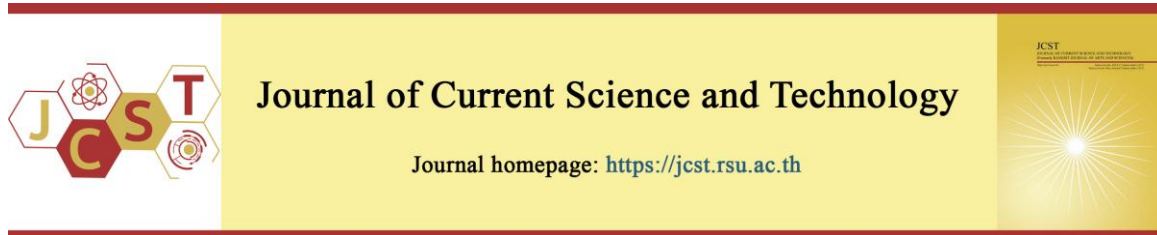


Cite this article: Sathaporn, P., Krungseanmuang, W., Chaowalittawin, V., Benjangkaprasert, C., Archevapanich, T., & Purahong, B. (2026). DDoS Detection Framework Using Machine Learning Optimized by Bayesian and PSO Techniques. *Journal of Current Science and Technology*, 16(3), Article 204. <https://doi.org/10.59796/jcst.V16N3.2026.204>



DDoS Detection Framework Using Machine Learning Optimized by Bayesian and PSO Techniques

Posathip Sathaporn¹, ORCID NO: 0009-0008-0329-1276
Woranidtha Krungseanmuang², ORCID NO: 0009-0008-1601-5019
Vasutorn Chaowalittawin¹, ORCID NO: 0009-0006-1021-9438
Chawalit Benjangkaprasert³, ORCID NO: 0009-0006-5832-2450
Tuanjai Archevapanich⁴, ORCID NO: 0000-0001-5020-6697
Boonchana Purahong^{3,*}, ORCID NO: 0009-0008-1172-1870

¹Department of Robotics and AI, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

²Department of Biosystems and Agricultural Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

³Department of IoT and Information Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

⁴Department of Electronics and Communication Engineering, Faculty of Engineering and Architecture, Rajamangala University of Technology Suvarnabhumi, Nonthaburi, 11000 Thailand

*Corresponding author; E-mail: boonchana.pu@kmitl.ac.th

Received 29 January 2026; Revised 2 April 2026; Accepted 30 April 2026; Published 25 June 2026

Abstract

This paper presents a distributed denial of service (DDoS) detection framework using machine learning techniques enhanced with hyperparameter optimization for network traffic classification and evaluated on the BCCC-cPacket-Cloud-DDoS-2024 dataset. The framework includes data preprocessing with normalization and class imbalance handling via the synthetic minority over-sampling technique. A critical contribution of this study is the rigorous analysis of the trade-off between detection accuracy and model complexity. Unlike arbitrary feature selection methods, we empirically determined the optimal feature set using information gain, identifying that the top 100 features represent the saturation point that balances high accuracy with minimal overhead. Model performance was further improved through hyperparameter optimization using particle swarm optimization and Bayesian algorithms. The extreme gradient boosting (XGBoost) model optimized using Bayesian optimization and the top 100 features achieved the highest performance, with an accuracy of 99.29% and an F1-score of 98.91%. As a result, the proposed framework improves detection performance while reducing model complexity by selecting an optimal feature set to improve model stability and efficiency.

Keywords: DDoS detection framework, feature selection, hyperparameter optimization, machine learning, XGBoost

1. Introduction

Internet and data communication are among the most crucial components affecting our lives. They influence many aspects of daily life, including industry, banking, transportation, and telecommunications. These

information technologies offer many advantages, such as helping people work faster, improving communication, and making information easier to access. They also support businesses by reducing errors, saving time, and enabling better decision-

making. As the world becomes more connected, the role of the internet and data communication continues to grow and becomes increasingly necessary in everyday life.

As technology, data transmission, and network infrastructure continue to grow, the security of these systems has become increasingly important. Attackers or malicious users who attempt to damage the system or steal information can cause serious problems and compromise data security. To address these risks, one common solution is the intrusion detection system (IDS). This system helps monitor networks, detect abnormal traffic, and generate alerts when a cyberattack occurs so the system can respond and protect itself appropriately.

Distributed denial of service (DDoS) is a type of attack that often targets cloud systems. The core concept of DDoS is that an attacker takes control of many computers, called a botnet, and uses them to send a large volume of traffic to a server. When excessive traffic arrives simultaneously, the system experiences resource exhaustion, such as high CPU usage. By keeping connections open for extended periods, attackers drain the server's computational resources. Consequently, legitimate users are unable to access the service, as shown in Figure 1.

For businesses, a DDoS attack can cause systemic paralysis. If a service becomes unavailable, users cannot access important information and it

immediately affects the organization's perceived reliability. Recent reports indicate that DDoS attacks are occurring more frequently. In 2025, Yoachimik and Pacheco (2025) reported that, during the second quarter, Cloudflare mitigated more than six thousand large-scale attacks, well over seventy per day, and that the total number of attack cases reached over 27 million, exceeding the total recorded in the previous year.

Machine learning (ML) is a branch of artificial intelligence that enables a system to learn how to perform certain tasks based on previous experience. Over the years, large volumes of network traffic logs have been collected, including both normal traffic and traffic showing signs of attacks. These logs are usually stored as structured data, making it possible to analyze and identify patterns that may indicate abnormal or harmful activity. ML algorithms can be used to predict or estimate future attacks. For network traffic anomaly detection applications where there are many input features and a clear target to predict, supervised learning is often the most suitable approach. The model learns from past data that already have labels. Common algorithms used in this area include decision tree classifiers (DT), logistic regression (LR), k-nearest neighbors (KNN), random forest (RF), categorical boosting (CatBoost), and extreme gradient boosting (XGBoost).

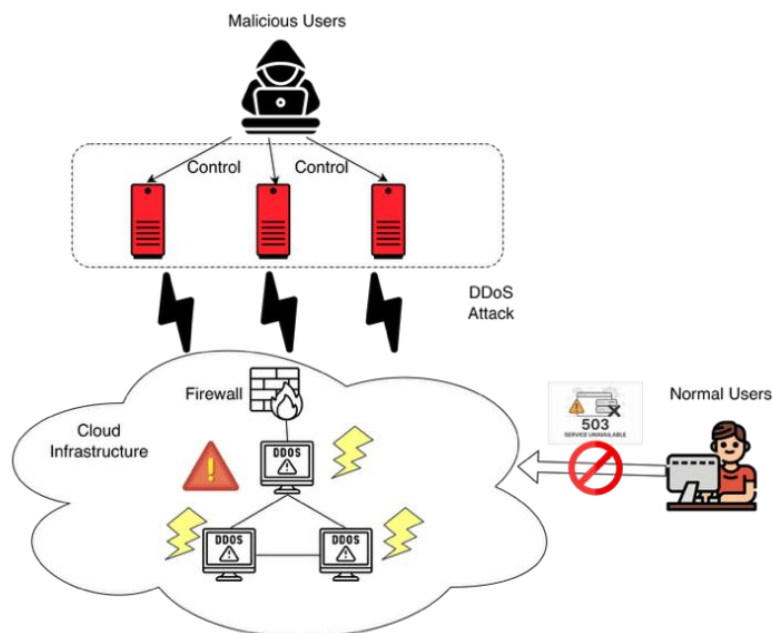


Figure 1 DDoS attacks on Cloud Infrastructure

There are also methods that combine several algorithms to improve accuracy and stability, known as ensemble learning, such as random forest and XGBoost, which are widely used for attack detection because they usually provide strong and reliable results. These algorithms help the system identify unusual behavior more quickly and reduce the likelihood of attacks that could damage network infrastructure.

Supervised machine learning has several advantages compared with deep learning or unsupervised methods, especially when working with network traffic. Since most network datasets contain clear labels, supervised models can separate normal and attack behavior more effectively. Additionally, machine learning models use fewer resources and run faster than deep learning models, which makes ML more suitable for near-real-time detection in edge or cloud computing.

Compared with unsupervised methods, supervised models usually provide higher accuracy and more stable results. They also reduce false alarms and are easier to interpret, which is important for real-world network security, where quick and reliable decisions are needed.

Hyperparameters are essentially the key configuration settings that guide how a model learns during the training phase. When these parameters are set appropriately, the model can recognize patterns more efficiently and interpret the data with greater accuracy. In contrast, poorly chosen hyperparameters can cause the model to perform poorly, leading to weak pattern recognition and unreliable detection, especially in tasks involving network traffic. Proper tuning is crucial for achieving a model that is both stable and accurate.

To identify the most effective hyperparameter values, we applied automated optimization techniques, such as particle swarm optimization (PSO) and Bayesian optimization. PSO operates through a cooperative search process in which multiple agents explore the solution space and share information to gradually move toward an optimal configuration. Bayesian optimization takes a more analytical approach by learning from each evaluation and using that knowledge to make more informed predictions about the next set of parameters to test. These automated methods significantly reduce manual error, improve training efficiency, and consistently lead to better-performing models.

In recent years, many studies have focused on DDoS attack detection in modern networks. They use

different machine learning models, neural network architectures, and tuning techniques to improve performance. Most studies also include preprocessing steps, such as cleaning traffic data, extracting and selecting features, and normalizing values to make the models learn more effectively. In addition, these studies rely on well-known datasets that collect real attack traffic for training and testing.

Many datasets for DDoS detection have been collected and published by researchers. Sharafaldin et al. (2018) generated a new IDS dataset namely CICIDS2017, which covers all eleven necessary criteria and includes common updated attacks, such as DDoS, brute force, structured query language injection, infiltration, port scan, and botnet attacks. Sharafaldin et al. (2019) introduced CICDDoS2019, a dataset containing 11 types of DDoS attacks designed for evaluating IDS performance. In addition, Shafi et al. (2024) conducted a comprehensive review of state-of-the-art DDoS datasets, assessing 16 publicly available datasets and highlighting 15 major limitations from multiple perspectives. They further proposed a new cloud-based dataset named BCCC-cPacket-Cloud-DDoS-2024.

Based on these datasets, researchers have applied traditional ML classifiers, deep learning (DL) architectures, and hybrid models to learn patterns from network traffic and distinguish malicious behaviors from normal activities. Chen et al. (2025) proposed the use of adaptive grid search with cross-validation (GridSearchCV) to address the latency and efficiency limitations of traditional GridSearchCV. Dasari and Kaluri (2024) presented a hierarchical ML framework that classifies DDoS attacks in progressive stages. The model integrates hyperparameter optimization to boost accuracy and reduce false positives. Sanchez et al. (2021) evaluated multiple ML algorithms for DDoS detection under a unified experimental setup. They applied grid search hyperparameter optimization to measure how tuning impacts the model.

Saghezchi et al. (2022) focused on detecting DDoS attacks in Industry 4.0 cyber-physical production systems (CPPSSs) using machine learning. The authors evaluated several ML models on traffic patterns specific to industrial environments. Cil et al. (2021) developed a feedforward deep neural network (DNN) to detect DDoS attacks with high accuracy, showing that deep architectures can improve robustness and precision in modern DDoS detection systems. Saini et al. (2020) compared multiple machine learning algorithms, including support vector

machine, RF, and KNN for detecting DDoS attacks using network traffic features and identifying the algorithms that achieve the best accuracy. Santos et al. (2020) proposed ML-based DDoS detection specifically designed for software-defined network environments. The system extracts OpenFlow-based features and evaluates models such as DT, RF, and multilayer perceptron.

Al-Shareeda et al. (2023) compared a wide range of machine learning and deep learning models for DDoS detection on CICIDS2017 and CICDDoS2019. They concluded that DL achieves higher accuracy, while ML is lighter and more practical for deployment. Mohmand et al. (2022) developed a hybrid ML pipeline for classifying and predicting DDoS attack patterns. Their approach uses feature engineering and ensemble learning to improve detection performance. Vinayakumar et al. (2019) introduced a deep learning-driven IDS using stacked autoencoders and deep neural networks and achieved strong detection performance across multiple benchmark intrusion datasets. Indrasiri et al. (2022) proposed a stacked ensemble classifier to detect malicious traffic in IoT environments. Their method integrates multiple base learners with a meta-classifier to boost accuracy.

Awan et al. (2021) developed a real-time DDoS detection system built on a big-data analytics framework. They leveraged distributed processing on Hadoop to handle high-volume network traffic efficiently. Bhattacharya et al. (2020) presented a hybrid principal component analysis (PCA), firefly algorithm, and XGBoost intrusion detection model, with PCA used to reduce dimensionality. Sathaporn et al. (2025) proposed a hybrid convolutional neural network (CNN) and recurrent neural network (RNN) architecture enhanced with multi-head attention for cloud-based DDoS detection. The results demonstrated significant accuracy gains and superior performance on cloud infrastructure traffic.

Rai and Mandoria (2019) evaluated several ML algorithms for network intrusion detection. The authors compared accuracy, processing speed, and suitability across multiple attack categories. Becerra-Suarez et al. (2024) presented the results of a comparative analysis of six machine learning techniques, namely RF, DT, AdaBoost, XGBoost, multilayer perceptron (MLP), and deep neural network (DNN), for classifying DDoS attacks. Sadhwani et al. (2023) proposed a lightweight IDS that combines machine learning with efficient preprocessing, including missing value handling, standardization, and feature

selection using an Extremely Randomized Trees classifier. Liu et al. (2023) proposed a DDoS detection method based on feature engineering and machine learning for software-defined networks, aiming to improve detection accuracy and efficiency. Ahmad et al. (2023) proposed an optimized hybrid deep intrusion detection model to enhance network security, combining multiple deep learning techniques for better performance.

Based on the existing literature, several critical research gaps remain. First, most existing frameworks rely on outdated benchmark datasets that fail to reflect the complexity of modern cloud-based traffic. Second, previous studies often arbitrarily select feature subsets without a rigorous analysis of the trade-off between model complexity and detection accuracy. Finally, direct comparative studies between probabilistic Bayesian optimization and heuristic PSO methods for fine-tuning XGBoost in high-dimensional network data are largely lacking. Therefore, to address these limitations, we present the following main contributions:

- We evaluate our framework on the modern cloud-traffic BCCC-cPacket-Cloud-DDoS-2024 dataset. This ensures that our detection model is robust against current TCP-based DDoS attack patterns, specifically within cloud infrastructures.
- We perform a rigorous trade-off analysis using information gain. We analytically identify that the top 100 features represent the stability saturation point, maximizing the F1-score while significantly reducing computational overhead for real-time detection.
- We provide a comprehensive benchmark of probabilistic Bayesian optimization and various heuristic techniques, including particle swarm optimization (PSO), genetic algorithm (GA), and simulated annealing (SA). Our results demonstrate that Bayesian optimization yields superior stability.

The remainder of this paper is structured as follows. Section 2 presents the objective of this study. Section 3 presents the workflow, preprocessing techniques, ML model, and hyperparameter optimization. Section 4 presents the experimental results. Section 5 discusses the results. Section 6 presents the conclusions and future work.

2. Objectives

The objective of this study is to develop an effective DDoS detection framework using machine learning techniques. The framework aims to accurately classify network traffic by selecting the

best-performing models from various machine learning and ensemble algorithms, integrating feature selection to balance detection accuracy with computational efficiency, and precisely optimizing model hyperparameters through the comparative application of particle swarm optimization (PSO) and Bayesian algorithms.

3. Material and Methods

3.1 Workflow of Experiment

In this experiment, we optimize machine learning models for DDoS attack detection by proposing a workflow consisting of model selection, feature selection, and hyperparameter optimization to achieve the best performance. The experimental workflow is shown in Figure 2.

First, during data preprocessing, we removed records with null values to ensure data integrity. Next, the dataset was split into an 80% training set and a 20% testing set using stratified sampling to preserve the class distribution in both subsets to prevent biased evaluation. The data were then normalized to a unified scale to prevent feature dominance and improve model stability. The selected DDoS datasets have a significant class imbalance between benign and attack traffic. Therefore, we employed synthetic minority over-sampling technique (SMOTE) to generate synthetic minority samples and improve class distribution, thereby enhancing the reliability of the learning process. Importantly, SMOTE was applied only to the training set to ensure the test data remained original. Second, we perform baseline model selection by evaluating various machine learning algorithms, including LR (Cox, 1958), KNN (Cover & Hart, 1967), DT (Breiman et al., 2017), CatBoost (Prokhorenkova et al., 2018), RF (Breiman, 2001), and XGBoost. The objective of this process is to identify the most suitable algorithm by comparing their performance in terms of accuracy and F1-score. This comparison identifies the best model architecture with sufficient

efficiency to capture the patterns of DDoS attack behavior. After model selection, we determined the optimal number of features using an information gain-based ranking method combined with cross-validated performance evaluation. The feature selection process helps retain only the most informative attributes while reducing redundancy and computational overhead.

Third, we utilized hyperparameter optimization using two optimization techniques: Bayesian optimization and PSO. These methods explore the search space more efficiently than manual tuning, allowing the model to reach its optimal configuration in terms of accuracy, F1-score, and generalization capability.

Finally, we evaluated the performance of the baseline model and the fully optimized model, including feature selection and hyperparameter tuning. Based on this comparison, each stage of the workflow helps improve the accuracy and stability of DDoS detection, ensuring robust detection coverage across all attack categories included in the training dataset.

3.2 Dataset

In this study, the BCCC-cPacket-Cloud-DDoS-2024 dataset, which is part of the cybersecurity datasets released by the Behaviour-Centric Cybersecurity Center (BCCC) at York University, was used as the main source of traffic data. This dataset is publicly available and was collected from a real cloud computing environment over a five-day period. It focuses on TCP-based DDoS attacks and was developed to address several issues commonly found in older benchmark datasets. Many traditional datasets contain synthetic or generated traffic in laboratory environments or include only a narrow range of attack types. In contrast, BCCC-cPacket-Cloud-DDoS-2024 provides traffic that reflects conditions closer to a modern cloud infrastructure and offers broader attack diversity through 17 distinct DDoS scenarios.

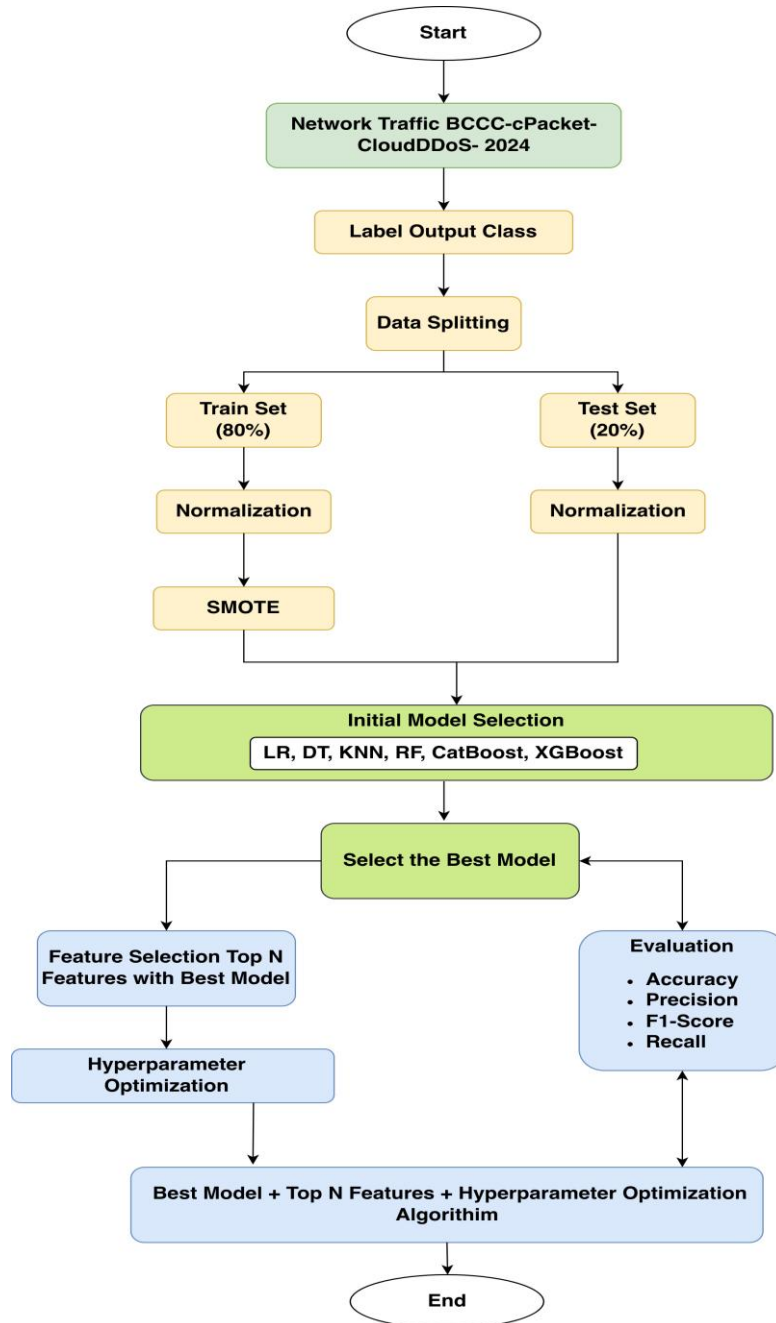


Figure 2 Experiment Workflow

One of the strong advantages of this dataset is the level of detail included in the data. Each record contains 326 features, allowing models to capture subtle traffic patterns that older datasets often fail to detect. The dataset is also relatively large, consisting of 641,668 traffic records, which helps reduce bias during model training and supports more reliable validation across multiple learning algorithms.

The dataset is organized into two traffic categories, as shown in Figure 3, consisting of 413,199 benign traffic samples and 228,469 attack traffic samples. This structure makes it suitable for testing both binary and multiclass detection approaches.

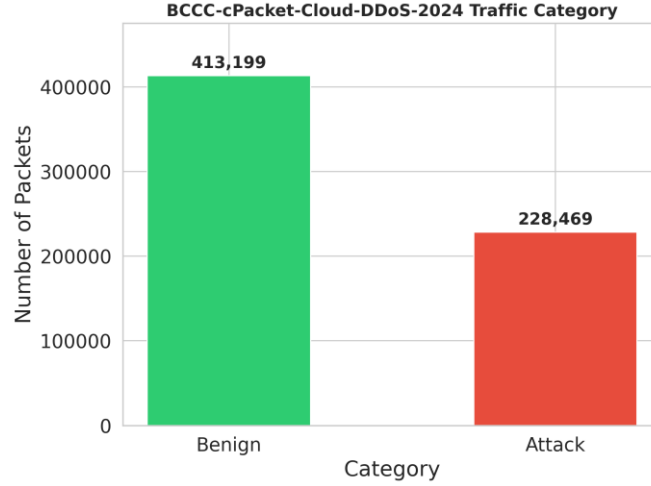


Figure 3 Class Distribution in the BCCC-cPacket-Cloud-DDoS-2024 Dataset

3.3 Data Preprocessing

3.3.1 Data Cleansing, Normalization and SMOTE

During the data preprocessing stage, the dataset was cleaned to make the data suitable for model training. This step was necessary because structured data often contain missing or empty fields. We cleaned the data by deleting records with empty values to enable complete model execution. Then, the categorical output label classes were encoded into numerical classes, with 0 representing the benign class and 1 representing the attack class.

After the initial data cleansing, the dataset was partitioned into training and testing sets using a stratified random split technique with an 80:20 ratio. This approach ensures that the class distribution between benign and attack traffic remains consistent in both subsets. A fixed random seed set to 42 was utilized to ensure the reproducibility of the experiments and to verify that the split was not time-dependent. Crucially, this data splitting was performed prior to normalization and SMOTE to prevent data leakage.

Next, we applied normalization (Krishna & Sahu, 2015) to scale the features to a similar range. Some columns had small values, while others were much larger values, and placing the data on the same scale makes the training smoother and prevents any single feature from dominating simply because it has larger values.

One major issue in intrusion detection datasets is class imbalance, where normal traffic records usually outnumber attack traffic by a large margin. To handle this issue, the SMOTE (Chawla et al., 2002) method was applied. SMOTE interpolates between

existing minority-class points to generate new synthetic samples rather than simply copying minority samples. This gives the model more balanced data to learn from and usually leads to better recognition of attack patterns. Importantly, this oversampling technique was applied exclusively to the training set after the data splitting process to prevent data leakage. The test set remained unmodified, ensuring that model evaluation was conducted on original, real-world traffic patterns.

3.3.2 XGBoost Model

As detailed in the experimental evaluation, XGBoost achieved the highest baseline performance. Therefore, its underlying mathematical formulation is explicitly described in this section.

XGBoost (Chen & Guestrin, 2016) is a boosting method that builds several decision trees one step at a time. Each new tree is added to correct the mistakes made by the previous ones, so the model gradually becomes more accurate without making the structure unnecessarily complicated. The core idea is to reduce the overall error as much as possible at every stage. The process is performed by optimizing the objective function, as shown in Equation 1.

$$\text{Obj} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \quad (1)$$

where y_i is the true label of sample i , $\hat{y}_i$ is the predicted value, $l(y_i, \hat{y}_i)$ is the loss between the true and predicted values, f_k is the k -th decision tree, $\Omega(f_k)$ is the regularization term that keeps the tree from becoming too complex, K is the number of trees, and n is the total number of samples.

To decide how each new tree should be built, XGBoost uses the gradient and the second derivative of the loss. These values indicate how much each sample contributes to the error. The score or weight for a node can be calculated using Equation 2.

$$\text{Score} = -\frac{\sum_{i \in I} g_i}{\sum_{i \in I} h_i + \lambda} \quad (2)$$

where g_i is the gradient of the loss for sample i , h_i is the Hessian second derivative of the loss, I is the group of samples that fall into that node, and λ is the regularization term controlling how strong the adjustment should be.

The algorithm looks at which parts of the data the current model handles poorly, then grows new trees to fix those specific areas. By doing this repeatedly, the model improves step by step while still being kept under control to avoid overfitting.

3.3.3 Feature Selection

Feature selection with information gain is a simple way to keep only the attributes that help the model predict the output. The idea is based on entropy; if a feature reduces uncertainty about the final label, then that feature is considered important. In contrast, features that do not substantially change the uncertainty are treated as less useful and can be removed. This helps reduce unnecessary computation and prevents the model from learning noise. The information gain between a feature X and the output label Y can be expressed as Equation 3. A higher value of information gain means that the feature carries more information about the output and is therefore more valuable for model training.

$$I(X;Y) = p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \quad (3)$$

where X is the feature input, Y is the output class, $p(x,y)$ is the joint probability of feature input and output, $p(x)$ is the probability of the feature x , $p(y)$ is the probability of the output y .

In our experiment, we calculated the information gain for all 326 features, and the features were ranked from the most to the least informative. To understand how the number of selected features affects the classifier, different values of n were tested, starting from very small subsets up to all 326 features. The number of top features was evaluated by training the model and measuring its performance, allowing us to

identify the optimal number of features that yields the highest accuracy and F1-score. This process ensures that the model learns only from features that truly help with prediction, reduces redundant data, speeds up computation, and stabilizes final performance.

3.4 Hyperparameter Optimization

After selecting the features and deciding which machine learning model to implement, the next step is to adjust the hyperparameters. These are fixed settings that guide how the model learns, such as the learning rate, tree depth, number of trees, or regularization strength. Because these values can substantially affect model performance, they must be chosen carefully. A simple and efficient search method helps identify a good setup without using excessive computing time. In this work, two optimization methods were used: PSO and Bayesian optimization. Both approaches aim to search for a hyperparameter set that provides better accuracy and stability while reducing manual error.

In this study, both optimization methods were applied to optimize the same set of five hyperparameters. The stopping criteria for both algorithms were defined by a fixed maximum number of iterations. In our experiment, we set the maximum value to 50 iterations. Additionally, for PSO, the initial positions of the particles were generated randomly within the predefined boundaries of the search space.

3.4.1 Particle Swarm Optimization

Particle swarm optimization (Kennedy & Eberhart, 1995) is a search method used to adjust hyperparameters without relying on gradients. In this approach, each candidate hyperparameter set is represented as a particle in a multidimensional search space. Every particle keeps track of two values: its best score obtained so far and the best score reached by any particle in the group.

During the search, the particle positions are updated by combining these two references, allowing the swarm to move toward regions that produce better results. Because PSO evaluates the model directly without requiring strict mathematical conditions, it can cover a large search area and locate suitable hyperparameter settings that may not be found by manual tuning. As illustrated in Table 1, particle updates are guided by the personal best (p_{best}) and the global best (g_{best}), representing the best solutions found by individual particles and the entire swarm, respectively.

Table 1 Pseudocode for PSO

Step	Description
1	Initialize a swarm of particles (p_i) with random hyperparameter sets
2	For each particle (p_i), evaluate the model using its current hyperparameters
3	Set the personal best position (p_{best}) for each particle
4	Set the global best position (g_{best}) from all particles
5	For ($t = 1$) to Max Iterations do
6	For each particle (p_i) do
7	Update velocity (v_i) using (p_{best}) and (g_{best})
8	Update position (x_i) using the new velocity
9	Evaluate the model at the new position (x_i)
10	If performance ((x_i)) better than performance ((p_{best})) then
11	Update ($p_{best} = x_i$)
12	End if
13	End for
14	Update (g_{best}) from all (p_{best}) values
15	End for
16	Return (g_{best}) as the selected hyperparameter set

Table 2 Pseudocode for Bayesian Optimization

Step	Description
1	Define the search space for hyperparameters
2	Initialize a set of random samples and evaluate their model performance
3	Fit a simple surrogate model using evaluated points
4	for $t = 1$ to Max Iterations
5	Use the surrogate model to compute the acquisition score for candidate points
6	Select the next hyperparameter set X_t with the highest acquisition score
7	Evaluate the model using X_t
8	Add X_t and its performance to the history
9	Update the surrogate model with all collected data
10	end for
11	Return the hyperparameter set with the best recorded performance

3.4.2 Bayesian Optimization

Bayesian optimization (Snoek et al., 2012) operates by creating a simple model that estimates which hyperparameter values might yield good results. It does not test many settings at random. Instead, it learns from each run and updates its estimates step by step. In each round, it chooses new values by combining two actions: first testing settings that already appear promising and then exploring new areas that may lead to better results. This balance helps identify good hyperparameters with fewer trials, which is helpful when model training takes a long time, as shown in Table 2.

In this algorithm, the search space for hyperparameters is first defined, and a small set of random samples is evaluated to initialize the surrogate model. At each iteration, the model computes an acquisition score for candidate hyperparameters, selects the one with the highest score, evaluates its performance, and updates the model with all collected

data. This iterative process continues until a maximum number of iterations is reached, after which the hyperparameter set with the best recorded performance is returned.

4. Experiments and Results

4.1 Experiment Setup

In this experiment, several machine learning models were tested and integrated with hyperparameter optimization to identify settings that provide the best accuracy and F1-score for DDoS detection. All experiments were carried out in a standard Google Colab environment using Python 3.12. The workflow steps included data preparation, feature selection, and model tuning. Several libraries were used: NumPy 2.0.2 and pandas 2.2.2 were used for handling and organizing the dataset, while scikit-learn 1.6.1 supported preprocessing tasks and feature selection. Matplotlib 3.10.0 and seaborn 0.13.2 were used to generate plots throughout the analysis. For hyperparameter search, Optuna 4.6.0

handled automatic tuning, and PySwarms 1.3.0 was applied for PSO. Additionally, XGBoost 3.1.2 and CatBoost 1.2.8 were utilized for model training and evaluation in the ensemble learning framework.

Model training and evaluation were executed on a regular Colab runtime with a shared CPU environment using an Intel Xeon-class processor and default RAM allocation. This experiment setup was sufficient for testing multiple algorithms and running the optimization algorithms. The versions of the libraries used in this study are listed in Table 3.

4.2 Model Evaluation

In this experiment, model performance was evaluated using confusion metrics, including accuracy, precision, recall, and F1-score. These metrics are based on the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). True positives represent instances where the system correctly detects an attack, while false positives are cases where the system incorrectly predicts an attack for benign traffic. True negatives are instances correctly classified as benign, and false negatives occur when attack traffic is mistakenly classified as benign. The evaluation metrics are defined in Equations 4–7.

Furthermore, to evaluate the model's ability to distinguish between classes at various threshold settings, we utilized the receiver operating characteristic (ROC) curve and the area under the

curve (AUC). The ROC curve plots the true positive rate, which is equivalent to recall, against the false positive rate (FPR). The FPR measures the proportion of benign traffic incorrectly classified as an attack and is defined in Equation 8:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (4)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (5)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (6)$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (7)$$

$$\text{FPR} = \frac{FP}{FP+TN} \quad (8)$$

4.3 Result of Model Selection

The performance of various machine learning models was evaluated to determine the most effective algorithm for DDoS detection. The models tested included LR, DT, KNN, CatBoost, RF, and XGBoost. All algorithms were trained on the same dataset, which had already undergone normalization and SMOTE for class balancing. At this stage, feature selection was not applied, and all features were used for testing. The evaluation results are summarized in Table 4.

Table 3 Software packages and versions used in the experiment

Package	Version	Description
Python	3.12	Python programming language
Scikit-learn	1.6.1	Preprocessing, feature selection, and classical ML models
XGBoost	3.1.2	Gradient boosting framework for optimized classification
CatBoost	1.2.8	Robust gradient boosting algorithm with symmetric trees
Pandas	2.2.2	Data structure handling in data frame format
NumPy	2.0.2	Numerical computing and array operations
Matplotlib	3.10.0	Graphical visualization and plotting
Seaborn	0.13.2	Statistical data visualization
Imbalanced-learn	0.14.0	Handling class imbalance
Optuna	4.6.0	Automatic hyperparameter optimization framework
PySwarms	1.3.0	PSO toolkit

Table 4 Comparative Performance of Baseline Machine Learning Models

Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
LR	95.18	97.73	87.30	92.23
DT	96.40	96.25	91.08	93.59
KNN	97.61	97.73	94.93	96.31
CatBoost	98.62	96.45	98.03	97.23
RF	99.00	98.97	98.16	98.56
XGBoost	99.14	99.40	97.98	98.69

From the results shown in Table 4 and Figure 4, ensemble learning models, including CatBoost, RF, and XGBoost significantly outperformed traditional models such as LR, KNN, and DT. While traditional algorithms struggle to capture the complex, non-linear relationships in high-dimensional network data or are prone to overfitting, ensemble methods overcome these limitations by combining multiple decision trees to reduce both bias and variance. Among these algorithms, XGBoost achieved the highest overall performance. Its success is attributed to its advanced gradient boosting framework, which sequentially builds decision trees to correct previous prediction errors. By optimizing a differentiable loss function, it precisely targets and minimizes classification mistakes. Furthermore, its integrated regularization techniques effectively manage complex feature interactions and imbalanced data to prevent overfitting.

4.4 Result of Features Selection

The next step of the experiment focused on the number of features with the highest information gain values to use in hyperparameter optimization. The advantage of feature selection techniques is that they not only reduce the number of variables, but it also decreases model complexity, enable faster model training and testing, reduce overfitting, and improve generalization. In this study, we searched for the optimal number of features by testing different feature set sizes. We started with the top 5 features and evaluated them using the XGBoost model with F1-score value. The results were recorded, and then the number of features increased to 10, 15, and 20 until using all features.

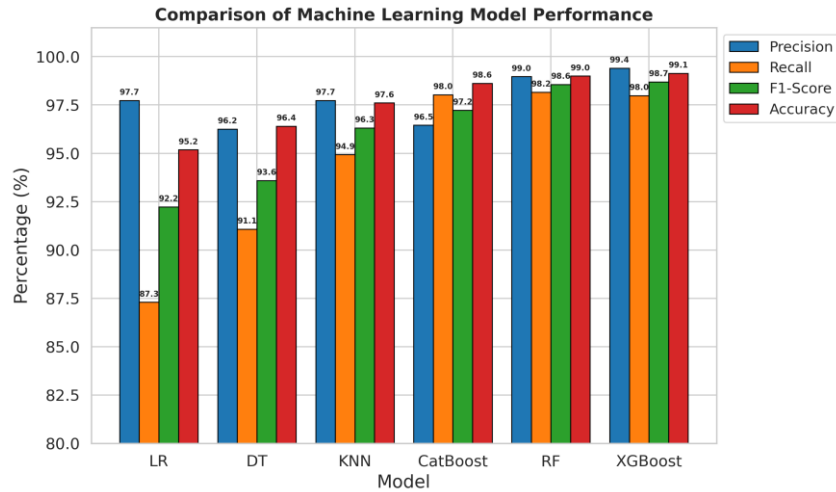


Figure 4 Comparison of Machine Learning Model Performance

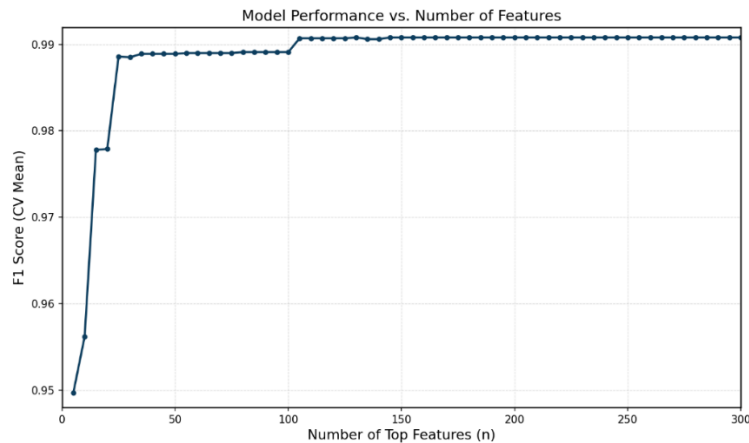


Figure 5 Effect of the Number of Selected Features on XGBoost

We evaluated feature selection performance by gradually adding more input features based on their information gain ranking. The results are shown in Figure 5; the x-axis represents the number of top-ranked features while the y-axis represents the cross-validated F1-score, which demonstrates the model's predictive reliability across varying feature subset sizes. The graph shows a rapid increase in the F1-score at the beginning, indicating that the top-ranked information gain features carry significant information. As training continued, the improvement in F1-score became very small. Therefore, we selected the top 100 features to balance the trade-off between the slight gain in F1-score of approximately 0.001 and the increased model complexity.

4.5 Performance of Hyperparameter Optimization

For additional performance improvement, two optimization methods were applied: Bayesian optimization and particle swarm optimization. Both methods explored the same set of key parameters, such as the number of trees, learning rate, depth, and sampling ratios. Bayesian optimization searched for the best values by building a simple probabilistic model and updating it after each trial. PSO, on the

other hand, searched through a group of candidate solutions that move across the search space. These two approaches helped the model identify better hyperparameters and improved overall detection performance. The optimized values obtained from the optimization methods are summarized in Table 5.

Hyperparameter optimization produced different results for XGBoost. PSO selected a lighter model with 391 trees, depth 8, and a small learning rate of 0.059. Therefore, PSO showed a more stable and less complex model. In contrast, the Bayesian method chose a stronger setup with 441 trees, depth 9, and a learning rate of 0.087. The results indicate that Bayesian search pushed the model to learn faster and use a deeper structure.

Furthermore, the genetic algorithm (GA) generated the deepest and most complex structure by selecting a maximum depth of 10, combined with 442 trees and a high learning rate of 0.176. Finally, simulated annealing (SA) selected the largest ensemble size with 452 trees, a depth of 9, and the highest learning rate of 0.189, all values are shown in Table 5. After applying these tuned parameters together with the top 100 selected features, the final performance results for the XGBoost model are presented in Table 6.

Table 5 Optimized Hyperparameters of XGBoost

ML Algorithm	Optimization technique	Hyperparameter	Type	Range	Optimized value
XGBoost	PSO	N estimators	Discrete	200 – 500	391
		Learning rate	Continuous	0.01 – 0.2	0.0595
		Max depth	Discrete	3 – 10	8
		Subsample	Continuous	0.7 – 1.0	0.7001
		Colsample by tree	Continuous	0.7 – 1.0	0.9754
XGBoost	Bayesian Optimization	N estimators	Discrete	200 – 500	441
		Learning rate	Continuous	0.01 – 0.2	0.0872
		Max depth	Discrete	3 – 10	9
		Subsample	Continuous	0.7 – 1.0	0.8387
		Colsample by tree	Continuous	0.7 – 1.0	0.9180
XGBoost	Genetic Algorithm	N estimators	Discrete	200 – 500	442
		Learning rate	Continuous	0.01 – 0.2	0.1763
		Max depth	Discrete	3 – 10	10
		Subsample	Continuous	0.7 – 1.0	0.896
		Colsample by tree	Continuous	0.7 – 1.0	0.876
XGBoost	Simulated Annealing	N estimators	Discrete	200 – 500	452
		Learning rate	Continuous	0.01 – 0.2	0.1891
		Max depth	Discrete	3 – 10	9
		Subsample	Continuous	0.7 – 1.0	0.8446
		Colsample by tree	Continuous	0.7 – 1.0	0.962

Table 6 Comparison of Hyperparameter Optimization Results for XGBoost

ML Algorithm	Features	Optimization Technique	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	ROC-AUC (%)
XGBoost	100	No	99.15	99.38	98.03	98.70	98.87
XGBoost	100	Bayesian	99.29	99.26	98.57	98.91	99.11
XGBoost	100	PSO	99.27	99.39	98.38	98.88	99.04
XGBoost	100	Simulated Annealing	99.27	99.24	98.53	98.88	99.08
XGBoost	100	Genetic Algorithm	99.25	99.13	98.57	98.85	99.07

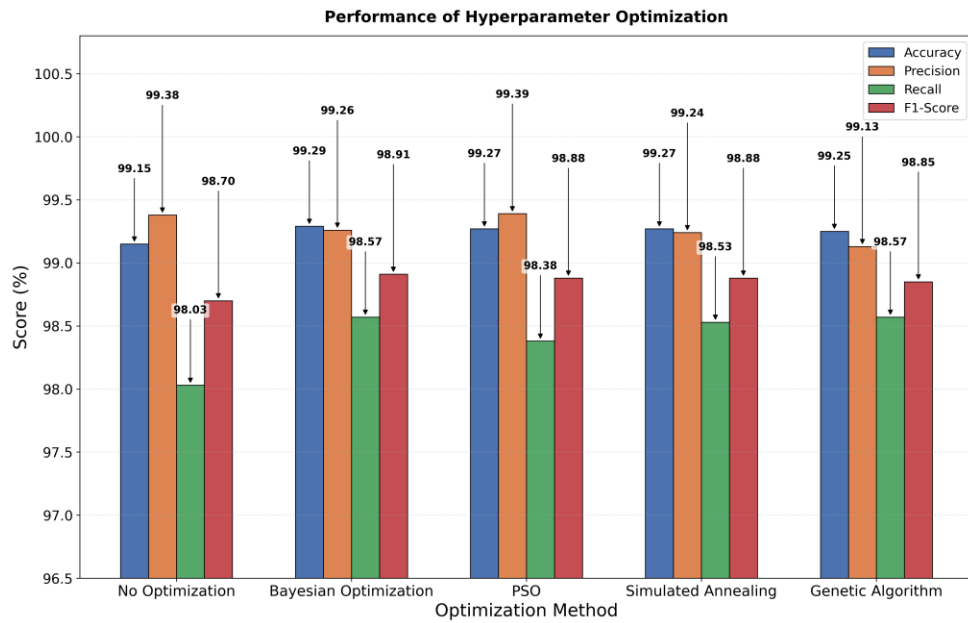


Figure 6 Hyperparameter Optimization Results for XGBoost

As shown in Figure 6 and Table 6, the results show that tuning the hyperparameters improves model performance. The baseline XGBoost model, trained without any optimization, achieved an accuracy of 99.15%, a precision of 99.38%, a recall of 98.03%, an F1-score of 98.70%, and an ROC-AUC of 98.87%. After applying Bayesian optimization, the model reached an accuracy of 99.29%, a precision of 99.26%, a recall of 98.57%, an F1-score of 98.91%, and an ROC-AUC of 99.11%. With hyperparameters tuned by PSO, XGBoost achieved an accuracy of 99.27%, a precision of 99.39%, a recall of 98.38%, an F1-score of 98.88%, and an ROC-AUC of 99.04%. These results indicate that hyperparameter tuning, particularly with Bayesian optimization, not only enhances accuracy and F1-score but also improves the ROC-AUC, demonstrating the model's enhanced capability to distinguish between benign and attack traffic compared with the baseline model.

To provide a more comprehensive comparison, we included simulated annealing and the genetic

algorithm to extend the experiments. As shown in Table 6, all techniques improved the baseline XGBoost model, but Bayesian optimization consistently maintained the highest overall performance in accuracy, F1-score, and ROC-AUC. Interestingly, the genetic algorithm achieved the same high recall as the Bayesian method but had the lowest precision, which leads to a lower F1-score. Meanwhile, simulated annealing provided a good balance, matching PSO in both accuracy and F1-score, while achieving slightly better recall and ROC-AUC. This analysis confirms that while search algorithms such as PSO, simulated annealing, and genetic algorithm are highly capable, Bayesian optimization remains the most reliable and efficient method for fine-tuning XGBoost in DDoS detection.

The confusion matrix results reveal distinct performance characteristics among the optimization strategies, as shown in Figure 7 and Table 7. The Bayesian optimization technique demonstrated superior capability in identifying malicious traffic by

achieving the highest number of TP at 33,600 and the lowest count of FN at 487 samples. This indicates that the Bayesian approach significantly reduces the risk of missed attacks compared with the non-optimized model, which had an FN of 672 and the PSO technique, which had false negatives of 552. Consequently, Bayesian optimization proves to be the most robust method for minimizing security breaches in a critical cloud environment.

Regarding false alarm rates, the PSO algorithm achieved the best stability with the lowest false positive count of 206 and a false positive rate of 0.295%. The baseline model performed similarly with an FP of 208, whereas the Bayesian optimization

model showed a slight increase to 250 FPs and false positive rate of 0.358%. While SA achieved a competitive detection rate, with a TP of 33,599, nearly on par with the Bayesian method, it produced the highest false positive rate of 0.421%. While GA achieved a competitive detection rate, with a TP of 33,599, nearly on par with the Bayesian method, it produced the highest false positive rate of 0.421%. Meanwhile, SA produced a lower false positive rate of 0.368%. Similarly, GA demonstrated slightly less optimal results, with a false positive rate of 0.368%. Although the Bayesian model caused slightly more false alarms, it provided the best detection of actual threats, effectively improving overall security.

Table 7 Confusion Matrix Results

ML Algorithm	Optimization Technique	True Positive	False Negative	True Negative	False Positive	False Positive Rate (%)
XGBoost	No	33,415	672	69,628	208	0.298
XGBoost	Bayesian	33,600	487	69,586	250	0.358
XGBoost	PSO	33,535	552	69,630	206	0.295
XGBoost	Simulated Annealing	33,585	502	69,579	257	0.368
XGBoost	Genetic Algorithm	33,599	488	69,542	294	0.421

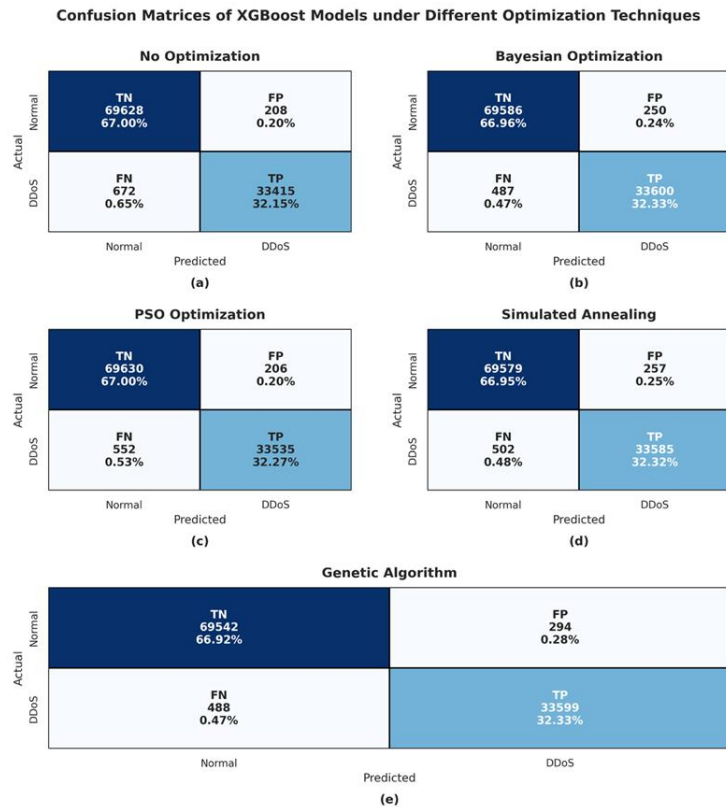


Figure 7 Confusion Matrix of XGBoost Models. (a) XGBoost (No Optimization) (b) XGBoost with Bayesian Optimization (c) XGBoost with particle swarm optimization (PSO) (d) XGBoost with simulated annealing (e) XGBoost with genetic algorithm

5. Discussion

The superior performance of XGBoost can be explained by its gradient boosting framework, which sequentially builds decision trees by optimizing a differentiable loss function. This approach allows XGBoost to effectively reduce both bias and variance, handle feature interactions efficiently, and manage imbalanced data. Furthermore, its regularization techniques prevent overfitting, which contributes to its robustness compared with other algorithms evaluated in this study. Therefore, XGBoost was selected for further experiments.

With the top 100 features, the model approaches the highest F1-score, and adding more features beyond 100 does not substantially improve performance but increases computational load. Therefore, selecting these top 100 features provides a good balance between model complexity and detection accuracy, keeping the system efficient while maintaining strong predictive performance.

The Bayesian-optimized model achieves the highest accuracy at 99.29% and also improves recall to 98.57%, which means it can detect more real attack instances. Its F1-score also increases to 98.91, showing a better balance between precision and recall. The PSO-tuned model produces a similar improvement, achieving 99.27% accuracy with a recall of 98.38% and an F1-score of 98.88. Although slightly lower than Bayesian, it still performs better than the non-optimized model.

Overall, the results confirm that hyperparameter optimization enhances the model's ability to capture attack traffic while keeping false predictions low. The improvements in recall and F1-score are especially important because they reflect the model's capability to identify real threats more effectively. Comparing the model with different hyperparameter optimization techniques reveals that Bayesian optimization yields

slightly better results than PSO, indicating that Bayesian optimization can consistently and more accurately identify optimal parameters.

The results show that Bayesian optimization produced slightly lower precision than the baseline model, but it greatly improved recall and the F1-score. This result reflects the trade-off between precision and recall. When a model attempts to detect more actual attacks, it can sometimes mistake normal traffic for an attack, which causes precision to decrease. For a DDoS detection system, high recall is more important for ensuring that harmful traffic is not missed. Therefore, Bayesian optimization balanced this trade-off well by improving the overall F1-score, making the model more reliable.

5.1 Ablation Study and Component Analysis

The ablation study comprises two comparative analyses: (i) the effect of feature dimensionality reduction and a comparison of the proposed information gain method with standard feature selection techniques and (ii) the performance gains achieved using Bayesian and PSO optimization compared with a non-tuned baseline. The results are presented in Tables 8 and 9, respectively.

Table 8 demonstrates that reducing the feature set from 326 to 100 using information gain does not degrade performance. Furthermore, given the large size of the test dataset, the slight improvements in accuracy (0.02%) and recall (0.05%) are statistically meaningful. Importantly, this reduction to approximately 30% of the original dimensionality significantly lowers computational complexity. The F1-score of 98.70% validates that the selected subset captures the essential attack patterns, resulting in a lighter and more efficient model suitable for deployment.

Table 8 Impact of Feature Selection Between the Baseline Model and Information Gain

Feature Strategy	Number of Features	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Baseline Model	326	99.14	99.40	97.98	98.69
Information Gain	100	99.16	99.38	98.03	98.70

Table 9 Comparison of Different Feature Selection Methods

Feature Selector	Number of Features	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Tree-based	100	99.06	99.14	98.73	98.93
Information Gain	100	99.16	99.38	98.03	98.70

Table 9 compares the proposed information gain method with a tree-based feature importance approach. Both methods were evaluated using the top 100 features. The results indicate that the proposed information gain method achieves superior accuracy of 99.16% and precision of 99.38% compared with the tree-based method at 99.06% and 99.14%, respectively. Although the tree-based approach yields a slightly higher recall, the information gain method was selected for the final framework due to its capability to minimize false positives with higher precision and maintain the highest overall accuracy.

6. Conclusion

Network security is a significant problem in the digital era, especially in cloud-based systems, and it can cause enormous damage to data privacy and services. This study developed a method to detect DDoS attacks using a machine learning framework designed to achieve high performance while maintaining a model complexity suitable for practical use. The experimental results demonstrated that the XGBoost algorithm achieved the highest accuracy when using data from the top 100 information gain features. Furthermore, optimizing hyperparameters significantly improves predictive quality. The proposed framework was evaluated on the BCCC-cPacket-Cloud-DDoS-2024 dataset, achieved the highest performance, with an accuracy of 99.29%, precision of 99.26%, recall of 98.57%, and F1-score of 98.91%. The experimental results demonstrate that the proposed framework improves classification performance while reducing model complexity by selecting an optimal and compact feature set.

However, it is important to acknowledge a limitation regarding the evaluation strategy employed in this study. We utilized a stratified random split technique, which effectively validates the model's performance on unseen instances of known attacks but may not fully simulate zero-day attack scenarios in which the model encounters completely new attack categories.

To address zero-day attacks, future work will incorporate leave-one-attack-type-out validation methods to rigorously assess robustness against novel threats. Additionally, we plan to explore a hybrid resampling approach combining downsampling of the majority class with the SMOTE technique. Furthermore, we aim to extend the framework to multiclass classification and deploy the proposed model in host-based or network-based software-defined network (SDN) tools for real-time detection.

7. Acknowledgements

We sincerely thank King Mongkut's Institute of Technology Ladkrabang (KMITL) for their invaluable support in providing tuition fee funding and research resources.

8. Abbreviations

Abbreviation	Full term
CatBoost	Categorical Boosting
DDoS	Distributed Denial-of-Service
DT	Decision Tree
IDS	Intrusion Detection System
KNN	k-Nearest Neighbors
LR	Logistic Regression
ML	Machine Learning
PSO	Particle Swarm Optimization
RF	Random Forest
ROC-AUC	Receiver Operating Characteristic–Area Under the Curve
SMOTE	Synthetic Minority Over-sampling Technique
XGBoost	Extreme Gradient Boosting

9. CRediT Statement

Posathip Sathaporn: Conceptualization, Methodology, Software, Formal analysis, Writing - original draft. **Woranidtha Krungseanmuang:** Data curation, Validation, Investigation.

Vasutorn Chaowalittawin: Visualization, Validation, Writing - review & editing.

Chawalit Benjangkprasert: Resources, Project administration, Validation, Writing - review & editing.

Tuanjai Archevapanich: Validation, Writing - review & editing.

Boonchana Purahong: Supervision, Conceptualization, Funding acquisition, Validation

10. Data Availability Statement

The BCCC-cPacket-Cloud-DDoS-2024 dataset presented in this study is openly available in the Behaviour-Centric Cybersecurity Center (BCCC) repository at the following website: <https://www.yorku.ca/research/bccc/ucs-technical/cybersecurity-datasets-cds/> (accessed on 14 March 2025). The dataset is officially described in the literature via DOI: <https://doi.org/10.3390/info15040195>.

11. References

Ahmad, I., Imran, M., Qayyum, A., Ramzan, M. S., & Alassafi, M. O. (2023). An optimized

- hybrid deep intrusion detection model (HD-IDM) for enhancing network security. *Mathematics*, 11(21), Article 4501. <https://doi.org/10.3390/math11214501>
- Al-Shareeda, M. A., Manickam, S., & Saare, M. A. (2023). DDoS attacks detection using machine learning and deep learning techniques: analysis and comparison. *Bulletin of Electrical Engineering and Informatics*, 12(2), 930-939. <https://doi.org/10.11591/eei.v12i2.4466>
- Awan, M. J., Farooq, U., Babar, H. M. A., Yasin, A., Nobanee, H., Hussain, M., ... & Zain, A. M. (2021). Real-time DDoS attack detection system using big data approach. *Sustainability*, 13(19), Article 10743. <https://doi.org/10.3390/su131910743>
- Becerra-Suarez, F. L., Fernández-Roman, I., & Forero, M. G. (2024). Improvement of distributed denial of service attack detection through machine learning and data processing. *Mathematics*, 12(9), Article 1294. <https://doi.org/10.3390/math12091294>
- Bhattacharya, S., S, S. R. K., Maddikunta, P. K. R., Kaluri, R., Singh, S., Gadekallu, T. R., Alazab, M., & Tariq, U. (2020). A novel PCA-firefly based XGBoost classification model for intrusion detection in networks using GPU. *Electronics*, 9(2), Article 219. <https://doi.org/10.3390/electronics9020219>
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. <https://doi.org/10.1023/A:1010933404324>
- Breiman, L., Friedman, J., Olshen, R. A., & Stone, C. J. (2017). *Classification and regression trees*. New York, US: Chapman and Hall/CRC. <https://doi.org/10.1201/9781315139470>
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321-357. <https://doi.org/10.1613/jair.953>
- Chen, S. R., Chen, S. J., & Hsieh, W. B. (2025). Enhancing machine learning-based DDoS detection through hyperparameter optimization. *Electronics*, 14(16), Article 3319. <https://doi.org/10.3390/electronics14163319>
- Chen, T., & Guestrin, C. (2016). *Xgboost: A scalable tree boosting system* [Conference presentation]. Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining, New York, US. <https://doi.org/10.1145/2939672.2939785>
- Cil, A. E., Yildiz, K., & Buldu, A. (2021). Detection of DDoS attacks with feed forward based deep neural network model. *Expert Systems with Applications*, 169, Article 114520. <https://doi.org/10.1016/j.eswa.2020.114520>
- Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27. <https://doi.org/10.1109/TIT.1967.1053964>
- Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 20(2), 215-232. <https://doi.org/10.1111/j.2517-6161.1958.tb00292.x>
- Dasari, S., & Kaluri, R. (2024). An effective classification of DDoS attacks in a distributed network by adopting hierarchical machine learning and hyperparameters optimization techniques. *IEEE Access*, 12, 10834-10845. <https://doi.org/10.1109/ACCESS.2024.3352281>
- Indrasiri, P. L., Lee, E., Rupapara, V., Rustam, F., & Ashraf, I. (2022). Malicious traffic detection in IoT and local networks using stacked ensemble classifier. *Computers, Materials and Continua*, 71(1), 489-515. <https://doi.org/10.32604/cmc.2022.019636>
- Kennedy, J., & Eberhart, R. (1995). *Particle swarm optimization* [Conference presentation]. Proceedings of ICNN'95-International Conference on Neural Networks, Perth, WA, Australia. <https://doi.org/10.1109/ICNN.1995.488968>
- Liu, Z., Wang, Y., Feng, F., Liu, Y., Li, Z., & Shan, Y. (2023). A DDoS detection method based on feature engineering and machine learning in software-defined networks. *Sensors*, 23(13), Article 6176. <https://doi.org/10.3390/s23136176>
- Mohmand, M. I., Hussain, H., Khan, A. A., Ullah, U., Zakarya, M., Ahmed, A., ... & Haleem, M. (2022). A machine learning-based classification and prediction technique for DDoS attacks. *IEEE Access*, 10, 21443-21454. <https://doi.org/10.1109/ACCESS.2022.3152577>
- Krishna, S. G., & Sahu, K. K. (2015). Normalization: A preprocessing stage. *International Advanced Research Journal in Science, Engineering and*

- Technology*, 2(3), 20–22.
<https://doi.org/10.17148/IARJSET.2015.2305>
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 31.
https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf
- Rai, M., & Mandoria, H. L. (2019). *Network Intrusion Detection: A comparative study using state-of-the-art machine learning methods* [Conference presentation]. 2019 international conference on issues and challenges in intelligent computing techniques (ICICT). IEEE, Ghaziabad, India.
<https://doi.org/10.1109/ICICT46931.2019.8977679>
- Sadhwani, S., Manibalan, B., Muthalagu, R., & Pawar, P. (2023). A lightweight model for DDoS attack detection using machine learning techniques. *Applied Sciences*, 13(17), Article 9937. <https://doi.org/10.3390/app13179937>
- Saghezchi, F. B., Mantas, G., Violas, M. A., de Oliveira Duarte, A. M., & Rodriguez, J. (2022). Machine learning for DDoS attack detection in industry 4.0 CPPSs. *Electronics*, 11(4), Article 602.
<https://doi.org/10.3390/electronics11040602>
- Saini, P. S., Behal, S., & Bhatia, S. (2020). *Detection of DDoS attacks using machine learning algorithms* [Conference presentation]. 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom), IEEE, New Delhi, India.
<https://doi.org/10.23919/INDIACom49435.2020.9083716>
- Sanchez, O. R., Repetto, M., Carrega, A., & Bolla, R. (2021). *Evaluating ML-based DDoS detection with grid search hyperparameter optimization* [Conference presentation]. 2021 IEEE 7th International Conference on Network Softwarization (NetSoft), IEEE, Tokyo, Japan.
<https://doi.org/10.1109/NetSoft51509.2021.9492633>
- Santos, R., Souza, D., Santo, W., Ribeiro, A., & Moreno, E. (2020). Machine learning algorithms to detect DDoS attacks in SDN. *Concurrency and Computation: Practice and Experience*, 32(16), Article e5402.
<https://doi.org/10.1002/cpe.5402>
- Sathaporn, P., Krungseanmuang, W., Chaowalittawin, V., Benjangkprasert, C., & Purahong, B. (2025). DDoS detection using a hybrid CNN–RNN model enhanced with multi-head attention for cloud infrastructure. *Applied Sciences*, 15(21), Article 11567.
<https://doi.org/10.3390/app152111567>
- Shafi, M., Lashkari, A. H., Rodriguez, V., & Nevo, R. (2024). Toward generating a new cloud-based distributed denial of service (DDoS) dataset and cloud intrusion traffic characterization. *Information*, 15(4), Article 195. <https://doi.org/10.3390/info15040195>
- Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). *Toward generating a new intrusion detection dataset and intrusion traffic characterization* [Conference presentation]. Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP), Madeira, Portugal.
<https://doi.org/10.5220/0006639801080116>
- Sharafaldin, I., Lashkari, A. H., Hakak, S., & Ghorbani, A. A. (2019). *Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy* [Conference presentation]. 2019 international carnahan conference on security technology (ICCST), IEEE, Chennai, India.
<https://doi.org/10.1109/CCST.2019.8888419>
- Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25.
https://proceedings.neurips.cc/paper_files/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html
- Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7, 41525–41550.
<https://doi.org/10.1109/ACCESS.2019.2895334>
- Yoachimik, O., & Pacheco, J. (2025). Hyper-volumetric DDoS attacks skyrocket: Cloudflare's 2025 Q2 DDoS threat report. Retrieved from <https://blog.cloudflare.com/ddos-threat-report-for-2025-q2/>