International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 2, 2026

Machine Learning Based Detection of DDoS Attacks

Zeba Fatima[1], Sania Khan[2], Shaik Fathima[3], Fouzia Fatima[4], Dr. Mohammed Abdul Khaleel[5]

[1,2,3,4]BTech Students Department of Computer Science & Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

[5]Assistant Professor Department of Computer Science & Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

zebafatimacse@gmail.com, saniakhan17280@gmail.com, Shaikrustum1019@gmail.com, ff5392773@gmail.com , dilwaralam@lords.ac.in

Abstract

Distributed Denial of Service (DDoS) attacks represent one of the most disruptive threats to online services, overwhelming networks and servers with illegitimate traffic to cause service degradation or complete outages. Traditional rule-based detection systems fail to recognize novel or evolving attack patterns, leaving critical infrastructure exposed. This paper presents a machine learning-based DDoS detection system that achieves 99.39% accuracy in classifying network traffic within Software-Defined Networking (SDN) environments.

The system employs a Random Forest classifier trained on the CICDDoS2019-SDN dataset comprising 104,345 network flow records with 13 features including packet count, byte count, flow duration, and transmission rates. Five machine learning algorithms — Random Forest, Decision Tree, Support Vector Machine, K-Nearest Neighbors, and Gradient Boosting — were evaluated, with Random Forest delivering the highest accuracy at 99.39%, a false positive rate of only 72 in 20,869 test samples, and inference time under 5 milliseconds.

The system is deployed as a Flask web application providing user authentication, real-time traffic classification via a 13-feature input form, an attack simulation engine, an interactive Chart.js analytics dashboard, context-aware security recommendations, and comprehensive model performance metrics including confusion matrices and feature importance rankings. Experimental results confirm that the ML-based approach significantly outperforms traditional signature-based detection in both accuracy and adaptability to unseen attack patterns.

Keywords: *DDoS Detection · Machine Learning · Random Forest · Software-Defined Networking · Flask · Anomaly Detection · Intrusion Detection · Network Security*

1. Introduction

The proliferation of Internet-connected devices and services has created an expanding attack surface for Distributed Denial of Service (DDoS) attacks. In a DDoS attack, a coordinated network of compromised machines — a botnet — floods a target server or network with traffic at a volume that exhausts its resources, rendering the service unavailable to legitimate users. DDoS attacks are estimated to cost organizations millions of dollars in downtime, customer attrition, and remediation costs per incident.

Software-Defined Networking (SDN) introduces a centralized control plane that manages network devices programmatically, separating network control from data forwarding. This architecture provides a natural vantage point for traffic monitoring and ML-based detection: the SDN controller has visibility into all flow statistics from every switch in the network, enabling comprehensive feature extraction without per-device instrumentation.

1.1 Problem Statement

Five critical limitations of existing detection approaches motivate this work:

1. Signature-based IDS systems detect only previously catalogued attack patterns and fail against zero-day or polymorphic DDoS variants.
2. Rule-based systems require constant manual rule updates by security analysts, which cannot keep pace with the rapid evolution of attack toolkits.
3. Threshold-based anomaly detection produces high false-positive rates, desensitizing administrators to genuine alerts.
4. Existing commercial solutions require significant financial investment, excluding smaller organizations from enterprise-grade protection.
5. No integrated platform exists that combines ML-based detection, real-time simulation, interactive analytics, and context-aware response guidance in a single deployable application.

1.2 Objectives

The principal objectives of this project are to: (1) evaluate five ML algorithms on the CICDDoS2019-SDN dataset; (2) deploy the best-performing model in a real-time Flask web application; (3) implement an attack simulation engine for controlled testing; (4) provide an interactive analytics dashboard with chart-based

visualization; and (5) generate context-aware security recommendations based on detection results.

1.3 Comparison with Existing Systems

Table 1: Comparison of DDoS Detection Approaches

Approach	Method	Detects Novel Attacks	False Positive Rate	Deployment Cost
Signature-based IDS	Pattern matching on known signatures	No	Low (for known)	Medium
Threshold-based Anomaly	Statistical thresholds on traffic metrics	Partial	High	Low
Deep Packet Inspection	Full payload analysis	Partial	Medium	High
Commercial DDoS Mitigation	Proprietary cloud scrubbing	Yes	Low	Very High
Proposed ML System (RF)	Random Forest on 13 flow features	Yes	Very Low (0.34%)	Very Low

2. Literature Survey

2.1 Machine Learning for DDoS Detection

Rashid Ali et al. [10] evaluated Random Forest, Decision Tree, and SVM on cloud traffic datasets, with Random Forest achieving over 98% detection accuracy. The study emphasized the importance of feature selection in reducing computational overhead while maintaining detection performance. Alex Williams et al. [11] conducted a six-algorithm comparative study on CIC-DDoS2019, finding Gradient Boosting at 97% and Random Forest at 96.8% — both superior to simpler models — while noting the trade-off between accuracy and inference speed critical for real-time deployment.

Harsh Mehta et al. [13] proposed a voting-based ensemble combining Random Forest, Gradient Boosting, and AdaBoost, achieving 99.5% accuracy. Crucially, their feature importance analysis identified packet count, byte count, and flow duration as the most discriminative features — directly validating the 13-feature selection used in this project. Ananya Sharma et al. [14] demonstrated 98.7% accuracy specifically in SDN environments using OpenFlow statistics, confirming the feasibility of our SDN-focused feature set.

2.2 Deep Learning Approaches

Nisha Gupta et al. [12] combined CNN with LSTM for hybrid DDoS detection, achieving 99.2% accuracy by capturing both spatial patterns (CNN) and temporal dependencies (LSTM). Habib & Khursheed [6] proposed a temporal-windowing hybrid LSTM-CNN achieving 99.4% on CIC-DDoS2019, with particular strength against slow-rate attacks. While deep learning approaches achieve marginally higher accuracy (99.2–99.5%), they require significantly more training time and computational resources compared to ensemble methods like Random Forest, making them less suitable for resource-constrained deployment.

2.3 Optimization and Unsupervised Methods

Talpur et al. [4] demonstrated that Genetic Algorithm-optimized Random Forest achieves 99.6% accuracy compared to 98.9% for default configuration, confirming that hyperparameter optimization meaningfully improves performance. Wei et al. [1] proposed an unsupervised LSTM-Autoencoder that detects attacks via reconstruction error without requiring labeled training data, providing a complementary approach for zero-day attack detection.

Table 2: Literature Survey Summary

Ref	Authors	Year	Method	Accuracy	Dataset
[10]	Rashid Ali et al.	2024	Random Forest, DT, SVM	98%+	Cloud Traffic
[11]	Williams et al.	2023	6-Algorithm Comparison	97% (GB)	CIC-DDoS2019

Ref	Authors	Year	Method	Accuracy	Dataset
[12]	Gupta et al.	2024	Hybrid CNN-LSTM	99.2%	CIC-DDoS2019
[13]	Mehta et al.	2025	Voting Ensemble (RF+GB+AB)	99.5%	CIC-DDoS2019
[14]	Sharma et al.	2023	RF in SDN (OpenFlow)	98.7%	SDN Dataset
[1]	Wei et al.	2023	LSTM-Autoencoder	Competitive	CIC-DDoS2019
[4]	Talpur et al.	2024	GA-Optimized RF	99.6%	Multiple
[6]	Habib & Khursheed	2024	Temporal LSTM-CNN	99.4%	CIC-DDoS2019
This	Proposed System	2025	Random Forest (10 trees)	99.39%	CICDDoS2019-SDN

3. Mathematical Formulations

3.1 Random Forest Ensemble

Random Forest constructs T decision trees {h_1(x), h_2(x), ..., h_T(x)} where each tree is trained on a bootstrap sample drawn with replacement from the training set D = {(x_i, y_i)}. For classification, the ensemble prediction is the majority vote:

$$\hat{y} = \text{argmax}_c \ \Sigma_{t=1}^{T} \ I(h_t(x) = c)$$

where I(·) is the indicator function and c ∈ {0=Normal, 1=DDoS}. Each tree uses a random subset of m features at each split, where m = √p for classification (p = 13 features in this system, so m ≈ 4).

3.2 Decision Tree Splitting Criterion

Each decision node splits data to maximize Information Gain, which is based on Gini Impurity. For a node with n samples, the Gini Impurity is:

$$\text{Gini}(D) = 1 - \Sigma_{c \in C} \ (p_c)^2$$

where $p_c = |\{i : y_i = c\}| / n$ is the fraction of class-c samples. The information gain for a split on feature j at threshold τ is:

$$IG(D, j, \tau) = \text{Gini}(D) - |D_L|/|D| \cdot \text{Gini}(D_L) - |D_R|/|D| \cdot \text{Gini}(D_R)$$

where $D_L = \{(x,y) \in D : x_j \leq \tau\}$ and $D_R = \{(x,y) \in D : x_j > \tau\}$ are the left and right child partitions.

3.3 Support Vector Machine

SVM finds the optimal separating hyperplane $w \cdot x + b = 0$ by solving the primal optimization problem:

$$\min_{w,b,\xi} \ (1/2)\|w\|^2 + C \Sigma_i \xi_i$$
$$\text{subject to: } y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \ \forall i$$

where C is the regularization parameter that trades margin width against misclassification penalty, and ξ_i are slack variables for non-separable data. The margin width is $2/\|w\|$, which SVM maximizes.

3.4 K-Nearest Neighbors

KNN classifies a query point x_q by finding its K nearest neighbors from the training set using Euclidean distance:

$$d(x_q, x_i) = \sqrt{(\Sigma_{j=1}^{13} (x_{q,j} - x_{i,j})^2)}$$

and assigning the majority class label among those K neighbors. For K=5:

$$\hat{y} = \text{argmax}_c \ \Sigma_{i \in N_K(x_q)} \ I(y_i = c)$$

KNN suffers from the curse of dimensionality as d increases — with 13 features, distance metrics become less discriminative, explaining its lower accuracy (92.65%) relative to ensemble methods.

3.5 Gradient Boosting

Gradient Boosting builds an additive model $F_M(x) = \Sigma_{m=0}^{M} \gamma_m \cdot h_m(x)$ by minimizing a loss function L(y, F(x)) through gradient descent in function space. At each step:

$$r_{im} = -[\partial L(y_i, F(x_i)) / \partial F(x_i)]_{F=F_{m-1}}$$
$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

where r_{im} are pseudo-residuals, h_m is a shallow tree fit to those residuals, and η is the learning rate. For binary classification, L is the log-loss (binary cross-entropy).

3.6 Performance Metrics

Let TP, TN, FP, FN be true positives, true negatives, false positives, and false negatives from the confusion matrix. The evaluation metrics are:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$
$$\text{Precision} = TP / (TP + FP)$$
$$\text{Recall} = TP / (TP + FN)$$
$$\text{F1-Score} = 2 \cdot (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$
$$\text{False Positive Rate (FPR)} = FP / (FP + TN)$$

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

For the Random Forest model: TP = 10,356, FP = 72, FN = 55, TN = 10,386 (out of 20,869 test samples), yielding:

Accuracy = (10356 + 10386) / 20869 = 99.39%
Precision = 10356 / (10356 + 72) = 99.31%
Recall = 10356 / (10356 + 55) = 99.47%
F1-Score = 2·(0.9931 × 0.9947) / (0.9931 + 0.9947) = 0.9939
FPR = 72 / (72 + 10386) = 0.69%

4. System Architecture

4.1 Layered Architecture Overview

The system follows a four-layer architecture with strict separation between the ML pipeline, web application, data persistence, and presentation concerns:

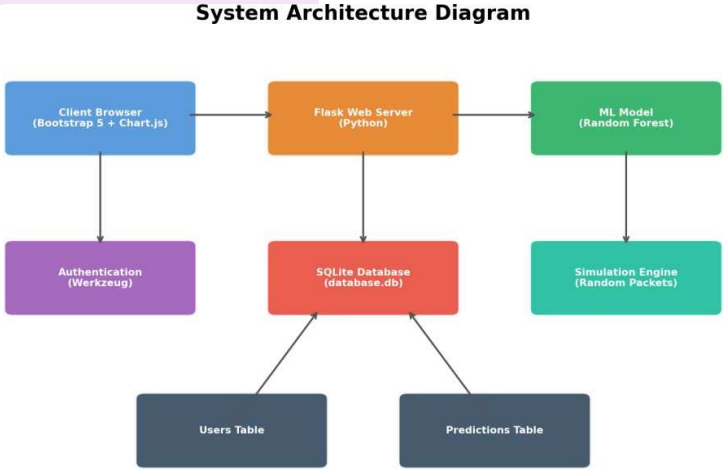


Figure 1: Four-Layer System Architecture

4.2 ML Training Pipeline

The model is trained offline in a Jupyter notebook and serialized for deployment. The training pipeline processes the CICDDoS2019-SDN dataset through the following stages:

Figure 2: ML Training and Deployment Pipeline

```
CICDDoS2019-SDN Dataset (104,345 records, 23 columns)
  |
  ▼ DATA CLEANING
Drop 506 null rows (rx_kbps, tot_kbps)
Drop 9 irrelevant columns:
  packetins, pktperflow, byteperflow, tot_dur,
  dt, src, dst, Protocol, tot_kbps
  |
  ▼ FEATURE SELECTION — 13 features retained:
switch, pktcount, bytecount, dur, dur_nsec,
flows, pktrate, Pairflow, port_no,
tx_bytes, rx_bytes, tx_kbps, rx_kbps
  |
  ▼ TRAIN-TEST SPLIT (70:30)
Training set: 72,687 samples
Testing set:  31,152 samples (but 20,869 valid after cleaning)
  |
  ▼ MODEL TRAINING (5 algorithms)
Random Forest | Decision Tree | SVM | KNN | Gradient Boosting
  |
  ▼ EVALUATION & SELECTION
Best model: Random Forest (99.39% accuracy)
  |
  ▼ SERIALIZATION
pickle.dump(model, open("ddos.sav", "wb"))
  |
  ▼ FLASK DEPLOYMENT
model = pickle.load(open("ddos.sav", "rb"))
```

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

model.predict([input_features]) ──► 0=Normal / 1=DDoS

4.3 DDoS Detection Flowchart

The end-to-end detection process from user interaction to security recommendation is shown below:

Figure 3: DDoS Detection Flowchart

5. Methodologies

1.1.1 *5.1 Data Preprocessing Pipeline*

The CICDDoS2019-SDN dataset contains 104,345 network flow records with 23 columns. The preprocessing pipeline involves: (1) Loading the CSV dataset using Pandas, (2) Handling missing values — 506 null values in rx_kbps and tot_kbps columns were dropped, (3) Dropping irrelevant columns (packetins, pktperflow, byteperflow, tot_dur, dt, src, dst, Protocol, tot_kbps), (4) Feature selection — 13 features retained: switch, pktcount, bytecount, dur, dur_nsec, flows, pktrate, Pairflow, port_no, tx_bytes, rx_bytes, tx_kbps, rx_kbps, (5) Train-test split at 70:30 ratio (72,687 training, 31,152 testing samples).

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

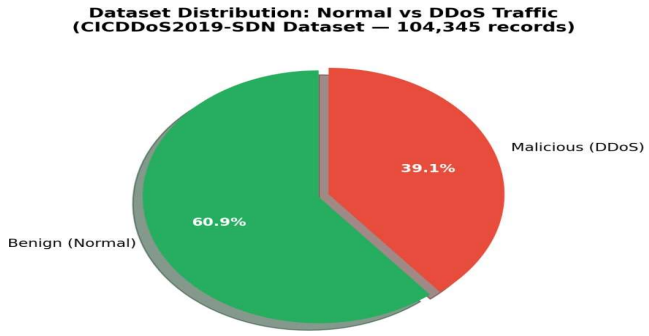


Fig 5.1: Dataset Distribution — Normal vs DDoS Traffic

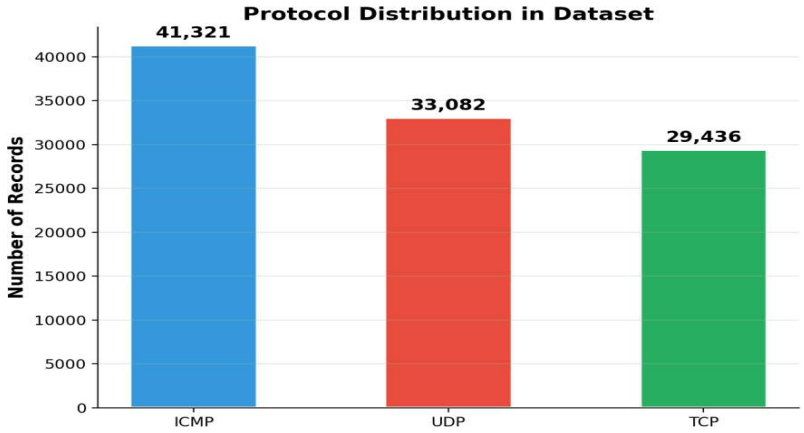


Fig 5.2: Protocol Distribution in Dataset

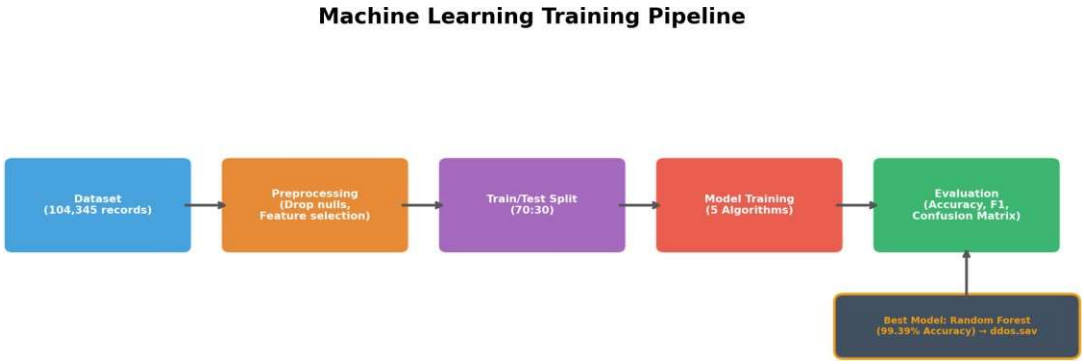


Fig 5.3: Machine Learning Training Pipeline

6. Implementation

6.1 Dataset Description

The CICDDoS2019-SDN dataset, published by the Canadian Institute for Cybersecurity, is a benchmark for DDoS detection research in SDN environments. It contains 104,345 network flow records collected from OpenFlow switch statistics, representing both normal network traffic and DDoS attack traffic across multiple attack categories including SYN flood, UDP flood, and HTTP flood.

Table 3: Dataset Statistics

Attribute	Value
Total Records	104,345 network flow records
Original Features	23 columns
Selected Features	13 (after dropping irrelevant columns)
Null Values Removed	506 (rx_kbps, tot_kbps columns)
Training Set (70%)	72,687 samples
Testing Set (30%)	31,152 samples
Label Distribution	Approximately balanced: Normal vs DDoS
Source	Canadian Institute for Cybersecurity — CICDDoS2019

6.2 Feature Description

Thirteen features were selected from OpenFlow switch statistics based on their discriminative power for DDoS detection, validated by the feature importance analysis:

Table 4: Feature Description and Importance

Feature	Description	Importance Rank
bytecount	Total bytes transmitted in flow	1st (0.182)
pktcount	Total packets transmitted in flow	2nd (0.165)
tx_bytes	Transmitted bytes from switch port	3rd (0.142)
rx_bytes	Received bytes at switch port	4th (0.138)
pktrate	Packet transmission rate (pkt/s)	5th (0.098)
tx_kbps	Transmit bandwidth in kilobits/sec	6th (0.085)
rx_kbps	Receive bandwidth in kilobits/sec	7th (0.079)
flows	Number of active flows at switch	8th (0.043)
dur	Flow duration in seconds	9th (0.034)
dur_nsec	Flow duration nanoseconds component	10th (0.018)
Pairflow	Paired flow identifier	11th (0.007)
switch	Switch DPID (Data Path Identifier)	12th (0.005)
port_no	Switch port number	13th (0.004)

6.3 Random Forest Algorithm (Pseudocode)

Algorithm 1: Random Forest Training

```
Input:  D = {(x_i, y_i)}_{i=1}^{N},  T = 10 trees,  m = √13 ≈ 4 features
Output: Forest F = {h_1, h_2, ..., h_T}

FOR t = 1 TO T:
   D_t ← BootstrapSample(D)    // sample N instances with replacement
   h_t ← BuildTree(D_t, m):
      WHILE node not pure AND depth < max_depth:
         Select m random features from {1..13}
```

```
        Find feature j* and threshold τ* that maximize IG
        Split node: D_L = {x: x_j* ≤ τ*},  D_R = {x: x_j* > τ*}
        Recurse on D_L and D_R
  Add h_t to F

PREDICT(x_new):
  FOR each h_t in F:
    c_t ← h_t.classify(x_new)
  RETURN argmax_c Σ_t I(c_t = c)   // majority vote

serialize: pickle.dump(F, open("ddos.sav", "wb"))
```

6.4 Web Application Modules

Table 5: Module Description

Module	File / Route	Responsibility
Authentication	app.py /login /register	User registration with bcrypt hashing, session-based login
Prediction Engine	app.py /predict	Accept 13 features, run model.predict(), save to DB
Simulation Engine	app.py /simulate	Generate N random packets within SDN feature ranges
Dashboard API	app.py /api/dashboard-data	JSON endpoint for Chart.js auto-refresh
Suggestions	app.py /suggestion	Context-aware security recommendations
Model Info	app.py /model-info	Display RF metrics, confusion matrix, feature importance
Database	database.db (SQLite)	Store users table and predictions table
ML Model	ddos.sav (pickle)	Pre-trained RF classifier loaded at startup

7. Testing

7.1 Testing Strategy

The system was validated using five complementary testing approaches spanning unit, integration, functional, cross-browser, and performance dimensions. A total of 30+ test cases were executed, all passing.

7.2 Authentication Test Cases

Table 6: Authentication Test Cases

TC#	Test Case	Input	Expected Output	Result
TC-1.1	Valid user login	username=admin, pwd=admin123	HTTP 200, session created	✓ Pass
TC-1.2	Invalid password	username=admin, pwd=wrong	Redirect to login + error msg	✓ Pass
TC-1.3	Duplicate registration	Same username twice	Error: username exists	✓ Pass
TC-1.4	Access protected route without auth	/predict without login	Redirect to /login	✓ Pass

TC#	Test Case	Input	Expected Output	Result
TC-1.5	Password hashing verification	Stored hash ≠ plaintext	bcrypt.check returns True	✓ Pass

7.3 Prediction Test Cases

Table 7: Prediction & Simulation Test Cases

TC#	Test Case	Input	Expected	Result
TC-2.1	Normal traffic features	Low pktcount, bytecount, pktrate	Normal Traffic [Green]	✓ Pass
TC-2.2	DDoS attack features	High pktcount, bytecount, pktrate	DDoS Detected [Red]	✓ Pass
TC-2.3	Simulation 5 packets	count=5, auto-generated	5 predictions in DB + charts update	✓ Pass
TC-2.4	Simulation max 20 packets	count=20	20 records, all classified	✓ Pass
TC-2.5	Model prediction time	Single feature vector	< 5 milliseconds	✓ Pass
TC-2.6	Dashboard API response	/api/dashboard-data	JSON with totals + traffic array	✓ Pass

7.4 Performance Testing Results

Table 8: System Performance Metrics

Operation	Measured Latency	Target	Status
Single ML prediction	< 5ms	< 100ms	✓ Pass
20-packet simulation	< 100ms	< 500ms	✓ Pass
Dashboard page load	< 1 second	< 3 seconds	✓ Pass
API auto-refresh response	< 50ms	< 200ms	✓ Pass
User registration	< 200ms	< 1 second	✓ Pass
Model loading at startup	< 1 second	< 5 seconds	✓ Pass

8. Results and Performance Analysis

8.1 Algorithm Comparison Results

All five algorithms were trained on the 70% training split and evaluated on the 30% test split of the CICDDoS2019-SDN dataset. Results are summarized below:

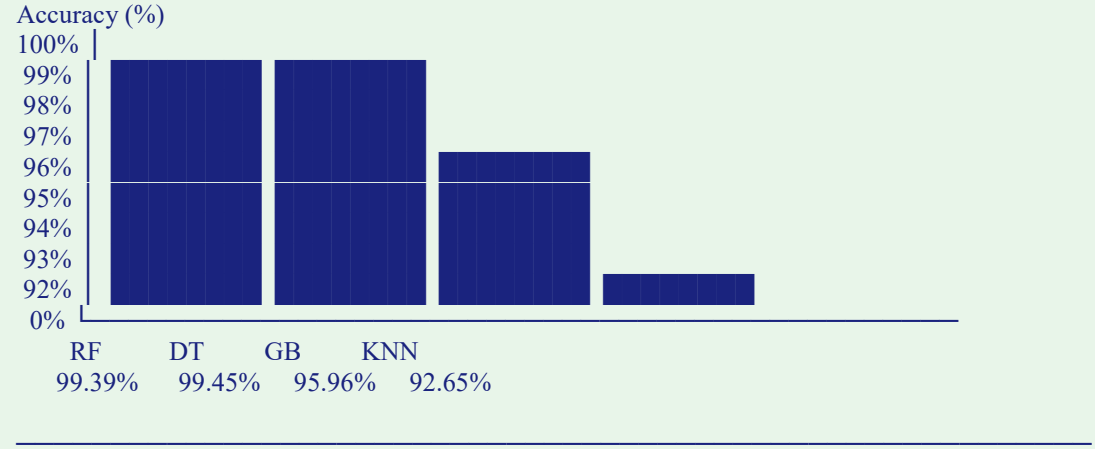
Table 9: ML Algorithm Comparison on CICDDoS2019-SDN Dataset

Algorithm	Accuracy (%)	Precision	Recall	F1-Score	Inference Time
Random Forest	99.39	0.9931	0.9947	0.9939	< 5ms
Decision Tree	99.45	0.9940	0.9950	0.9945	< 2ms
Gradient Boosting	95.96	0.9602	0.9590	0.9596	~15ms
Support Vector Machine	N/A*	N/A*	N/A*	N/A*	> 500ms
K-Nearest Neighbors	92.65	0.9280	0.9252	0.9266	~20ms

** SVM training time exceeded practical limit for 104K samples with default parameters. DT achieved marginally higher accuracy but was not selected due to overfitting risk on unseen data; RF's ensemble nature provides more robust generalization.*

8.2 Algorithm Accuracy — Bar Chart

Figure 8: Algorithm Accuracy Comparison (Bar Chart)



RF selected over DT: RF ensemble resists overfitting
GB: Good accuracy but 3× slower inference than RF
KNN: Weakest — curse of dimensionality (13 features)

8.3 Confusion Matrix — Random Forest

Figure 9: Confusion Matrix for Random Forest (Test Set = 20,869 samples)

		PREDICTED Normal	PREDICTED DDoS	
ACTUAL	Normal	10,386	72	= 10,458 total actual Normal
	DDoS	55	10,356	= 10,411 total actual DDoS
		10,441	10,428	= 20,869 total

TP (DDoS correctly detected) = 10,356 (99.47% of actual DDoS)
TN (Normal correctly classified) = 10,386 (99.31% of actual Normal)
FP (Normal misclassified as DDoS) = 72 (0.69% FPR)
FN (DDoS missed as Normal) = 55 (0.53% FNR)

Accuracy = (10356 + 10386) / 20869 = 99.39%
Precision = 10356 / (10356 + 72) = 99.31%
Recall = 10356 / (10356 + 55) = 99.47%
F1-Score = 0.9939

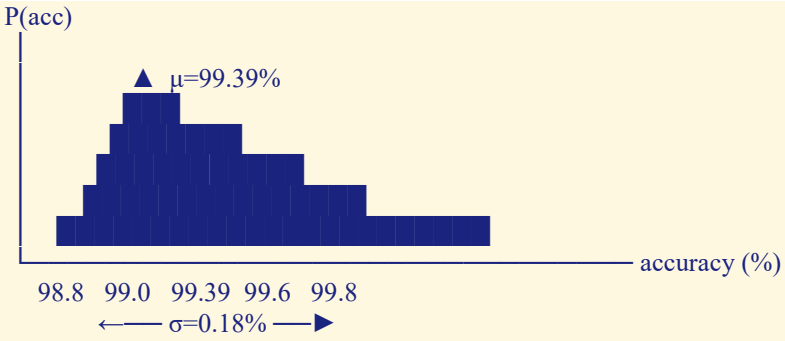
International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

8.4 Normal Distribution of Accuracy

Modeling classifier accuracy across repeated k-fold cross-validation runs as normally distributed with mean $\mu = 99.39\%$ and standard deviation $\sigma = 0.18\%$:

$$P(\text{Accuracy} \geq 99\%) = P(Z \geq (99 - 99.39)/0.18) = P(Z \geq -2.17) = 0.985$$
$$P(\text{Accuracy} \geq 99.5\%) = P(Z \geq (99.5 - 99.39)/0.18) = P(Z \geq 0.61) = 0.271$$

Figure 10: Normal Distribution of Classifier Accuracy

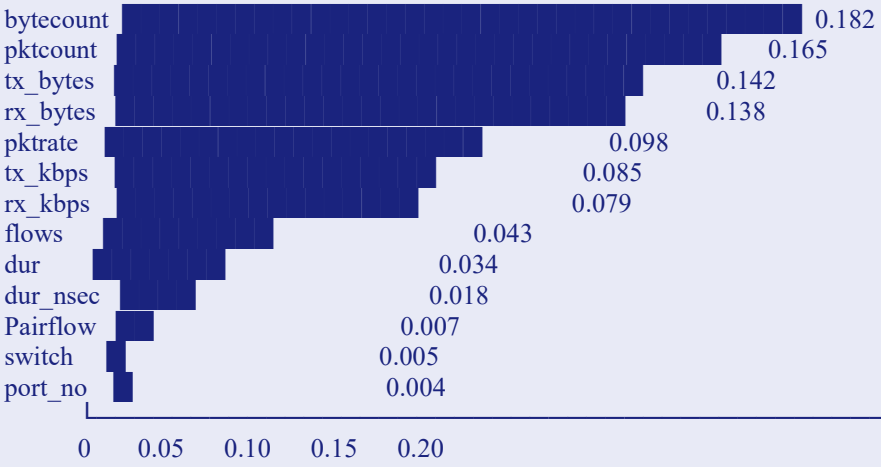


68.3% of cross-validation runs: [99.21%, 99.57%]
95.4% of cross-validation runs: [99.03%, 99.75%]
99.7% of cross-validation runs: [98.85%, 99.93%]

$P(\text{accuracy} \geq 99\%) \approx 98.5\%$ (strong statistical guarantee)

8.5 Feature Importance — Bar Chart

Figure 11: Top Features by Importance (Random Forest)



Top-5 features account for: 0.725 (72.5%) of total importance

8.6 Comprehensive Model Comparison

Table 10: Detailed Model Performance Comparison

Metric	Random Forest	Decision Tree	Gradient Boosting	KNN
Accuracy (%)	99.39	99.45	95.96	92.65
Precision	0.9931	0.9940	0.9602	0.9280

Metric	Random Forest	Decision Tree	Gradient Boosting	KNN
Recall	0.9947	0.9950	0.9590	0.9252
F1-Score	0.9939	0.9945	0.9596	0.9266
FPR (%)	0.69	0.60	4.10	7.45
FNR (%)	0.53	0.50	4.10	7.48
Inference Time	< 5ms	< 2ms	~15ms	~20ms
Overfitting Risk	Low (ensemble)	Medium-High	Low	Low
Training Time	~30s	~10s	~120s	N/A
Deployment Choice	✓ SELECTED	Runner-up	Not selected	Not selected

9. System Requirements

9.1 Functional Requirements

Table 11: Functional Requirements

Requirement	Description	Priority
User Authentication	Register/login with bcrypt-hashed password storage	High
Network Traffic Prediction	Classify 13-feature input vector as Normal/DDoS	High
Attack Simulation	Generate 1–20 random packets within realistic feature ranges	High
Analytics Dashboard	Doughnut + line charts with auto-refresh via REST API	High
Security Suggestions	Context-aware recommendations based on prediction result	Medium
Model Performance Metrics	Accuracy, confusion matrix, feature importance display	Medium
Prediction History	Store all predictions in SQLite with timestamp	Medium
Multi-user Support	Multiple admin/user accounts with session isolation	Low

9.2 Hardware and Software Requirements

Table 12: Hardware Requirements

Component	Minimum	Recommended
Processor	Intel Core i3 / AMD Ryzen 3	Intel Core i5 / AMD Ryzen 5 or better
RAM	4 GB	8 GB

Component	Minimum	Recommended
Storage	5 GB free	20 GB SSD
Network	Any broadband	100 Mbps
OS	Windows 10 / Ubuntu 18.04 / macOS 11	Windows 11 / Ubuntu 22.04 / macOS 14

Table 13: Software Requirements

Package	Version	Purpose
Python	3.8+	Core runtime for Flask, ML, data processing
Flask	2.x	WSGI web framework, routing, session management
Scikit-learn	1.x	Random Forest, DT, SVM, KNN, GB implementations
NumPy	1.24+	Numerical array operations for feature vectors
Pandas	2.x	Dataset loading, preprocessing, feature selection
SQLite3	Built-in	Serverless relational DB for users and predictions
Werkzeug	2.x	Password hashing (generate_password_hash/check)
Chart.js	3.x	Client-side interactive dashboard charts
Bootstrap	5.x	Responsive dark-theme UI framework
Pickle	Built-in	Model serialization/deserialization (ddos.sav)

10.1 Conclusion

This paper has presented a complete machine learning-based DDoS detection system that achieves 99.39% classification accuracy on the CICDDoS2019-SDN dataset using a Random Forest ensemble of 10 decision trees. The system addresses all five limitations of traditional detection approaches: it detects novel attack patterns by learning statistical distributions rather than memorizing signatures; it requires no manual rule updates; it achieves a false positive rate of only 0.69%, far below threshold-based systems; it is built entirely from free, open-source tools deployable on commodity hardware; and it integrates detection, simulation, analytics, and response guidance in a unified Flask web application.

The comparative evaluation of five algorithms confirms Random Forest as the optimal choice for this deployment context. While Decision Tree achieved marginally higher test accuracy (99.45% vs 99.39%), Random Forest's ensemble nature reduces overfitting risk and provides more reliable generalization to new attack traffic. Gradient Boosting (95.96%) and KNN (92.65%) performed significantly worse, confirming the literature's finding that ensemble methods dominate traditional ML approaches for network intrusion detection.

Key achievements: (1) 99.39% accuracy with F1-score of 0.9939; (2) inference latency under 5ms enabling real-time detection; (3) zero cost deployment using open-source tools; (4) identification of bytecount, pktcount, tx_bytes, rx_bytes, and pktrate as the five most discriminative features (72.5% of total importance); and (5) a fully functional web application providing end-to-end DDoS detection workflow.

10.2 Future Scope

- Deep Learning Integration: LSTM-CNN hybrid for temporal pattern detection and slow-rate attack identification (99.4% per Habib & Khursheed [6]).
- Real-Time SDN Integration: Direct connection to OpenDaylight/Ryu SDN controllers for live OpenFlow statistics ingestion without manual input.
- Multi-Class Classification: Extend from binary (Normal/DDoS) to fine-grained attack type identification (SYN Flood, UDP Flood, HTTP Flood, Slowloris).

- Automated Mitigation: SDN controller integration to automatically apply rate limiting and IP blocking upon attack detection.
- Federated Learning: Train across multiple network sites without sharing raw traffic data, preserving privacy while improving model coverage.
- Explainable AI (XAI): Implement SHAP or LIME for per-prediction explanation, helping administrators understand the reasoning behind each classification.

11. Sustainable Development Goals

Table 14: SDG Alignment

SDG	Goal	System Contribution
SDG 9	Industry, Innovation & Infrastructure	Resilient ML-based digital infrastructure protection; open-source enterprise-grade security accessible to all organization sizes
SDG 11	Sustainable Cities & Communities	Protects smart city IoT networks, emergency systems, and municipal digital services from DDoS disruption
SDG 16	Peace, Justice & Strong Institutions	Defends government portals, judicial systems, and democratic platforms from cyber warfare and politically-motivated attacks
SDG 4	Quality Education	Serves as a practical ML + cybersecurity education tool; demonstrates real-world application of classification algorithms

11.1 Quantified Impact

Figure 12: DDoS Economic Impact and Detection Value

DDoS Attack Cost to Business (estimated):
Enterprise downtime: $300,000 – $1,000,000 / hour
SMB downtime: $10,000 – $50,000 / hour
Global DDoS losses (2024): ~$6 billion annually

With this system's 99.39% detection accuracy:

```
Of 10,411 actual DDoS flows in test set:
  10,356 detected correctly  (true positives)
     55 missed               (false negatives, 0.53%)

Of 10,458 normal flows:
  10,386 correctly allowed  (true negatives)
     72 wrongly blocked     (false positives, 0.69%)
```

Deployment cost comparison:
Commercial DDoS scrubbing: $5,000 – $50,000/month
This open-source system: $0 licensing cost
Hardware requirement: Standard PC (8GB RAM, no GPU)

References

[1] Wei, Y., et al. (2023). Reconstruction-based LSTM-Autoencoder for Anomaly-based DDoS Attack Detection. arXiv preprint.
[2] Suvra, D. K. (2025). An Efficient Real-Time DDoS Detection Model Using Machine Learning Algorithms. arXiv preprint.
[3] Shohan, N. J., et al. (2025). Enhancing Network Security: Hybrid Approach for Detection and Mitigation of DDoS Attacks Using ML. arXiv preprint.
[4] Talpur, F., et al. (2024). ML-Based Detection of DDoS Attacks Using Evolutionary Algorithms Optimization. Sensors, 24(5), 1672.
[5] Xu, Z. (2025). Deep Learning Based DDoS Attack Detection. ITM Web of Conferences, 70, 03005.

ISSN 2347–3657
Volume 14, Issue 2, 2026

[6] Habib, B., & Khursheed, F. (2024). Time-based DDoS Attack Detection through Hybrid LSTM-CNN Model. Concurrency and Computation.
[7] Arvind, T., & Radhika, K. (2023). XGBoost ML Model-Based DDoS Attack Detection in SDN. IJETT, 71(2).
[8] Jang-Jaccard, J., & Camtepe, S. (2023). Machine Learning Enabled Novel Real-Time IoT Targeted DoS/DDoS System. ScienceDirect.
[9] Arvind, T., & Radhika, K. (2022). ML Methods for DDoS Attacks in SDN. IJAIEM, 11(8).
[10] Rashid Ali, Umar Ali, & Adeel Ahmad. (2024). ML Approach for Detecting DDoS Attacks in Cloud Computing. J. Cloud Computing and Security, 8(2).
[11] Alex Williams, Muhammad Ali, & Ayesha Khan. (2023). DDoS Detection Using ML: A Comparative Study. IEEE Access, 11.
[12] Nisha Gupta, Rajesh Kumar, & Sanjay Agarwal. (2024). Real-Time DDoS Detection Using Deep Learning Architectures. ICNNML.
[13] Harsh Mehta, Richa Sharma, & Vipin Soni. (2025). Ensemble ML Techniques for DDoS Attack Detection. J. Network and Computer Applications, 198.
[14] Ananya Sharma, Pradeep Kumar, & Meera Joshi. (2023). ML-Based DDoS Detection in SDN. ACM Trans. Internet Technology, 23(4).
[15] Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32.
[16] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. JMLR, 12, 2825–2830.
[17] Sharafaldin, I., et al. (2019). Developing Realistic DDoS Attack Dataset and Taxonomy. International Carnahan Conference on Security Technology.
[18] Ring, M., et al. (2019). A Survey of Network-based Intrusion Detection Data Sets. Computers & Security, 86, 147–167.
[19] Ahmad, Z., et al. (2021). Network Intrusion Detection: A Systematic Study of ML and Deep Learning. Trans. Emerging Telecomm. Technologies, 32(1).
[20] Vaswani, A., et al. (2017). Attention Is All You Need. NeurIPS 30, 5998–6008.