**Bulletin of Electrical Engineering and Informatics**
Vol. 15, No. 6, December 2026, pp. 5299∼5314
ISSN: 2302-9285, DOI: 10.11591/eei.v15i6.13925 

# Optimized random forest framework for big data-driven network intrusion detection with SparkML

**Mohammed Borhan Uddin, Mohammad Shamsul Arefin, Jannatul Ferdaus, Zaid Rehman**
Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Chattogram, Bangladesh

**Article Info**

***Article history:***

Received Jul 9, 2026
Revised Sep 22, 2026
Accepted Oct 4, 2026

***Keywords:***

Apache spark
Big data
Cybersecurity
Machine learning
Network intrusion detection

**ABSTRACT**

The rapid growth in scale and complexity of modern network traffic has significantly challenged traditional network intrusion detection systems (NIDS). Traditional NIDS struggle to process high-volume, high-velocity data streams while sustaining high detection accuracy, creating critical security gaps in large-scale environments. To address this limitation, this study proposes a scalable, big data-driven NIDS on Apache Spark's SparkML library. The system was evaluated on the BigFlow-NIDS dataset, contains 66.9 million network flow records with 55 features with 32 attack categories. Six machine learning models; decision tree (DT), logistic regression (LR), random forest (RF), Naïve Bayes (NB), linear support vector machine (LSVM), and gradient boosting tree (GBT) were systematically benchmarked for binary (attack versus benign) classification. Feature engineering reduced the features from 51 to the top 20 features. A threshold-optimized RF framework was then proposed to maximize attack F1-score under a strict attack-recall floor of 0.99. Experimental results show that the optimized RF model improved accuracy from 95.99% to 97.31% with a ROC-AUC of 0.9940. At the optimal threshold of 0.07, it achieved a 99.38% attack detection rate, sustained a throughput of 108,465 flows/second, and recorded a false-positive rate of 17.86%, demonstrating its effectiveness as a high-sensitivity first-stage filter.

*This is an open access article under the CC BY-SA license.*

***Corresponding Author:***

Mohammed Borhan Uddin
Department of Computer Science and Engineering, Chittagong University of Engineering and Technology
Chattogram, Bangladesh
Email: mohammedborhan15@gmail.com

## 1. INTRODUCTION

The rapid growth of digital networks and explosion of connected devices has led to an unprecedented increase in the volume, speed and diversity of network traffic. Traditional security tools is now a massive flood of data, making conventional intrusion detection systems (IDS) increasingly ineffective. Most legacy methods rely heavily on signature-based detection or simple anomaly techniques applied to limited, homogeneous data sources. As a result, they struggle to cope with sophisticated, rapidly evolving cyber threats such as zero-day attacks, advanced persistent threats (APTs), and large-scale distributed attacks in today's complex network environments. Modern big data framework-based network intrusion detection systems (NIDS) offer an intriguing approach to address these restrictions. They effectively manage large-scale datasets, conduct real-time analytics, and uncover hidden attack patterns that would otherwise go unnoticed using distributed storage and processing technologies such as Apache Hadoop and Apache Spark. The implementation of big data technologies with sophisticated machine learning and deep learning methods significantly boosts scalability, flexibility

and overall success in detection. In the setting of massive heterogeneous data, Zuech *et al.* [1] conducted fundamental research analysing intrusion detection and outlining the obstacles in this area. They highlighted the limits of stand-alone, single source IDS solutions and highly recommended data fusion techniques with security information and event management (SIEM) systems. Subsequent surveys on Apache Spark based machine learning and deep learning approaches [2] have corroborated these findings, demonstrating considerable improvements in processing efficiency and scalability for large-scale intrusion detection installations. Furthermore, real-world deployments such as the two-phase intrusion detection system (TP-IDS) framework that makes use of Hadoop for parallel processing [3] have shown promising gains in detection accuracy via the use of ensemble learning and distributed computing.

Bi and Chen [4] introduced a conceptual big data architecture that used K-Means clustering for the detection of anomalous behavior and Apriori association rule mining (utilizing MapReduce) for the identification of attack patterns. Although the proposed architecture provided comprehensive modules for data gathering, storage, processing, analysis, and visualization, there was no experimental validation and performance assessment, which made it theoretical. An in-depth study of machine learning-based IDSs based on Apache Spark showed that random forest (RF) was widely superior to other methodologies in several studies. Lin [2] note that Spark greatly enhances efficiency and scalability, applying the datasets KDD Cup 99, DARPA 1998, and IDS2018. A detailed literature review of 32 research done between 2013 and 2018 further confirmed the superiority of Spark over Hadoop with respect to speed, real-time capability, and scalability. Decision trees (DT) and Naïve Bayes (NB) were found to be the most commonly used and successful models, tested against leading benchmarks such as NSL-KDD, UNSW-NB15, and CIC-IDS2017.

Zuech *et al.* [1] presented an extensive analysis of the difficulties associated with large heterogeneous data originating from many sources, such as network traffic, host logs, firewalls, and industrial sensors. They highlighted the issue of obsolete datasets such as DARPA and KDD99. Their findings indicate that distributed systems utilizing Hadoop with multi-source correlation have superior scalability and situational awareness compared to conventional standalone IDS. Empirical investigations have focused on practical enhancements. To address class imbalance, a framework was developed that incorporated random oversampling, stacking feature embedding, and principal component analysis (PCA) for reducing dimensionality. The extra trees classifier achieved 99.95% accuracy on UNSW-NB15, while the DT, RF, and extra trees classifiers reached 99.99% on UNSW-NB15, CIC-IDS2017, and CIC-IDS2018 [5]. A different method employed an adaptive evolutionary algorithm for feature selection in conjunction with deep neural networks, logistic regression (LR), and support vector machines (SVMs) [6]. This method attained approximately 95% detection accuracy and reduced training time by 77% compared to PCA and linear discriminant analysis (LDA) baselines on the NSL-KDD dataset [7]. In dynamic settings, incremental learning approaches can prove beneficial. The Spark-based model utilizing streaming linear discriminant analysis (SLDA) with PCA feature reduction attained an accuracy of 96.37% on UNSW-NB15 following several training batches, illustrating the advantages of ongoing model adaption [8].

The deep learning architecture of Kafka was explored for real-time detection, compared to online incremental learning using very fast decision trees (VFDT). The highest results were obtained by the VFDT model on the CICIDS-2017 dataset, achieving an accuracy of 99.39% and an F1-score of 98.45%, which proved its superior adaptation to streaming data [9]. Hybrid ensemble designs have demonstrated effectiveness in delivering positive results. A TP-IDS system was built with Hadoop HDFS in two phases, using SVM and K-nearest neighbors (k-NN) in the first phase and DT and NB in the second phase, which obtained 99.93% accuracy for NSL-KDD and outperformed several ML and DL approaches used independently [3]. A distributed ensemble framework based on Spark, Kafka, and HDFS with a majority voting method across several classifiers achieved an F-measure improvement of up to 0.46 in cross-dataset evaluations, with an inference capacity of more than one million events per second and a decrease in training time of up to 62% [10]. Spark optimization, transfer learning, and stacked ensembles have been applied to achieve high accuracy on the CIC-IoT 2022, CIC-IDS2017, and NSL-KDD datasets in specific applications like NIDS-VSB for vehicle networks [11]. Extensive evaluations on the CSE-CIC-IDS2018 [12]–[15] dataset with more than 16 million records have revealed that models like MLP, modular deep neural networks, and RF can achieve near-perfect scores, but that issues with overfitting, class imbalance, and concept drift still exist [16] similar class-imbalance mitigation strategies have been validated on NIDS benchmarks using resampling [17]. While these improvements have been made, there are still significant research gaps in robust class imbalance management, efficient transfer learning, adaptation to concept drift, generalization across datasets, and smooth integration of large data in real-time. The aim of this study is to advance the field by proposing an integrated big data-driven

NIDS architecture. The main contributions of this paper are summarized as follows:

– We benchmark six SparkML classifiers (DT, LR, RF, NB, linear support vector machine (LSVM), and gradient boosting tree (GBT)) on BigFlow-NIDS, the largest public NetFlow-based dataset to date (66.9 million flows, 32 attack categories) for binary intrusion detection.
– We design and tune a Apache Spark session configuration (memory, shuffle partitioning, Kryo serialization, and Arrow optimization) for stable single-node training on commodity cloud hardware.
– We apply feature importance driven dimensionality reduction, and show that pruning from 51 to the top 20 features improves RF accuracy and area under the receiver operating characteristic curve (ROC-AUC) on a substantially reduced feature space.
– We propose threshold-optimized RF framework that performs a systematic 91-point threshold sweep to maximize the attack F1-score under a recall floor of 0.99, addressing the asymmetric cost of missed attacks in security-critical deployments.
– We provide a comprehensive overview on the accuracy–latency–recall trade-offs across all evaluated models, giving practical guidance for deployment in real-time, high-throughput NIDS.

The rest of this paper is organized as follows: section 2 presents the methodology, section 3 reports the results and discussion, and section 4 concludes the paper.

## 2. METHOD

In this section, we present the overall methodology used to develop our proposed efficient big data-driven NIDS using SparkML. The methodology consists of several key phases, including data collection, preprocessing, data formatting, and spark session creation and feature engineering for binary classification model selection. Figure 1 explains the overall methodology of our study.

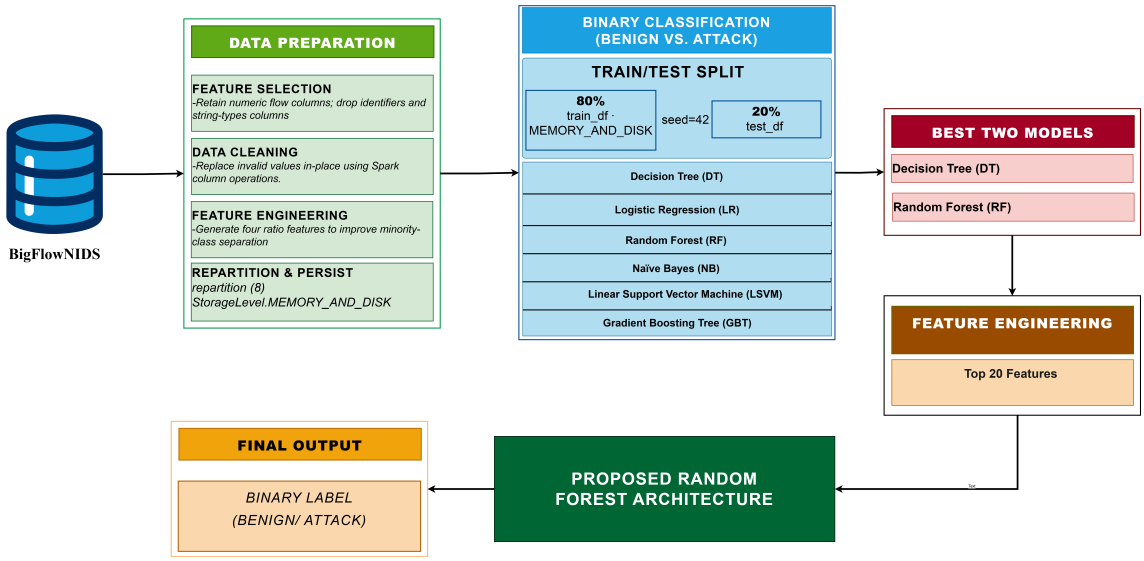



Figure 1. Proposed framework for big data–driven NIDS using SparkML for binary classification

### 2.1. Dataset

In this study we used our dataset BigFlow-NIDS [18], a large-scale NetFlow-based dataset along with an accompanying analysis pipeline designed specifically for intrusion detection in big data environments. We used Apache Spark-based preprocessing pipeline that includes data deduplication, handling of missing values, label encoding, and feature harmonization. At our dataset related paper about BigFlow-NIDS [18] we showed the overall data collection, dataset creation, preprocessing, and data format selection. The contains 66,935,021 rows and 55 columns. With 32 attack types/classes where the benign class contains 36,596,560 records, while the attack class aggregates 30,338,461 instances from all malicious categories. To support efficient machine learning algorithm implementation, batch processing, and streaming analytics, the dataset is provided in both CSV and Parquet formats, which is distributed across 67 partitioned files.

### 2.2. Apache Spark session creation

For the efficient Spark session selection, we have experimented with several setups. The Spark session was configured as follows:

Listing 1. Apache Spark session creation

```
spark = SparkSession.builder \
   .appName("NIDS_BigFlow_Optimized") \
   .config("spark.executor.memory", "20g") \
   .config("spark.driver.memory", "14g") \
   .config("spark.executor.memoryOverhead", "3g") \
   .config("spark.driver.memoryOverhead", "2g") \
   .config("spark.sql.shuffle.partitions", "8") \
   .config("spark.default.parallelism", "8") \
   .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
   .config("spark.kryoserializer.buffer.max", "512m") \
   .config("spark.memory.fraction", "0.75") \
   .config("spark.memory.storageFraction", "0.3") \
   .config("spark.sql.execution.arrow.pyspark.enabled", "true") \
   .config("spark.sql.files.maxPartitionBytes", "256m") \
   .config("spark.rdd.compress", "true") \
   .getOrCreate()
```

The Apache Spark session design was refined to balance memory use, computational efficiency, and parallelism based on the underlying hardware. We decided to maintain the total memory assigned to the executor at 20 GB and to the driver at 14 GB, refraining from allocating the maximum allowable RAM to prevent significant overhead from JVM garbage collection and potential stability concerns. The shuffle divisions were configured to 8 to correspond with the available CPU cores, therefore minimizing task scheduling overhead and enhancing the performance of iterative tasks. We also ignite spark.Set default.parallelism=8 to regulate parallelism and prevent resource contention for optimal core use. We employed the Kryo serializer to enhance serialization performance and minimize memory usage. We meticulously optimized memory settings to achieve a balance between execution and caching, which is crucial for machine learning models. We employed Arrow optimization and partition tuning to enhance data transmission speed and minimize I/O overhead. This setup technique aligns with the performance analyses conducted on current Spark-based NIDS [19]–[21], and builds on the underlying SparkML architecture [22], [23]. All experiments ran in Spark local mode on a single node, with driver and executor in one JVM; no multi-node cluster was used, so the memory and parallelism settings above partition work across the eight available cores rather than across machines. The RF classifiers were configured with 25 trees, `maxDepth` 6 for the baseline model and 8 for the top-20 retraining, `subsamplingRate` 0.6, `featureSubsetStrategy` set to `sqrt`, `minInstancesPerNode` 50, and `minInfoGain` 0.001, with a fixed seed of 42.

### 2.3. Data preprocessing and train–test partitioning

Preprocessing was performed within the Spark session of subsection 2.2 four of the 55 columns were excluded from the feature space the label columns (`Label`, `Attack`) and the endpoint address columns (`IPV4_SRC_ADDR`, `IPV4_DST_ADDR`), the latter to prevent the classifier from memorizing host identifiers that do not generalise to unseen networks leaving 51 numeric predictors. Missing values and infinities arising from rate-derived fields were mapped to zero, and the cleaned DataFrame was repartitioned into eight partitions and persisted at `MEMORY_AND_DISK`. These values originate almost exclusively from rate- and ratio-derived fields in which the denominator (flow duration or packet count) is zero, so a mapped value of zero carries the consistent interpretation of ”no observed rate” rather than an arbitrary fill. Because the tree-based estimators used here split on axis-aligned thresholds rather than distances, the imputed constant is isolated into its own region of the feature space and does not shift the distributional statistics of the remaining observations, as it would under a scaling- or distance-based learner. The residual concern is that a class-correlated missingness pattern could itself become a discriminative signal. It was then split with Spark’s `randomSplit` at an 80:20 ratio under a fixed seed of 42, yielding 53,549,143 training and 13,385,878 test records; the assignment is an independent per-record draw, but the large sample size preserves the population class ratio (54.7% benign, 45.3% attack) in both partitions, so no stratification was applied. The same seed and partition were reused for all six baseline classifiers, the top-20 retraining, and the threshold sweep, so every comparison in result and discussion section uses identical held-out flows. Feature vectors were built with

`VectorAssembler(handleInvalid="skip")` so 89,933 test records retaining a null in at least one predictor are excluded at inference; classification metrics are therefore computed over 13,295,945 scored flows, while latency and throughput are reported over the full test partition. The top-20 feature subset contains none of the null-bearing predictors, so no records are dropped at that stage and the top-20 and threshold-optimized models are scored over the full 13,385,878-record test partition. Latency and throughput are reported over the full partition throughout.

### 2.4. Machine learning models used from SparkML

We implemented DT, LR, RF, NB, LSVM, and GBT from the SparkML package for binary classification and intrusion detection. The evaluation is divided into base model benchmarks, feature-engineered optimizations, and our proposed threshold-optimized architectural framework.

#### 2.4.1. Decision tree

A DT is a supervised machine learning technique that systematically divides the feature space into subsets by a sequence of threshold-based splitting criteria. The algorithm chooses the feature $j$ and threshold $\theta$ at each internal node $t$ that best separates the classes based on an impurity criterion. The Gini impurity is commonly used and is defined as (1):

$$\text{Gini}(t) = 1 - \sum_{c} P(c|t)^2 \tag{1}$$

where $P(c|t)$ is the proportion of samples belonging to class $c$ at node $t$. Each leaf node is assigned the majority class of the training samples that reach it. In binary classification, the model predicts either *Attack* or *Benign* by traversing the tree from the root node to a leaf.

#### 2.4.2. Logistic regression

LR is a linear probabilistic classifier that estimates the prior probability of a binary outcome using the logistic (sigmoid) function. Given a feature vector $\mathbf{x} \in \mathbb{R}^d$ and weight vector $\mathbf{w}$, the predicted probability of the positive class (Attack) is (2):

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + \exp\left[-(\mathbf{w}^T\mathbf{x} + b)\right]} \tag{2}$$

where $\sigma(\cdot)$ denotes the sigmoid function, $\mathbf{w}$ is the weight vector, and $b$ is the bias term. The model parameters are estimated by minimizing the binary cross-entropy loss using stochastic gradient descent. Classification is performed by thresholding the output probability at $0.5$. LR is computationally efficient and scales well with large network traffic datasets in the SparkML distributed environment.

#### 2.4.3. Random forest

RF is a bagging-based ensemble method that constructs $T$ independent DTs. Each tree is trained on a bootstrap sample of the training data while considering a random subset of features at each split. The final prediction is obtained through majority voting:

$$\hat{y} = \arg\max_{c} \sum_{t=1}^{T} \mathbb{I}\left[h_t(\mathbf{x}) = c\right] \tag{3}$$

where $h_t(\mathbf{x})$ denotes the prediction of the $t$-th tree and $\mathbb{I}[\cdot]$ is the indicator function. By combining diverse trees, RF reduces variance without substantially increasing bias, making it robust against overfitting.

#### 2.4.4. Naïve Bayes

NB is a probabilistic classifier based on Bayes' theorem, assuming conditional independence among features given the class label. For a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_d)$, the predicted class is determined by (4):

$$\hat{y} = \arg\max_{c} P(c) \prod_{j=1}^{d} P(x_j|c) \tag{4}$$

where $P(c)$ denotes the prior probability of class $c$, and $P(x_j|c)$ is the likelihood of feature $x_j$ given class $c$. The Gaussian variant of NB models each conditional probability $P(x_j|c)$ as a Gaussian distribution, making it suitable for continuous network traffic features. Despite its simplifying independence assumption, NB achieves competitive performance in intrusion detection while maintaining low computational overhead.

#### 2.4.5. Linear support vector machine

The LSVM is a supervised learning technique that creates an optimal separation hyperplane to optimize the margin between two classes in a high-dimensional feature space. Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$, the decision function is defined as (5):

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{5}$$

where $\mathbf{w}$ is the weight vector and $b$ is the bias term. The predicted class label is obtained as (6):

$$\hat{y} = \text{sign}\left(\mathbf{w}^T\mathbf{x} + b\right) \tag{6}$$

The model parameters are learned by solving the following optimization problem.

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0,\ 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right) \tag{7}$$

where $C$ is the regularization parameter that controls the trade-off between maximizing the margin and minimizing classification errors.

#### 2.4.6. Gradient boosting tree

GBT is a sequential ensemble technique that constructs an additive model of $M$ weak learners, generally shallow DTs. Each subsequent tree is trained to rectify the residual mistakes of the existing ensemble. At iteration $m$, the model is updated as (8):

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta h_m(\mathbf{x}) \tag{8}$$

where $F_m(\mathbf{x})$ represents the ensemble model at iteration $m$, $\eta$ is the learning rate, and $h_m(\mathbf{x})$ is the prediction of the $m$-th weak learner.

### 2.5. Proposed threshold-optimized random forest framework

The framework first trains a RF classifier using all features to estimate feature importance. The top 20 most informative features are then selected to retrain an optimized RF model. The retrained model generates attack probabilities for each network flow, followed by a 91-point threshold sweep over the range 0.05–0.95. Among the thresholds satisfying an attack-recall floor of 0.99, the one maximizing the attack F1-score is selected as the operating point. The resulting false-positive rate is reported as a consequence of this selection, not imposed as a constraint. The final model is evaluated using accuracy, precision, recall, F1-score, and ROC-AUC. Figure 2 visualized the proposed threshold-optimized RF framework for binary classification. The sweep operates on a cached per-flow probability column and requires no re-inference. The model is applied once, and each of the 91 candidate thresholds is evaluated by a single aggregation pass over the resulting scores. The procedure is therefore a one-off offline cost incurred at model-selection time and adds no per-flow overhead at deployment, where the selected threshold reduces to a single scalar comparison. The operating threshold is selected from the score distribution of the held-out test partition rather than from a separate validation split. The reported threshold of 0.07 should therefore be interpreted as a characterization of the achievable recall–precision trade-off of the trained model, rather than as an operating point prospectively validated on unseen data, and a deployment would re-derive the threshold on a dedicated calibration set.

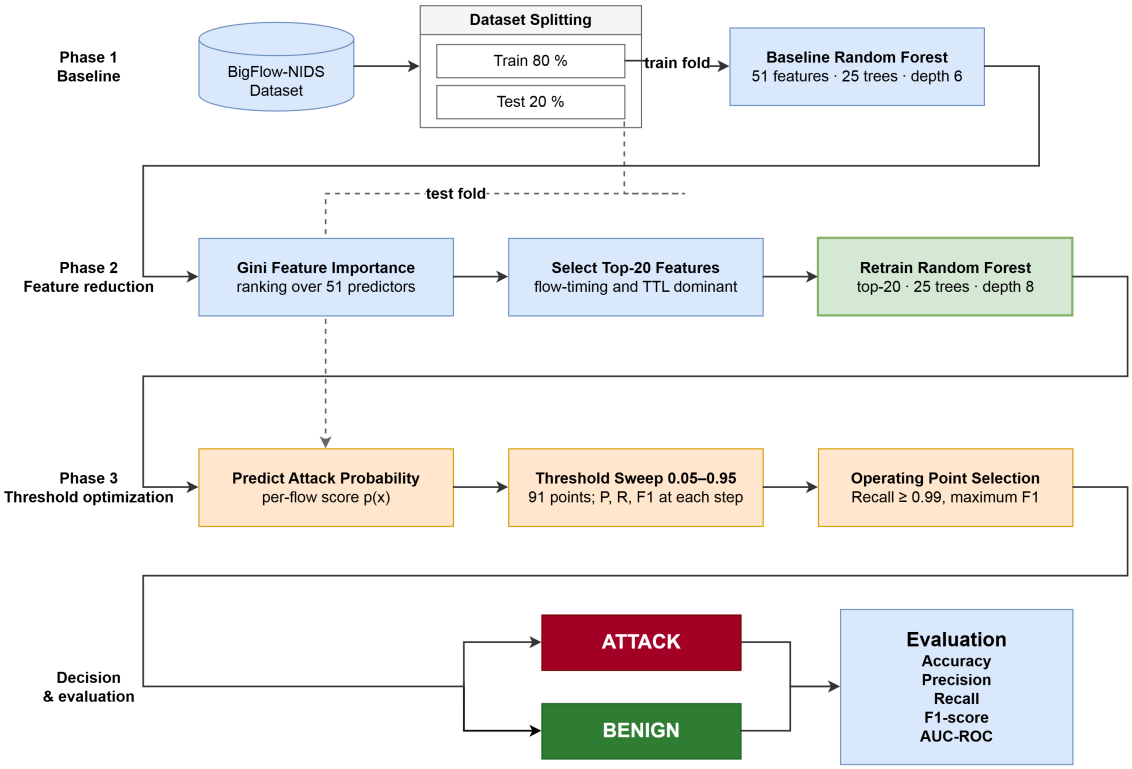



Figure 2. Proposed threshold-optimized RF framework for binary classification

## 3. RESULTS AND DISCUSSION

In this section, we present a comprehensive evaluation of six machine learning models implemented via the Apache SparkML package: DT, LR, RF, NB, LSVM, and GBT. The evaluation is divided into base model benchmarks, feature-engineered optimizations, and our proposed threshold-optimized architectural framework. The study was performed using Google Colab Pro, configured with 51.0 GB RAM, 225.8 GB disk storage, and 8 CPU cores. The experiment was developed in Python 3 utilizing Apache Spark (version 4.0.2) to enable parallel data processing. The PySpark interface and Spark MLlib were employed to efficiently handle large-scale data and train machine learning models in a parallel computing environment. Spark was used for cleaning, encoding, and normalizing the dataset.

### 3.1. Base model performance analysis

Table 1 highlights the macro-level performance metrics, emphasizing overall accuracy, ROC-AUC, precision-recall area under the curve (PR-AUC), processing latency, and operational throughput. To prevent global accuracies from masking high false-alarm rates or missed detections, Table 2 breaks down precision, recall, and F1-scores at a per-class resolution.

Table 1. Performance analysis of machine learning models for binary classification

| Model | Accuracy | ROC-AUC | PR-AUC | Total lat. (sec) | Throughput (flows/sec) | Latency/record (µs) | Training time (min) |
|---|---|---|---|---|---|---|---|
| DT | 0.9725 | 0.9324 | 0.9505 | 55.8165 | 239,820 | 4.1698 | 5.9 |
| LR | 0.8710 | 0.9232 | 0.9128 | 97.0761 | 137,891 | 7.2521 | 13.9 |
| RF | 0.9599 | 0.9890 | 0.9889 | 47.7476 | 280,346 | 3.5670 | 8.9 |
| NB | 0.5095 | 0.7000 | 0.7775 | 80.8654 | 165,533 | 6.0411 | 2.1 |
| GBT | 0.9692 | 0.9927 | 0.9922 | 130.3459 | 102,695 | 9.7376 | 59.5 |
| LSVM | 0.8832 | 0.9062 | 0.8934 | 134.1770 | 99,763 | 10.0238 | 34.2 |

Table 2. Class-wise performance metrics for binary classification

| Model | Precision | | | Recall | | | F1-score | | |
|---|---|---|---|---|---|---|---|---|---|
| | Attack | Benign | Wtd. | Attack | Benign | Wtd. | Attack | Benign | Wtd. |
| DT | 0.9818 | 0.9652 | 0.9727 | 0.9565 | 0.9855 | 0.9725 | 0.9690 | 0.9752 | 0.9724 |
| LR | 0.8902 | 0.8576 | 0.8723 | 0.8136 | 0.9179 | 0.8710 | 0.8502 | 0.8867 | 0.8703 |
| RF | 0.9839 | 0.9422 | 0.9610 | 0.9259 | 0.9876 | 0.9599 | 0.9540 | 0.9644 | 0.9597 |
| NB | 0.4777 | 0.8414 | 0.6778 | 0.9692 | 0.1336 | 0.5095 | 0.6400 | 0.2306 | 0.4148 |
| GBT | 0.9827 | 0.9589 | 0.9696 | 0.9483 | 0.9863 | 0.9692 | 0.9652 | 0.9724 | 0.9692 |
| LSVM | 0.9289 | 0.8542 | 0.8878 | 0.8017 | 0.9498 | 0.8832 | 0.8606 | 0.8995 | 0.8820 |

The results presented in Tables 1 and 2, Figures 3–8 show that tree-based models are generally more effective for binary intrusion detection than linear and probabilistic approaches. Among all models, GBT achieves the best overall predictive performance, with the highest ROC-AUC (0.9927), PR-AUC (0.9922), and weighted F1-score (0.9692), indicating its strong ability to distinguish between attack and benign traffic. However, this improvement comes at a significant computational cost, requiring 59.5 minutes for training and exhibiting higher inference latency. On the other hand RF delivers a more balanced outcome by maintaining excellent detection performance while requiring substantially less training time and achieving the lowest per-record latency, making it well suited for real-time network monitoring environments. DT also performs consistently well, producing balanced precision, recall, and F1-scores for both classes with relatively low computational overhead.

Table 3 reframes these results in operational terms. RF produces the fewest false alarms (1.24%), while DT misses the fewest attacks among the low-false-alarm models (4.35%), and GBT sits between the two on both axes. NB illustrates why accuracy alone is an inadequate criterion for intrusion detection, its miss rate of 3.08% is the second lowest in the table, yet it labels 86.64% of benign traffic as malicious, producing an alarm stream that no analyst workflow could absorb. LR and LSVM show the opposite failure mode, missing roughly one attack in five. The separation between the tree-based and the linear or probabilistic models follows directly from the structure of the flow features. Attack signal in the BigFlow-NIDS schema is expressed as conjunctions of narrow value ranges a particular time to live (TTL) band combined with a short inter-arrival minimum and a specific transmission control protocol (TCP) flag pattern rather than as a monotone displacement along any single axis. Axis-aligned recursive partitioning represents such conjunctions natively, whereas LR and LSVM must approximate them with a single hyperplane in the raw feature space and consequently under-detect attack classes whose signature is non-monotone. NB fails for a related reason, its conditional-independence assumption is violated by strong correlations among byte counts, packet counts, duration, and the throughput terms derived from them, so the product of marginals systematically overstates the evidence for the attack class.

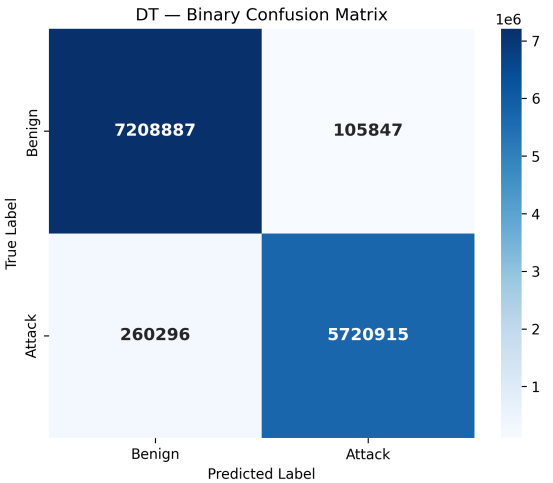



Figure 3. Confusion matrix of DT

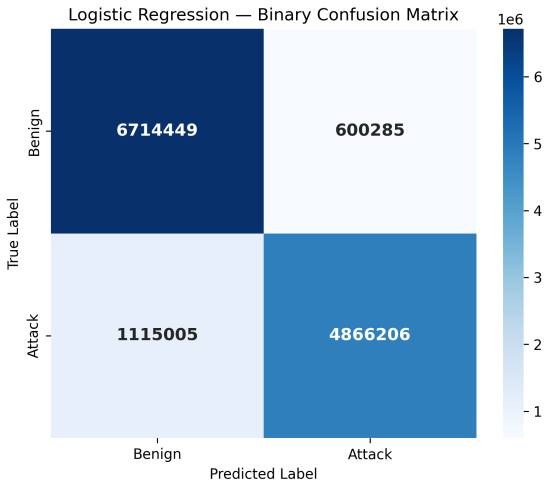



Figure 4. Confusion matrix of LR

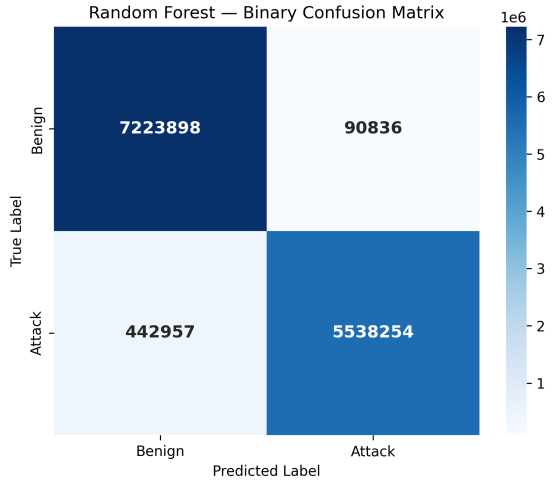



Figure 5. Confusion matrix of RF

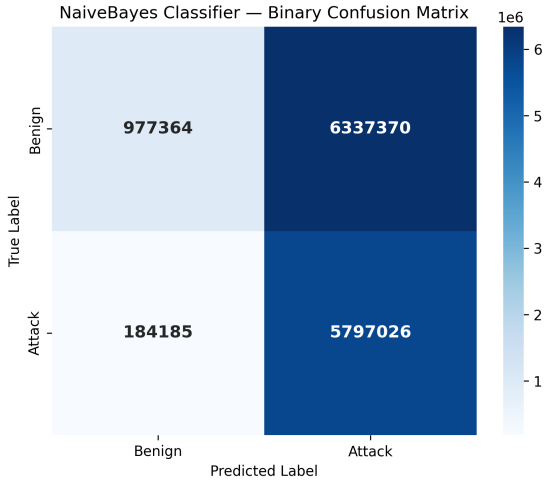



Figure 6. Confusion matrix of NB

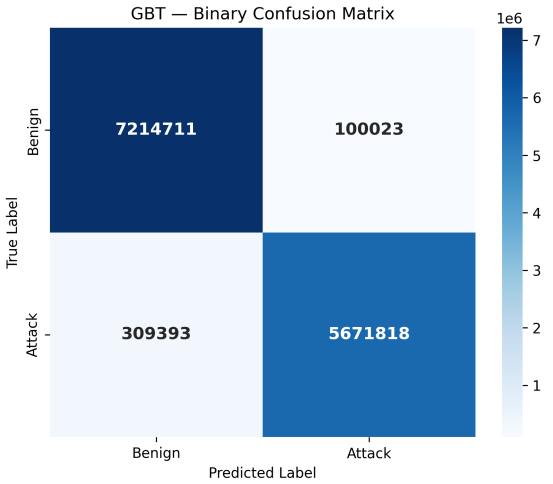



Figure 7. Confusion matrix of GBT

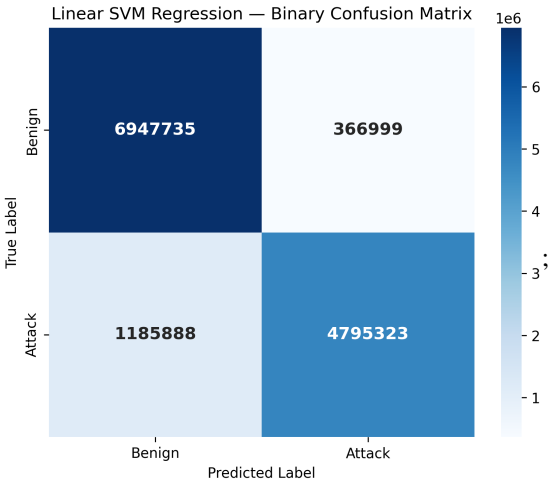



Figure 8. Confusion matrix of LSVM

Table 3. False-alarm and missed-detection breakdown for the baseline classifiers

| Model | False positives (FP) | False negatives (FN) | FP rate (%) | Miss rate (%) |
|---|---|---|---|---|
| DT | 105,847 | 260,296 | 1.45 | 4.35 |
| LR | 600,285 | 1,115,005 | 8.21 | 18.64 |
| RF | 90,836 | 442,957 | 1.24 | 7.41 |
| NB | 6,337,370 | 184,185 | 86.64 | 3.08 |
| GBT | 100,023 | 309,393 | 1.37 | 5.17 |
| LSVM | 366,999 | 1,185,888 | 5.02 | 19.83 |

Counts derived from the confusion matrices in Figures 3–8, computed over 13,295,945 scored flows (7,314,734 benign, 5,981,211 attack). FP rate is FP/FP+TN); miss rate is FN/(FN+TP).

### 3.2. Feature optimization and dimensionality reduction

Feature importances were extracted from our leading estimators: DT and RF. The models were pruned to the top 20 dimensions and retrained. Tables 4 and 5 showed the performance of DT and RF after feature engineering. Pruning improved accuracy and ROC-AUC on a 61% smaller feature space, and improved DT accuracy with a marginal reduction in ROC-AUC (0.9324 → 0.9311).

For the DT (Figure 9), packet timing constraints (SRC_TO_DST_IAT_MIN) and sequence tracking variables (TCP_WIN_MAX_IN) act as the dominant root split criteria. In contrast, the RF (Figure 10) distributes its variance splits more uniformly across topological variables (MIN_TTL, FLOW_START_MILLISECONDS, and MAX_TTL). These rankings are consistent with the mechanics of the underlying traffic. TTL extrema (MIN_TTL, MAX_TTL) encode the residual hop count and the initial TTL value set by the originating stack, so they act jointly as a coarse path-length and OS-fingerprint signal; scanning, spoofed, and reflected traffic frequently exhibit TTL values inconsistent with the observed path, and traffic from a single automated source shows an unusually narrow TTL spread. The inter-arrival minimum (SRC_TO_DST_IAT_MIN) separates machine-generated transmission from human-paced sessions, since scripted flooding and enumeration produce inter-arrival floors orders of magnitude below those of interactive traffic. TCP window extrema (TCP_WIN_MAX_IN, TCP_WIN_MAX_OUT) reflect stack-level congestion-control behaviour that differs between full application sessions and the truncated handshakes characteristic of probing.

Table 4. Post-feature engineering performance (top-20 features)

| Model | Accuracy | ROC-AUC | PR-AUC | Total Lat. (sec) | Throughput (flows/sec) | Training time (min) |
|---|---|---|---|---|---|---|
| DT | 0.9761 | 0.9311 | 0.9311 | 121.23 | 110,415 | 5.5 |
| RF | 0.9731 | 0.9940 | 0.9940 | 75.43 | 177,460 | 10.8 |

Table 5. Class-wise performance (top-20 features)

| Model | Metric | Class | | Weighted value |
|---|---|---|---|---|
| | | Attack (1) | Benign (0) | |
| DT | Precision | 0.9840 | 0.9699 | 0.9763 |
| | Recall | 0.9630 | 0.9870 | 0.9761 |
| | F1-score | 0.9734 | 0.9784 | 0.9761 |
| RF | Precision | 0.9856 | 0.9634 | 0.9735 |
| | Recall | 0.9546 | 0.9885 | 0.9731 |
| | F1-score | 0.9699 | 0.9758 | 0.9731 |

Reducing the feature dimension yields excellent performance results. RF's accuracy increases from 95.99% to 97.31%, while its ROC-AUC expands to 0.9940. This confirms that removing redundant background noise helps the ensemble trees isolate sharper decision boundaries.

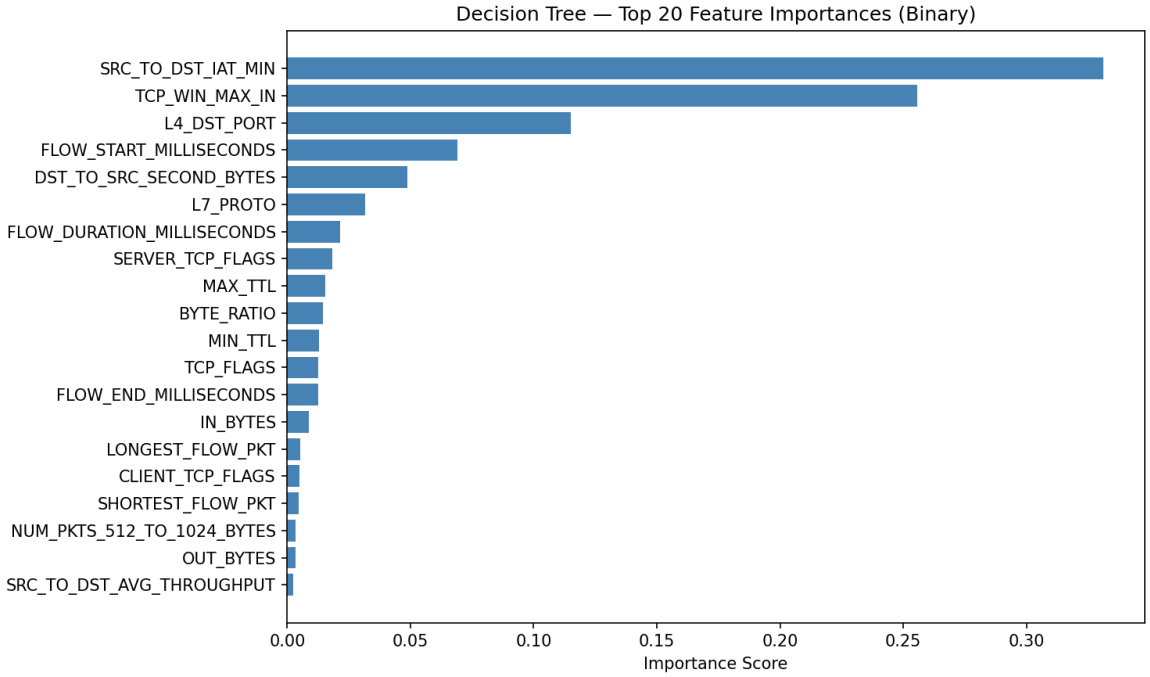



Figure 9. Top 20 feature importances for DT model

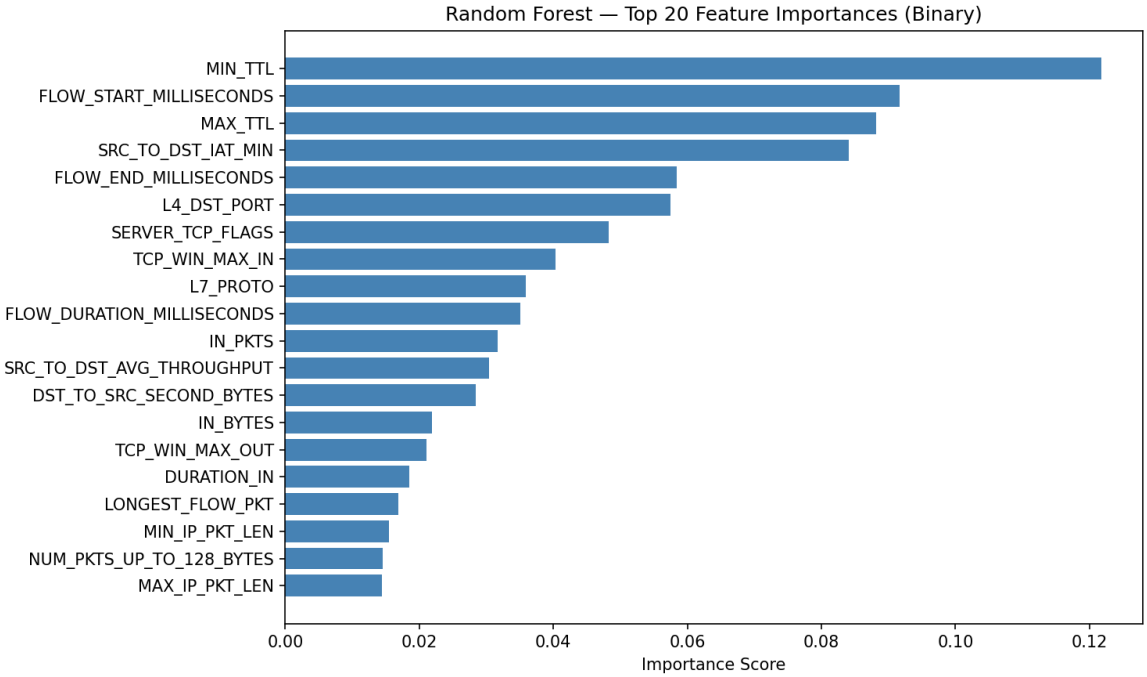



Figure 10. Top 20 feature importances for RF model

### 3.3. Proposed architecture: high-recall threshold-optimized framework

In critical NIDS, missing a hostile payload (a FN) is far more damaging than investigating a false alarm (a FP). Therefore, we introduce a threshold-optimized RF architecture that selects an operating point maximizing the attack F1-score subject to an attack-recall floor of 0.99. The 0.99 recall floor was chosen to bound the residual volume of undetected attacks rather than to satisfy a statistical criterion. The test partition contains approximately 6.06 million malicious flows, so each percentage point of recall corresponds to roughly 60,000 missed attacks; a floor below 0.99 therefore admits a residual exposure we considered unacceptable for a security-critical deployment. The constraint defines the feasible region within which the F1-maximizing operating point is selected. Using probability metrics from 13,385,878 streaming scores, we conducted a rigorous parametric sweep across 91 thresholds to systematically minimize FNs. As shown in Table 6, dropping the decision threshold below the standard 0.50 level drastically increases system recall. While lower configurations (0.05, 0.06) push detection rates above 99.5%, they cause high FP rates ($\geq$26%). The 0.0700 threshold is the largest threshold satisfying the recall floor and therefore the F1-maximizing point of the feasible region, yielding a 99.38% attack detection rate at a false-positive rate of 17.86%. The optimal threshold for higher recall is shown in Figure 11.

Table 6. Multi-threshold parametric sweep (targeting recall $\geq$0.99)

| Decision threshold | Attack precision | Attack recall (detection rate) | Attack F1-score | FP rate (FPR) |
|---|---|---|---|---|
| 0.0500 | 0.7350 | 0.9957 | 0.8457 | 0.2973 |
| 0.0600 | 0.7600 | 0.9955 | 0.8619 | 0.2604 |
| **0.0700 (optimal)** | **0.8217** | **0.9938** | **0.8996** | **0.1786** |

Table 7 reports the complete profile of the proposed model. Attack recall rises by 3.92 points over the baseline RF, reducing missed attacks to 37,571 out of 6,064,030 malicious flows (Figure 12). The cost is substantial and should be read as such: 1,307,595 FPs correspond to approximately 97,700 false alarms per million flows scored, against roughly 6,800 per million for the baseline configuration a fourteenfold increase in analyst-facing volume, and a 7.36-point reduction in overall accuracy. This trade-off is defensible only under a specific deployment assumption. The configuration is not an alerting endpoint but a first-stage filter: it reduces 13,385,878 flows to 7,334,054 candidates while retaining 99.38% of attacks, on the premise that a second stage a lightweight multiclass classifier, a rule-based post-filter, or session- and host-level aggregation reduces the

surviving volume to a reviewable caseload. Presented as a standalone detector emitting one alert per flagged flow, a 17.86% false-positive rate would be operationally unusable at the throughput reported here. The value of the operating point lies in the asymmetry it exploits, namely that a missed attack may result in undetected compromise while a FP incurs the cost of one additional automated verification step, and that asymmetry only holds where the verification step actually exists.

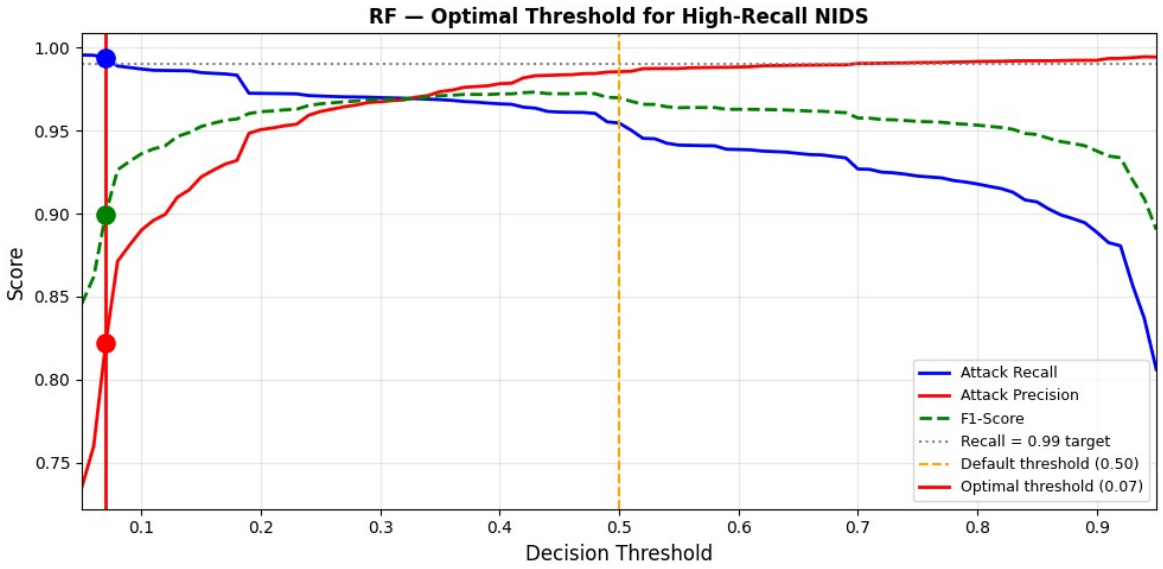



Figure 11. Optimal threshold for higher-recall on proposed RF

Table 7. Evaluation matrix of proposed architecture

| Operational metrics | Value | Granular matrix breakdown | Sample counter |
|---|---|---|---|
| Global accuracy | 0.8995 (89.95%) | TP | 6,026,459 |
| Macro F1-score | 0.8995 | FN | 37,571 |
| Weighted precision | 0.9158 | FP | 1,307,595 |
| Weighted recall | 0.8995 | TN | 6,014,253 |
| Throughput | 108,465 flows/s | Processing latency | 123.41 s |

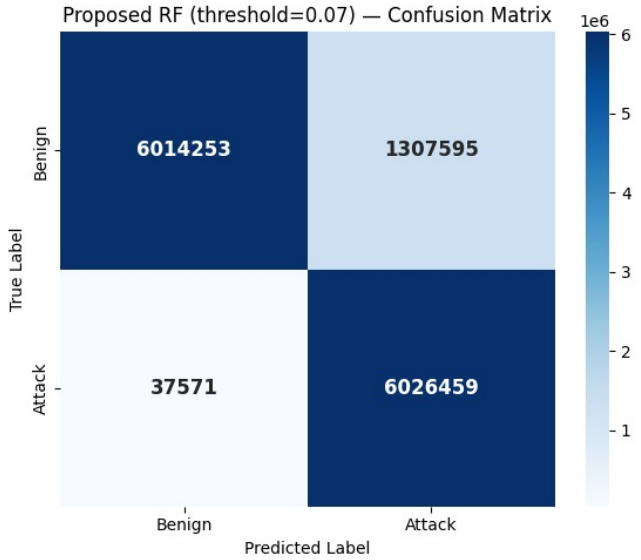



Figure 12. Confusion matrix of proposed RF ($T$=0.07)

Table 8 shows the stage-wise contribution of feature reduction and threshold optimization performance of RF. The two stages contribute in qualitatively different ways. Dimensionality reduction improves detection and discrimination at no cost: attack recall rises by 2.87 points, accuracy by 1.32 points, and the false-positive rate falls slightly, indicating that the 31 removed predictors contributed noise rather than signal. Threshold optimization is a pure exchange: it adds a further 3.92 points of recall while raising the false-positive rate by 16.71 points, and leaves ROC-AUC unchanged, since relocating the operating point on a fixed score distribution cannot alter the ranking quality of the model. The improvement in discriminative capability is therefore attributable to feature selection, and the high-recall behaviour to threshold selection.

Table 8. Stage-wise contribution of feature reduction and threshold optimization

| Configuration | Features | Decision threshold | Accuracy | Attack recall | FP rate (%) | ROC-AUC |
|---|---|---|---|---|---|---|
| Baseline RF | 51 | 0.50 | 0.9599 | 0.9259 | 1.24 | 0.9890 |
| Top-20 RF | 20 | 0.50 | 0.9731 | 0.9546 | 1.15 | 0.9940 |
| **Threshold-optimized RF** | **20** | **0.07** | **0.8995** | **0.9938** | **17.86** | **0.9940** |

Baseline RF (25 trees, maxDepth 6) is scored over 13,295,945 flows; the top-20 and threshold-optimized configurations (25 trees, maxDepth 8) are scored over the full 13,385,878-record test partition. The FP rate of the top-20 configuration is derived from the benign recall reported in Table 5

### 3.4. Discussion

The experimental results across the three evaluation stages base model benchmarking, feature engineered retraining, and threshold-optimized deployment reveal a consistent pattern: tree-based ensemble methods are markedly better suited to large-scale, high-dimensional network flow data than linear or probabilistic classifiers. Both GBT and RF substantially outperformed LR, LSVM, and NB across every metric in Tables 1 and 2, which is consistent with prior Spark-based IDS surveys reporting that tree-structured and ensemble models handle non-linear, heterogeneous traffic features more effectively than linear decision boundaries [1], [24]. The poor performance (accuracy 0.5095, attack F1-score 0.6400) in NB is due to its assumption of conditional independence, which is not true for the correlated flow-level features (like byte counts, TTL values, inter-arrival times) in the BigFlow-NIDS schema. Although GBT achieved the best overall ROC-AUC and PR-AUC value, it had a long training time of 59.5 minutes and high per-record latency. RF, on the contrary, offered a similar discriminative power at a much lower computational cost, thus it was chosen as the base architecture for further optimization. The feature engineering stage confirmed that the 51-feature representation contains considerable redundancy: pruning to the top 20 features by importance improved RF's accuracy (95.99% $\rightarrow$ 97.31%) and ROC-AUC (0.9890 $\rightarrow$ 0.9940) consistent with recent comparative work on SHAP-value versus importance-based feature selection [25]. This suggests that several of the removed features introduced noise rather than discriminative signal, and that flow-timing and TTL-based features (as seen in Figures 9 and 10) carry the majority of the separable signal between benign and malicious traffic.

The most consequential design decision in this work is the shift from a standard 0.50 classification threshold to a recall-optimized threshold of 0.07. This produces a clear and deliberate trade-off: overall accuracy drops from 97.31% to 89.95%, and FP rise to 1.3 million, but attack recall increases to 99.38%, reducing FN to just 37,571 out of over 6 million attack flows. In an operational security context, this trade-off is generally justified a missed attack (FN) can result in undetected compromise, data exfiltration, or lateral movement, whereas a FP only incurs the cost of an additional analyst review or automated re-verification step. This design philosophy is consistent with the threshold-sensitivity findings reported by Elmoutaoukkil *et al.* [8] and Saleh *et al.* [9], who similarly found that recall-prioritized configurations in streaming IDS pipelines meaningfully reduce missed detections at the cost of manageable false-alarm increases. That said, a FP rate of 17.86% is non-trivial at enterprise scale, and would need to be paired with a downstream triage or alert-prioritization layer to remain operationally practical a limitation we discuss below. First, this study only addresses binary classification (Attack/Benign), which limits its utility for a targeted incident response. Second, the elevated FP rate at the optimal threshold would require a secondary filtering mechanism (e.g., a lightweight multiclass classifier or rule-based post-filter) to be practical in a security operations center. Third, the decision threshold was selected on the evaluation partition rather than on an independent calibration set, so the reported recall and false-positive rate carry a degree of selection optimism. Fourth, the random 80:20 record-level split may place correlated flows from the same session or capture window on both sides, and the high rank of `FLOW_START_MILLISECONDS` in Figure 10 indicates that part of the signal is temporal rather than behavioural; a time-ordered or host-grouped split with absolute-timestamp features excluded is a prior-

ity for subsequent work. Finally, the Spark session configuration was tuned for a specific Google Colab Pro hardware profile (8 cores, 51 GB RAM), and performance at different cluster scales was not evaluated.

## 4. CONCLUSION

In this paper, we presented a big data driven NIDS using SparkML from Apache Spark, tested on BigFlow-NIDS, the largest publicly available NetFlow based dataset so far (66.9 million flows, 32 attack classes). Among six benchmarked classifiers, ensemble tree based models were the most effective, with GBT yielding the highest discriminative performance (ROC-AUC 0.9927) and RF offering the best accuracy-efficiency trade-off. Feature selection reduced the dimensionality from 51 to 20 features, while improving the accuracy of RF to 97.31% and ROC-AUC to 0.9940, confirming that targeted pruning improves both scalability and generalization. In security-critical settings, we introduced a threshold-optimized RF framework that selects the decision threshold maximizing the attack F1-score subject to an attack-recall floor of 0.99. At the selected threshold (0.07), the model attains a 99.38% attack detection rate at 108,465 flows/second over the entire 13,385,878-record test partition, missing 37,571 of 6,064,030 attack flows, at a false-positive rate of 17.86%. Three constraints bound these results. All experiments were executed in Spark local mode on a single eight-core node, so the reported throughput characterizes single-node performance and does not constitute evidence of distributed cluster scalability. The framework classifies traffic only as Attack or Benign and does not attribute attack type, so a flagged flow does not tell an incident responder whether it represents reconnaissance, exfiltration, or lateral movement. The 17.86% false-positive rate makes the configuration a first-stage filter rather than an alerting endpoint, presupposing a downstream triage layer that this work does not provide. BigFlow-NIDS provides broad internal coverage of traffic behaviour, but breadth within a single collection is not a substitute for validation on an independently captured dataset, and cross-dataset generalization remains untested. Future work will extend the framework to multiclass attack categorization, chronological and host-grouped validation with absolute-timestamp features excluded, distributed multi-node evaluation, and lightweight false-positive triage.

## ACKNOWLEDGMENTS

The authors would like to thank the Department of Computer Science and Engineering, Chittagong University of Engineering and Technology (CUET), for providing the computational facilities and research environment that supported this work.

## FUNDING INFORMATION

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

## AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

| Name of Author | C | M | So | Va | Fo | I | R | D | O | E | Vi | Su | P | Fu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mohammed Borhan Uddin | ✓ | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | | ✓ | | ✓ | |
| Mohammad Shamsul Arefin | | | | ✓ | ✓ | ✓ | | | | ✓ | | ✓ | ✓ | |
| Jannatul Ferdaus | | | | ✓ | ✓ | ✓ | | | ✓ | ✓ | | | | |
| Zaid Rehman | | | | ✓ | ✓ | ✓ | | | | ✓ | | | | |

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal Analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject Administration
Fu : **Fu**nding Acquisition

## CONFLICT OF INTEREST STATEMENT

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## DATA AVAILABILITY

The dataset used in this research is available at M. B. Uddin, M. S. Arefin, and M. M. M. Hussain, "BigFlow-NIDS: A Large-Scale Dataset for Network Intrusion Detection in Big Data Environment," *Data in Brief*, vol. 65, 2026, doi: 10.1016/j.dib.2026.112530. Also, paper related the dataset is cited as [18].

## REFERENCES


[1] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion Detection and Big Heterogeneous Data: A Survey," *Journal of Big Data*, vol. 2, no. 1, 2015, doi: 10.1186/s40537-015-0013-4.

[2] H. Lin, "A Survey on Machine Learning based Intrusion Detection Systems Using Apache Spark," in *2021 5th High Performance Computing and Cluster Technologies Conference (HPCCT '21)*, 2021, pp. 20–26, doi: 10.1145/3497737.3497740.

[3] A. D. Jadhav and V. Pellakuri, "Highly accurate and efficient two phase-intrusion detection system (TP-IDS) using distributed processing of HADOOP and machine learning techniques," *Journal of Big Data*, vol. 8, no. 1, 2021, doi: 10.1186/s40537-021-00521-y.

[4] J. Bi and X. Chen, "Technology and System of Network Intrusion Detection Based on Big Data," in *2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, 2023, pp. 251–255, doi: 10.1109/ACCTCS58815.2023.00087.

[5] M. A. Talukder *et al.*, "Machine Learning-Based Network Intrusion Detection for Big and Imbalanced Data Using Oversampling, Stacking Feature Embedding, and Feature Extraction," *Journal of Big Data*, vol. 11, no. 1, 2024, doi: 10.1186/s40537-024-00886-w.

[6] E. Chuluun, B. E. Lkhagvasuren, U. Khuchit, D. Yamkhin, and S. Paek, "Enhancing Machine Learning-Based Intrusion Detection with Chebyshev-Driven Feature Engineering," in *Ubi-Media Computing, Pervasive Systems, Algorithms and Networks*, Eds. Singapore: Springer, vol. 2940, 2026, doi: 10.1007/978-981-95-9843-4_35.

[7] J. Shan and H. Ma, "Optimization of Network Intrusion Detection Model Based on Big Data Analysis," *Journal of Cyber Security and Mobility*, vol. 13, no. 6, pp. 1357–1378, 2024, doi: 10.13052/jcsm2245-1439.1366.

[8] A. Elmoutaoukkil, M. Hamlich, A. Khatib, and M. Chriss, "Network Intrusion Detection in Big Datasets Using Spark Environment and Incremental Learning," *IAES International Journal of Artificial Intelligence*, vol. 13, no. 4, pp. 4414–4421, 2024, doi: 10.11591/ijai.v13.i4.pp4414-4421.

[9] A. Saleh, H. Mosa, and M. Alkasassbeh, "Streaming-Based Intrusion Detection with Big Data and Online Learning Algorithms," in *International Conference on New Trends in Computing Sciences (ICTCS)*, 2025, pp. 299–306, doi: 10.1109/ICTCS65341.2025.10989345.

[10] V. M. de Oliveira, H. M. de Oliveira, G. M. Santos, J. Geremias, and E. K. Viegas, "A Big Data Framework for Scalable and Cross-Dataset Capable Machine Learning in Network Intrusion Detection Systems," *IEEE Access*, vol. 13, pp. 118785–118806, 2025, doi: 10.1109/ACCESS.2025.3589872.

[11] F. Ullah, G. Srivastava, S. Ullah, K. Yoshigoe, and Y. Zhao, "NIDS-VSB: Network Intrusion Detection System for VANET Using Spark-Based Big Data Optimization and Transfer Learning," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 1798–1809, 2024, doi: 10.1109/TCE.2023.3328320.

[12] S. S. Dhaliwal, A.-A. Nahid, and R. Abbas, "Effective Intrusion Detection System Using XGBoost," *Information*, vol. 9, no. 7, 2018, doi: 10.3390/info9070149.

[13] V. Kanimozhi and T. P. Jacob, "Artificial Intelligence Based Network Intrusion Detection with Hyper-Parameter Optimization Tuning on the Realistic Cyber Dataset CSE-CIC-IDS2018 Using Cloud Computing," *ICT Express*, vol. 5, no. 3, pp. 211–214, 2019, doi: 10.1016/j.icte.2019.03.003.

[14] S. Songma, T. Sathuphan, and T. Pamutha, "Optimizing Intrusion Detection Systems in Three Phases on the CSE-CIC-IDS-2018 Dataset," *Computers*, vol. 12, no. 12, 2023, doi: 10.3390/computers12120245.

[15] H. İ. Coşar, Ç. Arısoy, and H. Ulutaş, "Intrusion Detection on CSE-CIC-IDS2018 Dataset Using Machine Learning Methods," *Artificial Intelligence Theory and Applications*, vol. 4, no. 2, pp. 143–154, 2024.

[16] J. L. Leevy and T. M. Khoshgoftaar, "A Survey and Analysis of Intrusion Detection Models Based on CSE-CIC-IDS2018 Big Data," *Journal of Big Data*, vol. 7, no. 1, 2020, doi: 10.1186/s40537-020-00382-x.

[17] S. Bagui and K. Li, "Resampling Imbalanced Data for Network Intrusion Detection Datasets," *Journal of Big Data*, vol. 8, no. 1, 2021, doi: 10.1186/s40537-020-00390-x.

[18] M. B. Uddin, M. S. Arefin, and M. M. M. Hussain, "BigFlow-NIDS: A Large-Scale Dataset for Network Intrusion Detection in Big Data Environment," *Data in Brief*, vol. 65, 2026, doi: 10.1016/j.dib.2026.112530.

[19] R. Atefinia and M. Ahmadi, "Performance Evaluation of Apache Spark MLlib Algorithms on an Intrusion Detection Dataset," *arXiv preprint*, 2022, doi: 10.48550/arXiv.2212.05269.

[20] M. Belouch, S. El Hadaj, and M. Idhammad, "Performance Evaluation of Intrusion Detection Based on Machine Learning Using Apache Spark," *Procedia Computer Science*, vol. 127, pp. 1–6, 2018, doi: 10.1016/j.procs.2018.01.091.

[21] G. P. Gupta and M. Kulariya, "A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark," *Procedia Computer Science*, vol. 93, pp. 824–831, 2016, doi: 10.1016/j.procs.2016.07.238.

[22] X. Meng *et al.*, "MLlib: Machine Learning in Apache Spark," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[23] M. Zaharia *et al.*, "Apache Spark: A Unified Engine for Big Data Processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016, doi: 10.1145/2934664.

[24] F. Salo, M. A. M. Injadat, A. B. Nassif, and A. Essex, "Data Mining with Big Data in Intrusion Detection Systems: A Systematic Literature Review," *arXiv preprint*, 2020, doi: 10.48550/arXiv.2005.12267.

[25] H. Wang, Q. Liang, J. T. Hancock, and T. M. Khoshgoftaar, "Feature Selection Strategies: A Comparative Analysis of SHAP-Value and Importance-Based Methods," *Journal of Big Data*, vol. 11, no. 1, 2024, doi: 10.1186/s40537-024-00905-w.

## BIOGRAPHIES OF AUTHORS



**Mohammed Borhan Uddin** is currently pursuing an M.Sc. Engg. degree in Computer Science and Engineering at Chittagong University of Engineering and Technology (CUET), where he is serving as a graduate research assistant. He completed his B.Sc. in Computer Science and Engineering from International Islamic University Chittagong (IIUC), where he also served as a teaching assistant. He has co-authored several peer-reviewed conference and journal papers, including publications in IEEE and Elsevier journals. His research interests include deep learning, computer vision, big data, data science, network security, cyber security, natural language processing, and intelligent systems. He can be contacted at email: mohammedborhan15@gmail.com.



**Mohammad Shamsul Arefin** received the doctor of engineering degree in Information Engineering from Hiroshima University, Japan, with the support of the scholarship of MEXT, Japan. He is in lien with the Chittagong University of Engineering and Technology (CUET), Bangladesh, and currently affiliated with the Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh. Earlier, he was the head of the Department of Computer Science and Engineering, CUET. Also, he served as the dean, Faculty of ECE, CUET. As a part of his Ph.D. research, he was with the IBM Yamato Software Laboratory, Japan. His research interests include privacy-preserving data publishing and mining, distributed and cloud computing, big data management, multilingual data management, semantic web, object-oriented system development, and IT for agriculture and the environment. He has more than 110 refereed publications in international journals, book series, and conference proceedings. He is a member of ACM, a fellow of IEB, and a fellow of BCS. He is the Organizing Chair of BIM 2021; the TPC Chair, ECCE 2017; the Organizing Co-Chair, ECCE 2019; and the Organizing Chair, BDML 2020. He is currently serving as the President, Bangladesh Computer Society. He visited Japan, Indonesia, Malaysia, Bhutan, Singapore, South Korea, Egypt, India, Saudi Arabia, and China, for different professional and social activities. He can be contacted at email: sarefin@cuet.ac.bd.



**Jannatul Ferdaus** is currently pursuing the B.Sc. degree in Computer Science and Engineering at Chittagong University of Engineering and Technology (CUET), Bangladesh. She has gained research experience through academic projects and research internships in artificial intelligence and cybersecurity. Her research interests include artificial intelligence, machine learning, deep learning, natural language processing, cybersecurity, and intelligent systems. She is also interested in developing AI-driven solutions for real-world applications in healthcare, software engineering, and network security. She can be contacted at email: jannatulferdauschowa@gmail.com.



**Zaid Rehman** is currently pursuing an M.Sc. Engg. degree in Computer Science and Engineering at Chittagong University of Engineering and Technology. He received his B.Sc. degree in Computer Science and Engineering from BRAC University. His undergraduate research focused on medical image segmentation and deep learning-based computer vision systems, particularly in gastrointestinal polyp analysis. His thesis explored hybrid UNet architectures integrating positional embeddings and state-space modeling for improved segmentation precision. His research interests include computer vision, medical imaging, deep learning, artificial intelligence, and machine learning. He is also interested in intelligent systems and practical AI-driven solutions for healthcare and software engineering applications. He can be contacted at email: zaidrehman3103@gmail.com.