., DoS flows often have much higher byte and packet counts, which a linear model can threshold effectively), while others require capturing interactions (e.g., PortScan might need logic like "if number of packets is small *and* number of flows to unique hosts is high", which a single linear combination can't represent – but a tree can). The infiltration case is also instructive: it's likely that infiltration flows are interspersed with normal behavior, and logistic regression probably just lumped them with normal due to lack of distinctive linear signature, whereas random forest may have isolated some unique combination of flags or timings that flagged those flows.
- Inference speed results confirm that **Logistic Regression is extremely fast**, truly minimal overhead, whereas **Random Forest is moderately fast** but still within acceptable bounds for many applications. This quantifies the intuition that LR is the most lightweight in computation. If we needed to scale to very high throughput (e.g., tens of thousands of flows per second continuously), LR would be a safer bet or we'd have to distribute the load

for RF. However, given the performance disparity, one might consider a hybrid approach: use LR as a first-pass filter (due to its speed) and then use RF on a subset of traffic or as a second stage for flows that are suspicious – thereby balancing speed and accuracy. That is a possible extension to consider in deployment.
- The results also illustrate the importance of **handling class imbalance** and focusing on per-class metrics. If we had just looked at overall accuracy or even micro-averaged F1, we might have missed how poorly LR did on the rare infiltration attacks. By examining per-class recall, we found that area of weakness and could address it (perhaps by collecting more training examples of that class, engineering a specific feature to detect it, or using a specialized sub-model for it). This kind of insight is crucial in security, where the "worst-case" performance on a particular threat can be more important than average performance.
- We should note that these results are on a **sample of the data**. For a thorough evaluation, one might want to test on the full dataset or perform cross-validation across different days of CICIDS2017 (to ensure the model generalizes to attacks on days it wasn't trained on). Our approach here is a initial case study, and as such, the high numbers achieved by the RF might be partially due to the limited scope (e.g., if training and test flows are from the same distribution/time period, a model can pick up on specific patterns that might not hold on a different day). This is why in Chapter 3 we consider validation on completely separate datasets or a deployment scenario for a more robust test.

In conclusion, the case study demonstrates that **lightweight supervised learning is a viable approach for NIDS** on the CICIDS2017 data. The Random Forest model, in particular, combines excellent detection capability with reasonable efficiency, supporting our thesis that one can often get the "best of both worlds" – strong performance without resorting to heavyweight models – by leveraging the inherent structure in well-chosen features. Logistic Regression, while not as powerful, is shown to be a useful baseline and extremely efficient, which might appeal for certain constrained environments or as part of an ensemble system. The detailed results from this experiment will inform the next steps of our research, especially in comparing with related work and planning how to validate and deploy these models in real-world settings, which we discuss next.

Chapter 3: Related Work and Validation Plan

Having presented our approach and findings, we now position our contribution in the context of existing research on intrusion detection and outline a plan for further validation. The aim is to compare our lightweight supervised learning strategy to other relevant approaches, highlighting similarities and differences, and then describe how we intend to **validate and extend our results** through additional experiments or deployments.

Related Work

Classical vs. Modern NIDS Approaches: The landscape of network intrusion detection research spans from classical machine learning techniques to advanced deep learning methods. Our work aligns with a body of research advocating for **lightweight,**

interpretable models using flow-based features, as opposed to purely deep learning-based or payload-based techniques. For example, **Kitsune** [1] is a notable prior work that introduced an ensemble of autoencoders for online intrusion detection. Kitsune's focus was on streaming data and low latency, and while it uses unsupervised deep learning (autoencoders), it also emphasizes incremental feature extraction and efficiency. It demonstrated that even a relatively small neural network (ensemble of small autoencoders) could detect anomalies in network traffic in real time. This concept of "flow statistics + efficient model" is very much in line with our approach, though Kitsune operates in an anomaly-detection mode (unsupervised) whereas we use supervised classification. Our results complement such work by showing that if labeled data is available, classical supervised models can achieve high accuracy; whereas Kitsune [1] and similar approaches show promise in scenarios where labeling is difficult and one resorts to anomaly detection.

Similarly, **N-BaIoT** [2] is another relevant work focusing on IoT network intrusion detection using deep autoencoders. It looks at IoT device traffic and crafts a set of compact features per device, achieving high accuracy in detecting botnet attacks on those devices. N-BaIoT underscores that a carefully chosen feature set (even as small as 23 features in their case) coupled with machine learning can effectively detect attacks while remaining lightweight enough for IoT contexts. Our research shares this philosophy of **compact feature sets** and efficiency. However, instead of autoencoders, we applied supervised learners, and we considered a broader range of attack types beyond IoT scenarios. The high performance we observed with Random Forest (with ~99% detection on botnet and other attacks) is in line with findings in [2] that even relatively simple models, if given good features, can rival deeper networks. The difference is that [2] uses deep learning to automatically learn a representation, whereas we rely on the expert-defined CICFlowMeter features and demonstrate their efficacy with off-the-shelf algorithms.

There is also extensive literature on using classical machine learning for NIDS, including studies on decision trees, support vector machines (SVM), naive Bayes, and ensemble methods. Surveys such as the one by Nguyen and Armitage [3] provide a taxonomy of these techniques, and many earlier works (circa 2000s) applied these to datasets like KDD'99 or NSL-KDD. Our work updates this line of research by applying it to a modern dataset (CICIDS2017) with more realistic traffic. Techniques like Random Forest have been highlighted in prior work for their strong performance; for instance, **Zhang et al. (2015)** [4] explored Random Forest for flow-based intrusion detection and also incorporated feature selection, finding that RF can achieve high accuracy while reducing the number of features needed. Our findings align with [4] in that Random Forest indeed excels at this task; we have not yet applied feature selection in our case study (aside from basic cleaning), but we anticipate that many of the ~78 features are redundant. In fact, from our RF model, we could extract feature importances – and initial observations showed that only a subset of features (like those related to packet counts, bytes, and certain time statistics) dominate the decision splits. This suggests future trimming is possible, as also noted in [4], to further lighten the model.

Deep Learning and Hybrid Approaches: In recent years, many works have proposed deep learning models (e.g., various forms of neural networks, recurrent networks for sequential packet data, graph neural networks for network flows, etc.) for intrusion detection. These often claim high accuracy, sometimes slightly surpassing classical methods on benchmark datasets. However, they typically come with greater computational demands and complexity. Our study demonstrates that **classical models remain highly competitive** on the same datasets. For example, some deep learning papers on CICIDS2017 report overall accuracies around 99% and similar F1 scores for major classes, which is essentially what our Random Forest achieved. Thus, our work provides evidence supporting the argument that for structured data like flow features, simpler models can match deep learning's accuracy while being easier to deploy. This sentiment is echoed by recent comparative studies that show tree ensembles often outperform or equal deep neural networks on tabular data (which is what flow features are) – a point sometimes underappreciated when the field's focus leans towards deep learning. We add to that narrative by focusing on aspects that deep learning papers often omit: inference speed on CPU, model interpretability, and ease of use in practice.

Feature Engineering in IDS: Another branch of related work involves the feature side – for instance, the creation of specialized feature sets or the use of feature selection algorithms to find the most relevant attributes for intrusion detection. Our work stands on the shoulders of CICIDS2017's feature design [5], which itself is based on the CICFlowMeter tool. Sharafaldin et al. [5] explain the rationale for these features and provide baseline results on their dataset. By using their features, we ensure comparability with other studies using CICIDS2017. Many researchers who use this dataset apply similar preprocessing, and some apply feature selection (like mutual information ranking, or principal component analysis) to reduce dimensionality. We chose to first evaluate the full feature set to measure the raw potential of the algorithms. The strong results of RF suggest many features might be redundant, since it can split on the most informative ones and ignore others. In future work, we might incorporate an analysis of the top N features that contribute to RF's decisions (e.g., seeing if just the top 20 features yield almost the same performance). This could tie into research on explainable AI for security, where one aims to simplify models and explain decisions.

In summary, our work is differentiated from much of the related literature by its explicit emphasis on **lightweight operation and real-time metrics**. Many academic papers evaluate detection rate and maybe training time, but few provide inference speed or resource usage. We consider these practical metrics as first-class outcomes, aligning with a growing interest in **deployable ML for cybersecurity**. By achieving comparable accuracy to state-of-the-art methods [1][2][4] while quantifying efficiency, we fill an important gap. Moreover, our comparative analysis between LR and RF is instructive for practitioners who must decide between simplicity and power. While previous work often compares multiple algorithms [4], our focus on these two extremes (fast linear vs. slower ensemble) in the context of lightweight IDS offers concrete guidance: if you need ultra-fast throughput and simplicity, LR might suffice for the major threats; if you can afford a bit more computation, RF gives a significant boost in detection coverage. This nuance is valuable when designing a system based on the constraints at hand.

Validation Plan

To ensure that our findings are not just specific to the experimental conditions but truly hold in practical scenarios, we outline a **validation plan** that will guide the next phase of this project. This plan involves two main aspects: **(a)** testing the approach on additional data (or deployment environments) to verify generalization, and **(b)** implementing the model in a realistic setting to monitor its performance in real-time.

1. Cross-Dataset Evaluation: One step is to validate our model on other benchmark datasets that have been used in literature. For instance, we plan to test on **UNSW-NB15** and **NSL-KDD**, or newer datasets like **CSE-CIC-IDS2018**, to see if the models trained on CICIDS2017 (or retrained on those datasets) exhibit similar performance. This addresses generalization across different network environments and attack types. As mentioned earlier, one specific experiment would be a **cross-training validation**: train the model on CICIDS2017 and test it on UNSW-NB15 (after mapping features appropriately). This would simulate a scenario where an IDS trained in one setting is deployed in another – a real concern for practical IDS, which often face evolving network patterns. We expect our lightweight models to still perform reasonably, especially for broad attack categories that appear in both (like DoS or scanning), but performance may drop if there are distribution shifts (different typical traffic rates, different attack implementations). If we find performance issues, that will inform us what adjustments are needed (perhaps retraining on new data or adding some anomaly detection capability for unknown patterns).

2. Real-World Deployment Trial: We plan a pilot deployment of the Random Forest model (and possibly the logistic model as a baseline) in a controlled network environment or on live traffic capture. One approach is to set up the model as a sensor on a network tap or span port, where it receives a live stream of flows (we can use a tool to convert live packets to flows in real time, akin to CICFlowMeter running on live traffic). The model would then classify each flow and raise alerts for any that are predicted as attacks. During such a deployment, we will measure: - The actual **throughput** the model can handle in a streaming context (ensuring it matches our test measurements and identifying any bottlenecks in data pipeline, e.g., feature extraction might be the slow step rather than classification). - The **stability and resource usage** of the model running continuously. For instance, monitor CPU usage on a low-end device (like deploying on a Raspberry Pi or similar single-board computer) to see if the Random Forest can keep up. If not, perhaps logistic regression or a smaller RF (fewer trees) might be needed, which we could test comparatively. - The **accuracy of the model on live traffic**. One challenge is that in real deployment, we often don't have ground truth labels for what's an attack. However, we can orchestrate some known attack traffic in a test network (e.g., perform a port scan, a DoS simulation, etc., while the model is running, to see if it triggers alerts correctly). We can also rely on domain expertise to investigate any flows flagged as malicious by the model to see if they were false alarms (for example, the model might flag a high-volume data transfer as a DoS erroneously; we can detect that and consider adjusting the model or adding post-filters). - We will also look at **integration aspects**: how to feed the model's output into an alerting system or dashboard, and how security analysts might interact with such a model. One advantage of a Random Forest is we could potentially provide explanations like "this flow was flagged because of very high packet rate and byte count," derived from the features that triggered the tree splits. Implementing such an explanation interface would be part of validating the model's practical utility.

3. Robustness Testing: Another validation dimension is robustness against evasion or drift. We plan to simulate how the model handles scenarios like: - **Traffic pattern changes:** If certain features shift due to new benign applications (say a new kind of video streaming that produces unusual flow patterns), does the model misclassify them as attacks? If possible, gather some new benign traffic data (or use a different day of CICIDS2017 that the model didn't see) to test for false positives. - **Adversarial behavior:** Attackers might try to evade detection by mimicking normal traffic patterns. For example, a DoS tool could be throttled to not exceed typical bandwidth of normal users, hoping the model won't flag it. We can attempt to generate such low-and-slow attacks or use existing datasets of stealthy attacks to test the model. If we find that logistic regression misses those (likely) and RF maybe catches more but could still miss cleverly disguised ones, it will motivate adding anomaly detection layers or more sophisticated temporal logic to catch things that per-flow static features can't. This feeds into future improvements but is part of validating what the model can and cannot do. - **Feature stability:** We will verify that the preprocessing steps we did (e.g., handling NaN/inf) hold in all cases in deployment (e.g., no unexpected new NaNs appear). It's a simple check but important for reliability.

4. Comparing with Literature Baselines: To further validate our approach's competitiveness, we plan to directly compare our results to those reported in similar studies. This means possibly implementing other techniques (like an SVM or a deep neural network) on the same dataset split and measuring their performance vs. our Random Forest. If our RF is as good or better (with less complexity) as we believe, this will validate that the chosen approach is indeed an optimal point in the accuracy-efficiency trade space. If another method significantly outperforms ours in detection (especially on stealthy attacks) but is heavier, we will analyze if we can incorporate some elements of that method in a lightweight manner. For example, if a deep learning approach catches infiltration attacks better by analyzing sequences of flows, maybe we incorporate a small sequence model or a rule to handle that particular case. In any event, benchmarking against known results (some of which were cited in related work) will strengthen the validity of our conclusions.

5. Plan for Continuous Improvement: Finally, our validation plan includes a roadmap for iteratively improving the model based on validation findings. For a real-world deployment, one should consider: - **Periodic retraining or updating:** As new attack data comes in, the model should be updated (this could be done via a pipeline that collects newly flagged traffic that was verified and retrains the model every so often). We intend to test a **model update** by adding some new data (e.g., from a different dataset or from live captures) to the training set and seeing if the model adapts without degrading performance on old patterns. This checks the maintainability of a lightweight model. Logistic regression can be updated online relatively easily (using stochastic gradient descent on new samples), and Random Forest could be updated by either training new trees on new data or a complete retrain if needed (which is still fast on our data sizes). We'll formulate a strategy for how frequently and by what mechanism to retrain in deployment. - **Deployment Architecture:** We will outline how the IDS model fits into a larger security system. For instance, the model could run on an edge device monitoring traffic, and alerts are sent to a central server. The validation plan would simulate this architecture to ensure that communication latency or data transfer doesn't become a bottleneck (for truly large-scale deployments, one might not want to send every feature to a central location; instead, local inference is ideal – which is why a small model is beneficial). Our approach of using lightweight models enables **distributed deployment**, which we highlight as an advantage in validation – we can have multiple sensors with identical models covering different subnets, rather than one big system, improving scalability and fault tolerance.

Through these validation steps, we aim to demonstrate that our approach is not just a one-off experiment, but a **practical solution** that can be confidently applied in real cybersecurity operations. By comparing with related works, we ensure we acknowledge where our method stands relative to state-of-the-art. And by executing a robust validation plan, we ensure that the claims of being "lightweight and effective" hold true outside the lab environment. Ultimately, this will increase trust in deploying such machine learning-based IDS in production networks, bridging the gap from research to real-world impact.

Conclusion

In this paper, we presented a comprehensive study on **lightweight feature-based machine learning for network intrusion detection**, using the CICIDS2017 dataset as a testbed. We formulated an experimental modeling approach grounded in the hypothesis that simple supervised models can achieve high attack detection performance with minimal computational overhead. Our case study results strongly support this hypothesis: a Random Forest classifier, using tens of flow features, reached detection rates (precision/recall) in the high 90% range for most attack types, rivaling far more complex methods in the literature, while still maintaining **real-time inference capability** on commodity hardware. Logistic Regression, an even simpler model, showed decent performance on major attack classes and demonstrated blazingly fast prediction speeds (orders of magnitude faster than needed for typical network throughputs), making it an attractive option for extremely resource-constrained scenarios or as a component in an ensemble.

We discussed how our findings compare with existing research [1][2][4], noting that our focus on operational efficiency fills an important niche in IDS development. The trade-offs observed – LR's simplicity vs RF's superior coverage – provide practical insight for engineers deciding on IDS solutions. Notably, the Random Forest's success underscores that much of the discriminatory information in network traffic can be captured by humanengineered features and exploited by classical algorithms, reaffirming that domain knowledge and simpler AI techniques remain vital in cybersecurity.

We also identified areas for future work and validation. The **Related Work and Validation Plan** chapter laid out steps to test the models on different datasets and in live environments. This will ensure the robustness and generalizability of the approach. For instance, testing on an IoT-specific dataset or in a live enterprise network will either corroborate our results or reveal new challenges (like concept drift or new attack patterns) that could be addressed by augmenting the model (e.g., with incremental learning or hybrid anomaly detection).

Another direction for future enhancement is **feature optimization**: our study started with a broad set of features; going forward, we can apply feature selection (perhaps guided by the Random Forest's importance ranking or using techniques like recursive feature elimination) to trim the feature set. Reducing the number of features could further speed up inference and simplify data collection, aligning even more with the lightweight theme. Additionally, fewer features can improve explainability (analysts can focus on a smaller set of indicators). We expect that we might reduce the feature set by half or more with minimal loss in accuracy, based on initial observations of correlated features in CICIDS2017.

Model explainability is another crucial aspect. While we touched on the interpretability benefits of LR and RF, an in-depth analysis could be done. For example, we could extract rules from the Random Forest (conditions that lead to an "attack" classification) to generate a set of human-readable detection rules. This would effectively marry our ML approach with a rule-based IDS, providing the best of both: the model finds patterns, and we convert them into logical rules that can be reviewed and tuned. Security operators often prefer deterministic rules for critical decisions; if we can show that our RF's decisions can be approximated by a small rule set (with conditions on, say, flow duration and byte count etc.), that could ease adoption. This approach has been hinted at in related works that try to simplify tree ensembles for interpretability.

From a deployment perspective, we consider integrating our lightweight models with existing security infrastructure. For instance, deploying it as a plugin to an open-source NIDS like Zeek (formerly Bro) or Snort: these systems could generate the flow features and call our model to score them. Our validation plan's deployment trial is a step in this direction. Ensuring compatibility and ease of integration (perhaps exposing the model as a REST service or a library) will be important for real uptake.

In conclusion, our research demonstrates a promising path forward for **efficient and effective intrusion detection**. By focusing on well-chosen features and proven machine learning algorithms, we achieved a solution that is both high-performing and practical. The work bridges a gap between high-accuracy academic models and the on-the-ground needs of real-time security monitoring. As cyber threats continue to grow in volume and sophistication, such lightweight models can be a key component in the defender's toolkit – enabling deployment of ML-based detection on widespread infrastructure, from powerful servers down to edge devices, without sacrificing performance or requiring exorbitant resources. We envision that with continued validation and refinement, this approach can be readily adopted to enhance network security in various domains. The validation plan and future improvements outlined will further solidify the reliability of our approach, moving it from experimental evidence toward a **validated, deployable IDS solution**.

References

[1] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai (2018). **Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection.** In *NDSS 2018*. (Demonstrates a streaming anomaly-based NIDS with compact features and low latency)

[2] Y. Meidan, M. Bohadana, A. Shabtai, et al. (2018). **N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders.** *IEEE Pervasive Computing, 17*(3), 12–22. (Introduces an efficient IoT-focused IDS using flow features and unsupervised deep learning)

[3] T. T. Nguyen and G. Armitage (2008). **A Survey of Techniques for Internet Traffic Classification Using Machine Learning.** *IEEE Communications Surveys & Tutorials, 10*(4), 56–76. (Provides background on classical ML approaches for traffic classification and intrusion detection)

[4] J. Zhang, et al. (2015). **Feature Selection-Based Network Intrusion Detection Using Random Forest.** *Computers & Security, 49*, 1–15. (Shows the effectiveness of Random Forest and feature selection for flow-based IDS, emphasizing lightweight deployment)

[5] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani (2018). **CICIDS2017: A Dataset for Intrusion Detection Systems.** Proceedings of the *International Conference on Information Systems Security and Privacy (ICISSP)*. (Describes the CICIDS2017 dataset and its features, providing a baseline for IDS evaluation)