IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 31, No. 4, April, 2026, pp. 1137-1154
Journal homepage: http://iieta.org/journals/isi

Hyperparameter Optimization for Tree-Based Machine Learning Models in Internet of Things (IoT) Intrusion Detection: Comparative Binary and Multi-Class Study

Arshad M. Khaleel[1*], Abdulqader M. Hashim[2], Ali Saadoon Ahmed[2], Abdolkader M. Khaled[3]

[1] 1st Iraq Ministry of Education 2nd International Smart Card Al-Anbar, Al Anbar 31001, Iraq
[2] Department of Computer Sciences, College of Sciences, University of Al Maarif, Al Anbar 31001, Iraq
[3] Faculty of Engineering, Department of Electrical Techniques Engineering, Al-Ma'moon University College, Baghdad 10001, Iraq

Corresponding Author Email: eng.arshed9@gmail.com

https://doi.org/10.18280/isi.310411

Received: 22 November 2025
Revised: 30 January 2026
Accepted: 11 April 2026
Available online: 30 April 2026

Keywords:
Internet of Things intrusion detection, tree-based machine learning, hyperparameter optimization, Salp Swarm Optimization, binary classification, multi-class classification

ABSTRACT

Intrusion detection in the Internet of Things (IoT) networks is a vital area of research. Tree-based ML algorithms are highly sensitive to tuning of hyperparameters. The objective of this paper is to analyze the effect of hyperparameter optimization (HO) of 5 popular tree based ML algorithms (Decision Tree, Random Forest (RF), Extra Trees, XGBoost and Gradient Boosting) using Salp Swarm Optimization (SSO). The experiments are performed under binary classification and multi-class classification tasks across 461,043 network traffic records, where 300,000 samples are benign and 161,043 samples belong to 9 attack types. The class imbalance is handled by downsampling to achieve balanced samples for training purpose. Evaluation metrics of model performance: Following evaluation metrics are used to evaluate the model performance: accuracy, precision, recall, F1-score, confusion matrix, and training time. The experimental results indicate that the detection performance of all classifiers can be significantly enhanced by hyperparameter tuning with SSO. The best classification accuracy for binary classification is 99.45% with optimized Decision Tree classifier and the highest classification accuracy for multi-class classification is 98.41% with optimized Extra Trees classifier. More detailed investigation reveals that systematic hyperparameter optimization algorithm can significantly alleviates false positives and false negatives, enhances the capacity to recognize minority class samples and results into more robust class-wise performance. Sensitivity analysis and ablation experiment demonstrate that the effect of optimized parameter values on better generalization capability and more stable model robustness is notable and it is strongly appropriate for security applications. In conclusion, the main contribution of this work is that, it provides a valuable and a repeatable method to evaluate and deploy a top-ranked optimized classifier for binary security problem and multi-class intrusion discrimination tasks.

1. INTRODUCTION

Internet of Things (IoT) devices have been a major contributor to the complexity of today's networked world. The IoT systems are being used in smart homes, healthcare, transportation, industrial automation and critical infrastructure. This growth has, however, also made IoT networks more vulnerable to cyber-attacks, and intrusion detection (ID) is an integral part of IoT security [1-3]. Machine learning (ML) has emerged as a key method for identifying abnormal network traffic, as it can be trained to identify discriminative patterns in the network traffic and classify the activity as either benign or malicious.

The problem of ID in IoT environments is typically treated as a binary classification or a multi-class classification problem. For binary classification, the network traffic is divided into two classes: benign and malicious. This setting can be helpful to determine if an attack is present. However, this does not specify what sort of threat. In contrast, a multi-class approach categorizes malicious traffic into attack class types and can provide more detailed data for cyber security analysis and response [4]. Each of these scenarios is hard when the data being classified has many millions of noisy, high dimensional characteristics, and the number of non-malicious packets considerably outweighs malicious ones (i.e., imbalanced). This imbalance can cause classifiers to be biased towards the dominant non-malicious class, making it difficult to detect minority attack classes.

Preprocessing and model tuning have been identified as important to find solutions to these issues. Data preprocessing is used to improve the quality and robustness of the data set through missing data, duplicate cases, class imbalance and feature scaling [5]. For numerical input features that are on different scales (e.g., Min-Max scaling) feature scaling is performed. These pre-processing steps can be performed to reduce the noise ensure the stability of the model and provide

a basis for comparison of the classifiers.

Hyperparameter optimization (HO) is also crucial in enhancing classification performance, aside from preprocessing. ML models are very influenced by the dictation of their hyperparameters, for example the number of estimators, the deepness of the tree, the criterion to split, the minimum number of samples per split or even the learning rate. The Salp Swarm Optimization (SSO) algorithm is one of the most popular metaheuristic optimization methods that has been studied for feature selection and hyperparameter tuning due to its capability to explore complex search spaces [6]. In this study, SSO is used to tune selected tree-based classifiers, and to investigate the impact of optimization on the detection performance of these classifiers in IoT ID.

While there have been many previous studies that have used ML and deep learning (DL) techniques in ID, there are still some drawbacks. First, many studies only consider the overall accuracy, and only a few studies consider the impact of HO on the performance of the model in both binary and multi-class classification. Second, DL techniques can be complex and can be very powerful in terms of detection performance, but they can also be resource-intensive and may not be very interpretable, which can limit their applicability in resource-constrained IoT applications. Third, previous studies do not consistently compare optimized and non-optimized tree-based classifiers in the same experimental context, and optimized results are often reported. These restrictions make it important to have a systematic comparison of optimization effects, predictive performance and computational efficiency.

Thus, the impact of hyperparameter tuning using SSO on tree-based ML classifiers for IoT ID is explored. The classifiers evaluated are Decision Tree (DT), Random Forest (RF), Extra Trees (ET), XGBoost and Gradient Boosting (GB). The models are evaluated with both binary and multi-class classification in terms of accuracy, precision, recall, F1 score, error rate, confusion matrix analysis and training time [7-9]. This study attempts to shed some light onto the relative strengths and weaknesses of SSO-based tuning of tree based id models by comparing optimised and non-optimised models.

This study has made the following contributions:

1. This study provides a comparative assessment of various tree-based ML classifiers for IoT ID in both binary and multi-class classification scenarios.
2. It employs SSO as a hyperparameter optimizing approach; and investigate the impact of optimizing hyperparameters on the performance of models in comparison with none-optimized models.
3. It assesses the models with several indicators (accuracy, precision, recall, F1-score, error rate, confusion matrix and time needed for training) rather than only overall accuracy.
4. It offers a well-ordered methodological flow of preprocessing, classification, HO and assessment, making the reproducibility of the ID schema proposed.

The rest of this paper is structured as follows. In Section 2, the research needs this project fills are highlighted and similar work on ML-based ID is reviewed. The suggested approach, which includes preprocessing, binary classification, SSO-based optimization, exploratory data analysis (EDA), and multi-class classification, is presented in Section 3. The experimental findings are shown and discussed in Section 4. Section 5 summarizes the study's findings and suggests further paths of inquiry.

2. RELATED WORK

Hence, ML and DL are recognized to be efficient approaches to finding new intricate intrusions, including MitM and DoS attacks [10]. Several techniques have been used for ID, including the Support Vector Machine (SVM) [11, 12], the Semi-supervised Spatio-Temporal DL ID (SSDeep-ID) [13], and the Deep Feed-Forward Neural Network (DFFNN) [14]. In a comparative study, some of the most used ML algorithms, like Autoencoders, RF, K-Means clustering, and IF, were used to detect attacks in the IoT setting, though there was no clear explanation of the types of attacks considered [15].

Thus, the DL based network ID system (IDS) was discussed by Qazi et al. [16], which has been tested on the datasets including CIC-IDS2017 and UNSW-NB15. The accuracy and the time taken to predict the results were used as criteria for evaluating the system, to show the suitability of DL algorithms in the development of IDSs for IoT networks. The experiments focused on 32 kinds of attacks determined by CIC-IDS2017 and UNSW-NB15; however, it is crucial to mention that the above datasets are general and do not include MQTT-associated attacks.

In another study, authors presented a new dataset known as MQTTset and presented different ML algorithms suitable for ID [17]. Koroniotis et al. [18] investigated three ML techniques: The three models that are used herein for evaluation include SVM, Recurrent Neural Networks (RNN), and Long Short Term Memory Recurrent Neural Networks (LSTM-RNN) on the Bot-IoT sub-dataset containing all the 46 features used in the experiment. The performance comparison summarized from the findings revealed that although the training time exerted by SVM was shown to be the longest in all the experimental methods explored for its application, it claimed to have been the most accurate and had the highest recall rate as well. Surprisingly, we find that even though a 10-feature SVM classifier is inferior to the full-feature SVM in their recall and F-score values, it gives the highest precision with the lowest false positive rate among the methods used.

Further, several researchers have implemented IDS for IoT attack detection through a signature-based mechanism. Another such kind of approach presented in this study [19] employed a J48 DT model for the purpose of creating attack signature rules from an extract of approximately 659,977 instances from the Bot-IoT dataset. The model does propose 24 rules in total, 16 of which are specifically related to attack traffic: ‘‘Theft,’’ ‘‘DDoS,’’ ‘‘Probing,’’ and so on. Although this improves on current public signature-based IDSs such as Snort and Suricata, this model is also intended to create an IDS light enough to be directly used on IoT devices. However, the given performance metrics and the evaluation results were not part of this study.

Another remarkable contribution provided a novel architecture of an IDS of the IoT, which aimed at increasing the efficiency of attack detection by using both signature-based and behavior-based approaches. This system used a globally detected IDS using a C5 DT and a behavioral IDS that used the One-Class SVM and selected only 13 features from 46 features in the Bot-IoT datasets for the binary classification. A first block of analysis focuses on the results relative to the difference in the signature-based and behavioral components of the model in which the former yielded a total accuracy of 93.30% while the latter yielded a total of 92.50%. By combining the results of the recording devices, the required level of accuracy was achieved and amounted to 99.97%.

When comparing the proposed HIDS with other algorithms such as C4.5, NB, RF, multi-layer perceptron networks, SVM, CARTs, and KNN, it was found that the accuracy of the new HIDS was slightly higher than that of the other algorithms, and its false alarm rate was significantly lower than that of all the other algorithms; however, more time was spent building the model.

Baig et al. [20] addressed DoS attacks targeting IoT wireless sensor networks (WSNs) by proposing two innovative frameworks: MultiScheme and Voting. The MultiScheme framework chooses classifiers according to their train error rate, and the Voting scheme takes the arithmetic mean of the outputs of chosen classifiers. The study based the measurement of accuracy and time for computation on the Bot-IoT dataset. The Results indicated that, even with just five original features, both the MultiScheme and A2DE classifiers had almost equal levels of detection rates, with the MultiScheme classifier taking much more time to train. Specifically, the A2DE classifier identified DoS-HTTP attacks with near-perfect accuracy, of about 99.8%, a figure significantly higher than possible with the RNN of approximately 98%. However, this study mainly concentrated on DoS attacks but other activities of botnets such as OS Fingerprinting, Service Scanning, Data Exfiltration, and Keylogging.

DL and attention-based architectures have been the focus of recent studies in the field of network and IoT ID. Akuthota and Bhargava [21] suggested a Transformer-based IDS for IoT networks and tested it on NSL-KDD and UNSW-NB15 datasets in both binary and multiclass classification modes. They found that self-attention mechanisms can learn the complex traffic relationships and enhance the detection performance. In a similar manner, Hnamte et al. [22] introduced a lightweight deep convolutional neural network for ID and tested it on CICIDS2017, CICIoMT2024, and InSDN, demonstrating that lightweight deep architectures can enhance detection accuracy while ensuring computational efficiency. Farhan et al. [23] used a Sequential DNN and ET feature selection on UNSW-NB15 dataset, which showed that feature reduction can enhance the efficiency of deep IDS. Doost et al. [24] suggested a hybrid CNN and RF method, where CNN was employed to extract features and RF was employed to classify the features. Alsubaei [25] also highlighted the significance of preprocessing and HO of XGBoost and neural network models on three datasets: NSL-KDD, UNSW-NB15 and CICIDS2017.

While these recent studies prove the effectiveness of Transformer models, CNN-based architectures, hybrid DL models and optimized neural networks, they primarily concentrate on complex deep architectures or on the gains of the models on specific datasets. The present study, on the other hand, is based on interpretable and scalable tree-based classifiers and investigates the impact of hyperparameter tuning using SSO on the performance of the classifiers in the binary and multiclass ID scenarios. This separation is significant because tree-based models may be more computationally efficient and easier to interpret than some DL architectures, which is beneficial for practical use in IDS deployments with limited resources, such as in IoT deployments.

The literature review reveals that the recent IDS research has achieved significant progress in the areas of DL, Transformer-based architectures, feature selection and hybrid classification frameworks. But few studies have focused on comparing systematically optimized and non-optimized tree-based classifiers with the same experimental setup. Moreover, many of the studies focus on overall accuracy and do not have a lot of discussion about class-wise behavior, computational cost, and the impact of optimization on binary and multiclass classification. Hence, this study aims to fill this gap by testing the performance of the classifiers DT, RF, ET, XGBoost and GB classifiers and to examine the impact of HO using SSO on the classifiers by accuracy, precision, recall, F1-score, confusion matrix analysis and training time.

3. METHODOLOGY

The presented Figure 1 presents a glimpse of this methodology section, where the nature and type of data used for classification jobs are described, in addition to the pre-processing operations carried out on the data before feeding it to the model. This study's dataset consists of network traffic data, consisting of both normal and abnormal samples obtained from an IoT security context. This dataset corresponds indirectly to the activity typical for IoT networks and includes both normal and specific malicious activity, such as DoS attacks, probing, or information stealing. As for the informatics, the dataset is diverse and closely matches the real-world network and IT conditions due to the tendencies of IoT growth; therefore, the dataset is suitable for ML models' training and evaluation.

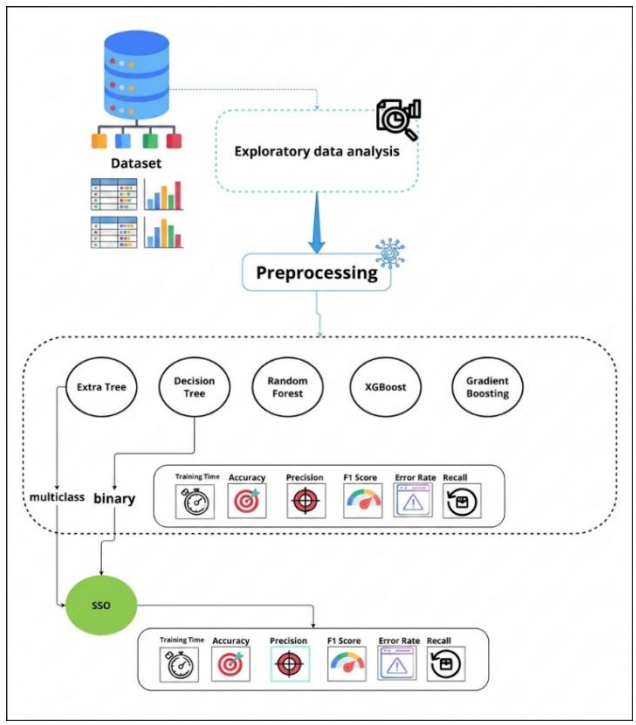

Figure 1. Proposed methodology

Preprocessing is a crucial phase within this study because the cybersecurity data is typically noisy and can be imbalanced. The first data set is preprocessed to identify and synchronize issues like input voids, replicate records, and categorized odd records, if any. Failure to address missing records may only result in a skewed performance of the model due to bias or errors that are occasioned by missing records. In this preprocessing stream, these missing entries are either supplied with reasonable values or dropped on the basis of

their nature and percentage of samples. In addition to that, problematic observations, such as duplicates, are also filtered out to improve learning and reduce the impact of over-representing patterns.

Another challenge is the imbalance of the samples between benign and malicious samples [26-28] which means that the size of benign data is always larger than the size of malware samples. Such an imbalance may put forward a model supporting a benign class and not be able to detect the attacks properly. To overcome this issue, various techniques such as under-sampling of benign class or down-sampling of some malicious classes are employed. Under-sampling subjudices the actual amount of benign samples that exist originally in favor of a quantitative alignment with the number of malicious entries present and thus guarantees that both classes got an equal portion of attention during the learning phase of the ML algorithm. It can be a technique to make sure that the classifier is trained with equal number of patterns from benign and malicious domains.

Another important aspect in the preprocessing is the normalization, particularly due to the large differences in the range of feature values [29-31]. Feature scaling using MinMaxScaler is applied to normalize specific network characteristics like packet size, protocol type and specific timestamp values. This scaling method keeps the data clean and also allows algorithms with specific requirements on the feature distributions to work on the data. As all the feature values are scaled to a fixed range, say [0, 1], MinMaxScaler prevents the true values, which possess huge variance individually, from shifting the model or determining the importance of features through scaling.

The last preprocessed dataset is intended to give a trustworthy and fair input to the whole modeling and testing procedure. Such pitfalls as the presence of missing values, duplicates, and imbalanced class distribution, and the feature values are normalized, resulting in a data preprocessing phase optimizing the dataset for a fair ML process. These preprocessing steps further enhance the reliability of the study because they prepare the dataset used in this study in a way that should not incorporate any abnormalities that could interfere with the performance of the developed models.

3.1 Exploratory data analysis

The EDA [32-34] section will be used in order to give readers an insight into the characteristics and the nature of the data that will be used in this study, in addition to assisting this study in identifying an appropriate model and feature engineering. EDA is therefore indispensable in acts of cybersecurity analysis, especially for datasets that have multiple characteristics of complex acts, such as IoT network traffic, in which it is difficult to discern between normal and anomalous behavior without understanding different characteristics, relations, and distributions prevailing in the dataset. Some of these are a careful analysis of the structure of the dataset, the interactions between the features, and the important distribution in a bid to find out information that would be useful in the detection of different cybersecurity threats.

First of all, EDA starts with the analysis of the largest amount of data set, which includes the number of samples, benign and malicious, and the range of the feature values, as shown in Figures 2-4. This first analysis defines data biases that are useful in determining skewed distributions or data gaps that may compromise the performance of the model. For example, when considering class distributions, we find that benign samples are more frequent than malicious ones, characteristic of cybersecurity data due to the fact that normal network traffic is much more common than attacks. Recognizing such an imbalance in the early steps helps to have a contextual understanding of the matter that will be useful in the next steps, such as model training and assessment.

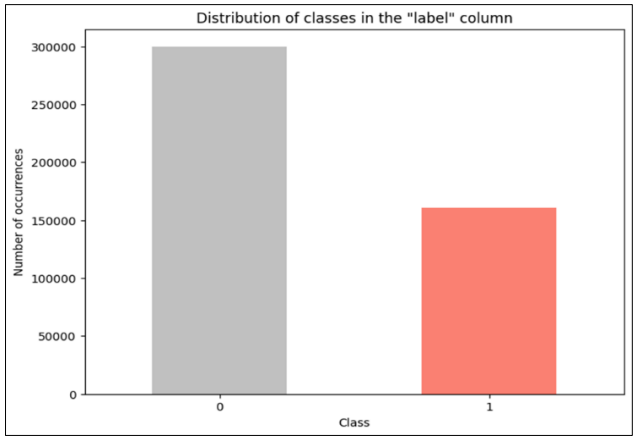

Figure 2. Overall class distribution of the original dataset

Figure 2 shows the number of samples in each traffic class before applying any balancing strategy. The distribution indicates the presence of class imbalance, where the majority classes contain substantially more samples than the minority attack classes. This imbalance motivates the use of class-balancing techniques before model training.

After this general study, each feature receives a detailed examination to understand its feature-specific peculiarities important in the context of the dataset. For the numerical features, distribution plots, histograms, and kernel density estimates are employed to understand the range and the frequency of values and to check which values are more frequent for the benign than for the malicious samples and vice versa. Such graphs make it possible to detect possible trends indicative of an attacker, in this case, increased packet size or deviations in protocol usage. In this manner, it will be easier to deduce which aspects might be most important, or lead to more accurate classification of the network traffic and, accordingly, which aspects should be magnified or altered during the feature engineering process.

Figure 3 highlights the dominance of majority traffic classes in the original training data. Such imbalance can bias classifiers toward frequent classes and reduce the detection capability for minority attack categories. Therefore, balancing is required to improve class-wise learning.

The features' dependencies are also explored by heatmaps and pairwise plots [35], which are also helpful for understanding whether there are a lot of dependencies or even the coincidence of attributes. The significance of this examination is that, in a ML environment, highly correlated features will create multicollinearity that can impact some models. A basic yet insightful visualization of feature pairs with potential linear correlations depending on mutual influence is achieved through compiling the heat map and scatter plot. So, by removing or combining features, the dataset is less influenced by overfitting and gives the model more specific targets and a more precise ranking, respectively.

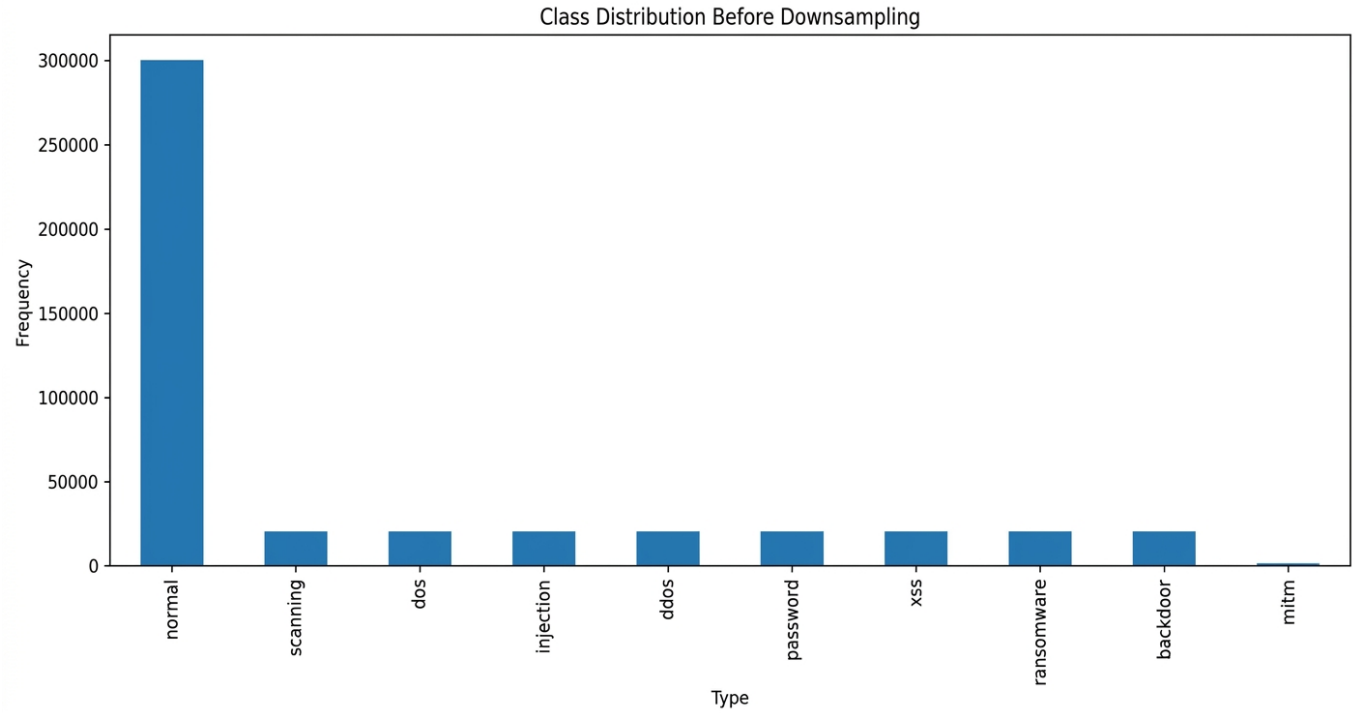

Figure 3. Class distribution before downsampling

Figure 4 presents the relative proportion of each class before balancing. The large share of majority classes confirms that overall accuracy alone may be misleading, since a classifier can achieve high accuracy while still failing to detect minority attack classes.

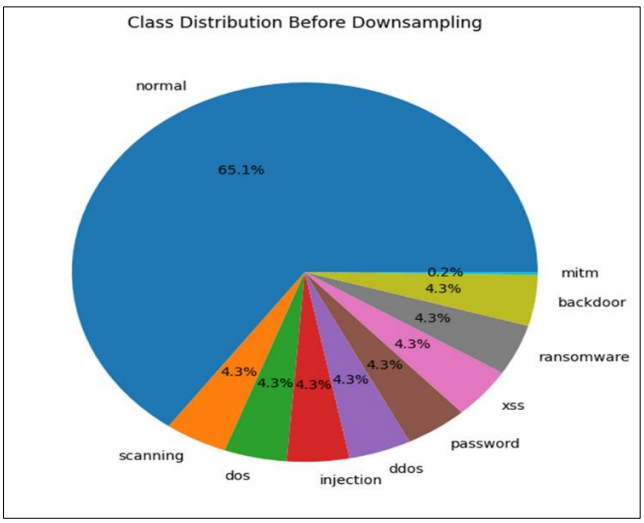

Figure 4. Percentage-based class distribution before downsampling

In network traffic data, extreme values can be either noise or an actual abnormal behavior that is related to a cyber attack, so these rare and irregular observations were carefully studied during EDA [36]. Outliers were inspected using graphical methods such as box plots and statistical methods such as z-score analysis. Records with extreme values were not automatically removed. Instead, values were reviewed in relation to their traffic class and feature behavior. An extreme value was kept if it was consistent with an attack pattern, as extreme values could be useful for the model to learn malicious traffic signatures. But values that were found to be incorrect and duplicate records were eliminated in the pre-processing stage to minimize noise and ensure unbiased model training.

A down sampling strategy was used prior to training the models to deal with class imbalance. The original data set consisted of 461,043 samples of which 300,000 were normal samples and 161,043 were attack samples.

The dataset was balanced for the binary classification experiment by downsampling the majority class to match the minority class, which yielded about 322,086 samples. The binary confusion matrices show that an 80/20 stratified split resulted in 257,668 training samples and 64,418 testing samples.

The data set in a multi-class setting contained 10 traffic classes: normal, scanning, DoS, injection, DDoS, password, XSS, ransomware, backdoor, and MitM. The normal class was dominant and the number of samples in the attack classes was less, so down sampling was used by retaining only 10,000 samples in each attack class. Normal, scanning, DoS, injection, DDoS, password, XSS, ransomware, and backdoor were the retained classes. The MitM class was not included in the down sampled multi-class experiment due to the small number of samples (1,043) in this class. Thus, the final balanced dataset comprised 90,000 samples, 10,000 samples per class, in 9 classes. The multi-class confusion matrices show that an 80/20 stratified split resulted in 72,000 training samples and 18,000 testing samples.

As seen Figure 5 after applying the balancing strategy, the distribution of samples across classes becomes more uniform. This reduces the dominance of majority classes and allows the classifier to learn more balanced decision boundaries across benign and malicious traffic categories.

The downsampling strategy results in 10,000 samples being retained per class, giving a balanced dataset of 90,000 samples across 9 traffic classes. This balancing step helps to minimize the influence of the normal class and enables the classifiers to learn more balanced decision boundaries between the retained benign and attack classes. The MitM class was not included in

this balanced subset since it had a small sample size of 1,043 in the original sample, which was below the down sampling threshold. In addition, trends were analyzed on a feature-by-feature basis to look for time-based or sequential patterns that could suggest attack behavior. For instance, if the packet transmission rate fluctuates repeatedly or if the connection duration is abnormal, it could be a DoS attack or any other type of network abuse. In IoT scenarios, where devices typically produce periodic traffic, sudden traffic variations from the normal behavior can be a sign of malicious activity, especially relevant in this temporal behavior. In the multi-class classification experiment, 10,000 samples were used for each class, after down sampling 9 classes were retained, so 90,000 samples were used.

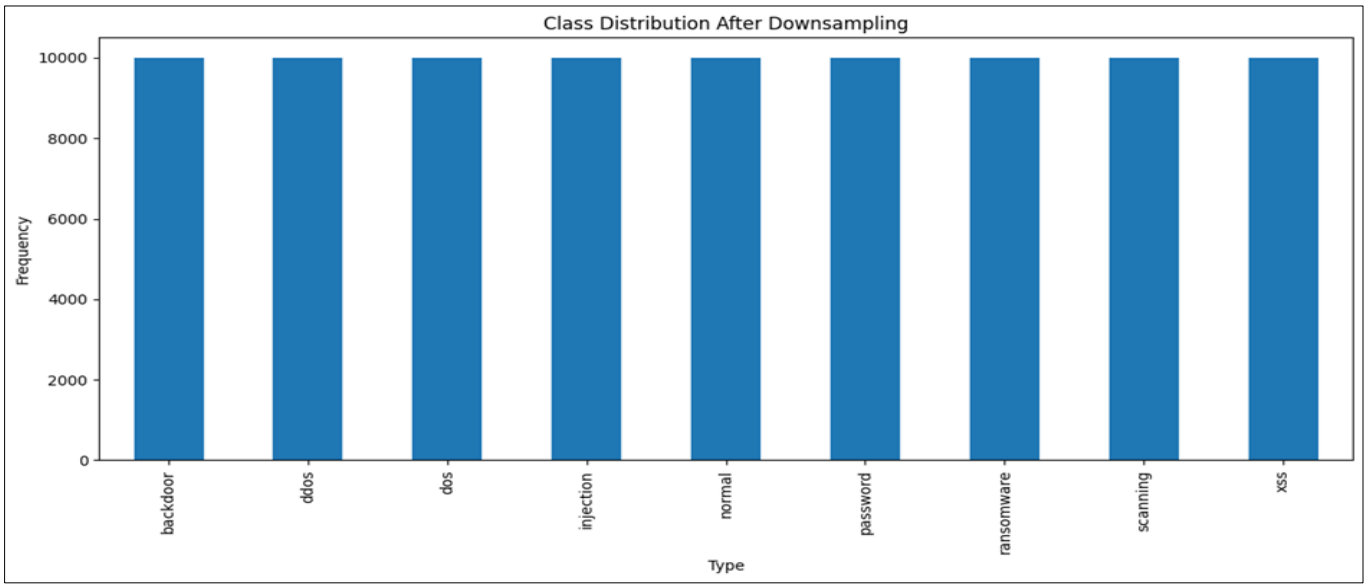

Figure 5. Class distribution after downsampling

3.2 Binary classification modeling

The binary classification stage [37, 38] aims to distinguish network traffic instances into two classes: benign and malicious. Given a preprocessed traffic sample x_i, the objective is to learn a mapping function $f(x_i) \rightarrow y_i$, where $y_i \in \{0,1\}$, with 0 representing benign traffic and 1 representing malicious traffic. The first step in the proposed ID framework is to determine if a traffic record represents normal activity or a possible cyberattack in this stage.

Once the data is preprocessed and EDA is performed, the cleaned data is split into training and testing sets in a stratified manner to maintain the same distribution of classes in both sets. Class balancing is applied to the training data only, as training data is often imbalanced, to prevent information leakage. Balancing is done by resampling methods like under-sampling and synthetic minority oversampling, if necessary, in this study. This way, the classifier will learn representative decision boundaries for both benign and malicious classes.

Then, feature selection is used to reduce the dimensionality and to eliminate redundant or weakly informative attributes. To keep the most discriminative features, recursive feature elimination (RFE) and model-based feature importance ranking are applied. This is crucial because network traffic data is usually high-dimensional, and irrelevant features can add computational complexity, noise, and decrease the generalization ability.

The following supervised ML models are considered for binary classification: DT, RF, XGBoost, GB. These models are chosen due to their capability of modeling non-linear relationships, their ability to process a heterogeneous feature space and their good performance in ID tasks. Interpretability, RF [39-41], DT [42-44] and GB [45-47] are the options. These models have unique characteristics in the field of cyber protection [48].

Hyperparameter tuning is done by systematically searching for the best parameters to enhance model performance [49-51]. The tuning process involves testing various combinations of parameters, including tree depth, number of estimators, learning rate, and splitting criteria, depending on the classifier. The best configuration is determined by the validation performance. To prevent overfitting and ensure the chosen model is robust, cross-validation is employed to test the model's performance on various subsets of the data.

Lastly, the accuracy, precision, recall and F1-score are used to evaluate the trained binary classifiers. Precision is the ability to lower the number of false alarms whereas Accuracy is measure of total accuracy of the classifier. Recall is particularly important for the cyber security topic, as an indicator of the ability of true malicious traffic identification by the model. The F1-score is a balanced measure that is suitable for the imbalanced ID cases.

For completeness of the methodology, the binary classification process is summarized in Algorithm 1.

Algorithm 1. Binary Classification Modeling Workflow

Input: Preprocessed network traffic dataset D, class labels $Y \in$ {benign,malicious}

Output: Best-performing binary classification model and evaluation metrics

1. Load the preprocessed network traffic dataset D.
2. Encode the target labels:
   benign → 0
   malicious → 1
3. Split D into training and testing sets using stratified sampling.

4. Apply class balancing only on the training set:
   a. Use under-sampling and/or SMOTE where required.
   b. Preserve the testing set unchanged for unbiased evaluation.

5. Perform feature selection:
   a. Apply Recursive Feature Elimination or feature-importance ranking.
   b. Retain the most informative features.

6. Define candidate classifiers:
   a. Decision Tree
   b. Random Forest
   c. XGBoost
   d. Gradient Boosting

7. For each classifier:
   a. Train the baseline model.
   b. Tune hyperparameters using the predefined search space.
   c. Validate the model using k-fold cross-validation.
   d. Store the best parameter configuration.

8. Evaluate the optimized model on the testing set using:
   a. Accuracy
   b. Precision
   c. Recall
   d. F1-score
   e. Confusion matrix

9. Select the best-performing classifier based on balanced performance,
   with particular attention to recall and F1-score.

10. Return the optimized binary classification model and final metrics.

3.3 Optimization of model performance using Salp Swarm Optimization

One of the main objectives of this study is to improve classifier performance using SSO. SSO is applied as a metaheuristic HO method to enhance the ability of the proposed ML models to distinguish between normal and anomalous network traffic [52-54]. Cybersecurity data sets tend to be sparse, noisy and imbalanced when high dimensionality, so in order to increase the predictive accuracy and decrease the computational cost selecting the optimal set of hyperparameters value is important.

SSO is based on salp swarming behavior in the ocean. In this algorithm, each salp is a candidate solution, with the location representing a set of hyperparameter values. The salp flock, part of which is on the leader-follower teams, is led to the food source and is tasked with searching for the new region of best solution, while the follower salps follows the front salp. The leader-follower technique makes the SSO exploratory to the new areas and intensive to high potential areas of solution.).

In this study, hyperparameters of the selected ML classifiers are tuned using SSO. The optimized parameters vary by model, and include the number of estimators, maximum tree depth, learning rate, splitting criterion, and minimum number of samples needed for splitting nodes. Each salp thus corresponds to one possible configuration of the hyperparameters. Each configuration is evaluated by training the classifier and measuring the validation performance.

The optimization objective is set to be the F1 score since ID needs to balance precision and recall. Precision is crucial to minimize false alarms, and recall is crucial to identify malicious traffic. Thus, maximizing the F1 score is a good metric for determining the best hyperparameter configuration. The optimization goal is given as:

$$\theta^* = \arg\max_{\theta \in \Theta} F1(M_\theta)$$

where, θ represents a candidate hyperparameter configuration, Θ denotes the hyperparameter search space, and M_θ is the classifier trained using configuration θ. Equivalently, the fitness function can be minimized as:

$$Fitness(\theta) = 1 - F1(M_\theta)$$

where, a lower fitness value indicates better classification performance.

The first step in the SSO process is to start with a population of salps in the hyperparameter search space defined. Cross validation is used to assess each salp on the training data. The best solution is chosen as the food source. In each iteration, the leader salp adjusts its position towards the food source, and the follower salps adjust their positions based on the salp chain mechanism. The updated values of the hyperparameters are constrained by the boundary control to be within the permissible range of the hyperparameter search. This process is repeated until the maximum number of iterations is reached or there is no further improvement.

Once the optimization process is done, the classifier is retrained with the optimal hyperparameter configuration. The optimized model is then tested on the independent test set with accuracy, precision, recall, F1-score and confusion matrix analysis. This is the final evaluation to test the optimized model on unseen data and to check the generalization capability of the optimized model.

Unlike the exhaustive search techniques like Grid Search, SSO does not need to test all possible combinations of hyperparameters. Instead, it uses a population-based search approach to more efficiently identify good solutions. This approach is very appropriate for cybersecurity classification problems where large feature spaces are typical and the population of data points can be large and thus computationally expensive.

To help clarify the methodology and to ensure reproducibility, the optimization process based on the SSO is summarized in Algorithm 2.

Algorithm 2. SSO-based Hyperparameter optimization

Input:
Training dataset D_{train}, validation strategy CV, classifier M, hyperparameter search space Θ, salp population size N, maximum number of iterations T

Output:
Best hyperparameter configuration θ^* and optimized classifier M_{θ^*}

1. Define the hyperparameter search space Θ for the selected classifier.

2. Initialize a population of N salps:

S = {S1, S2, ..., SN}

3. Assign each salp Si a candidate hyperparameter configuration θi.

4. For each salp Si:
   a. Train classifier M using θi on the training data.
   b. Evaluate M using cross-validation.
   c. Compute the fitness value:
      Fitness(θi) = 1 - F1(Mθi)

5. Identify the best candidate solution as the food source:
   θ* = argmin Fitness(θi)

6. For iteration t = 1 to T:

   a. Update the leader salp position based on the food source θ*.

   b. Update each follower salp position based on the position
      of the salp immediately ahead in the chain.

   c. Ensure that all updated hyperparameter values remain within
      the predefined search boundaries.

   d. For each updated salp:
      i. Train classifier M using the updated θi.
      ii. Evaluate the model using cross-validation.
      iii. Compute the new fitness value.

   e. If a better candidate solution is found:
      Update θ*.

7. Retrain the classifier using the best hyperparameter configuration θ*.

8. Evaluate the optimized classifier on the independent testing set.

9. Report accuracy, precision, recall, F1-score, and confusion matrix.

10. Return θ* and the optimized model Mθ*.

Table 1 presents the configuration parameters of the SSO algorithm integrated with the Optuna framework for hyperparameter tuning.

Table 1. Salp Swarm Optimization (SSO) algorithm configuration

Parameter	Value
Population size (number of salps)	30
Maximum iterations	100
Convergence criterion	No improvement for 20 consecutive iterations or maximum iterations reached
Objective function	Maximise classification accuracy via 10-fold cross-validation
Search strategy	SSO-guided exploration integrated with Optuna Tree-structured Parzen Estimator (TPE) sampler
Random seed	42 (for reproducibility)

Table 2. Hyperparameter search ranges

Model	Hyperparameter	Range	Type
Random Forest (RF)	n_estimators	[50, 500]	Integer
	max_depth	[3, 30]	Integer
	min_samples_split	[2, 20]	Integer
XGBoost	n_estimators	[50, 500]	Integer
	max_depth	[3, 12]	Integer
	learning_rate	[0.01, 0.3]	Log-uniform
	subsample	[0.6, 1.0]	Continuous
Gradient Boosting	n_estimators	[50, 500]	Integer
	max_depth	[3, 10]	Integer
	learning_rate	[0.01, 0.3]	Log-uniform
Decision Tree (DT)	min_samples_leaf	[1, 20]	Integer
	max_depth	[3, 30]	Integer
	min_samples_split	[2, 20]	Integer
	criterion	{gini, entropy}	Categorical
Extra Trees (ET)	n_estimators	[50, 500]	Integer
	max_depth	[3, 30]	Integer
	min_samples_split	[2, 20]	Integer

The hyperparameter search space for each ML model is defined in Table 2. Each hyperparameter was bounded within a predefined range to constrain the search to practically meaningful values.

Following SSO, the best-performing model for binary classification was the DT with the following optimal hyperparameters: max_depth = 18, min_samples_split = 3, criterion = entropy, achieving an accuracy of 99.45%. For multi-class classification, the Optimised ET achieved the best accuracy of 98% with n_estimators = 387, max_depth = 24, min_samples_split = 2.

3.4 Multi-class classification modeling

Multi-class classification is performed to identify the specific category of network traffic or attack type rather than only distinguishing between benign and malicious samples [55-57]. Unlike binary classification, where the output is limited to two classes, the multi-class setting assigns each input sample x_i to one of several predefined class labels $y_i \in \{C_1, C_2, \ldots, C_k\}$, where k represents the total number of traffic or attack categories in the dataset. This stage is also significant from the area of cyber security as class of network behavior can be precisely identified providing a better threat analysis and response.

Once preprocessed, the multi-class dataset is split into train and test data with a stratified sampling method. This implies that the distribution of all classes in the train and test data is kept consistent, decreasing the bias during the evaluation. In most of cybersecurity datasets, there is an imbalance in class distributions, meaning that some attack types occur more often than others. In this case, class balancing is only performed on the training data, either by resampling or class-weighting when necessary. The test data are left intact, so the evaluation is more accurate.

Feature selection is also performed prior to training of the classifier in order to reduce the dimensionality of the data set as much as possible while still maintaining the most informative features. This is especially critical when dealing with network traffic data sets, as the data can be redundant, noisy and weakly relevant, all of which slow down computation, can potentially lead to over-fitting and hinder

classification performance. Feature-importance ranking and selection techniques are used in network traffic analysis to determine the most significant attributes of network data for differentiating traffic classes from one another, and feature scaling is used where required to normalize feature ranges.

For the multi-class classification task, the supervised ML classifiers tested include DT, RF, ET, XGBoost, and GB. These classifiers are chosen as they are known to work well with high-dimensional tabular data and able to model nonlinear relationships between traffic features and class labels. Like for the binary-classification task, DT offers a baseline that is human-interpretable, RF and ET adds robustness due to ensemble learning, XGBoost and GB adds power to a weak learner with boosting, by training subsequent classifiers on samples that were falsely classified before.

For each classifier hyperparameter tuning is used in order to optimize the performance of the model for all the classes. The parameters used for each classifier are: number of estimators, maximum number of trees depth, learning rate, splitting criterion, minimum number of samples needed to split an internal node etc. Cross validation is applied in the training stage so overfitting is avoided and to check the validity of the proposed hyperparameters.

For the trained multi-class models the class-wise and aggregate performances are obtained. For each class the model performance measures how accurately the model detects each specific traffic class. The macro-average scores are used when the class-wise model performance has to be evaluated in a balanced way over all classes. When class sizes do differ, performance measures are calculated using weighted-averages. As further performance evaluating method confusion matrix analysis is used to analyze all correctly classified and misclassified instances and to recognize over predicted classes.

The entire multi-class classification process is summarized in Algorithm 3.

Algorithm 3. Multi-class classification modeling workflow

Input:
Preprocessed multi-class network traffic dataset D, class labels $Y = \{C_1, C_2, \ldots, C_k\}$
Output:
Best-performing multi-class classifier and evaluation metrics

1. Load the preprocessed multi-class network traffic dataset D.

2. Encode the target labels:
   Y = {C1, C2, ..., Ck}

3. Split D into training and testing sets using stratified sampling.

4. Handle class imbalance in the training set:
   a. Apply resampling techniques where required.
   b. Apply class weighting where supported by the classifier.
   c. Keep the testing set unchanged for unbiased evaluation.

5. Perform feature selection:
   a. Rank features using feature-importance or selection methods.
   b. Retain the most discriminative features.
   c. Remove redundant or weakly informative attributes.

6. Define candidate multi-class classifiers:
   a. Decision Tree
   b. Random Forest
   c. Extra Trees (ET)
   d. XGBoost
   e. Gradient Boosting

7. For each classifier:
   a. Train the baseline model on the selected features.
   b. Tune model-specific hyperparameters.
   c. Validate the model using k-fold cross-validation.
   d. Store the best hyperparameter configuration.

8. Retrain each optimized classifier using the full training set.

9. Evaluate each optimized model on the independent testing set using:
   a. Accuracy
   b. Class-wise precision
   c. Class-wise recall
   d. Class-wise F1-score
   e. Macro-average F1-score
   f. Weighted-average F1-score
   g. Confusion matrix

10. Compare the models based on predictive performance and training time.

11. Select the best-performing multi-class classifier.

12. Return the selected model and final evaluation results.

3.5 Evaluation metrics and cross-validation protocol

The assessment of the proposed ID models was carried out through the application of classification measures. A classification measure was preferred in this research project due to the fact that accuracy can provide misleading results for imbalanced data sets in the context of cyber security since the majority classes can contaminate the result while minority attack classes cannot be effectively found.

For binary classification, the confusion matrix consists of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Accuracy measures the proportion of correctly classified samples among all samples and is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision measures how many of the features predicted as positive are actually positive., and is an indicator of the accuracy of classifying the positive cases:

$$Precision = \frac{TP}{TP + FP}$$

Recall quantifies the percentage of relevant positive samples that are correctly retrieved:

$$Recall = \frac{TP}{TP + FN}$$

The F1-score is the harmonic mean of precision and recall and is defined by:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In the multi-class classification setting, each class is evaluated using a one-versus-rest strategy. For a given class c, TP_c represents the number of samples correctly classified as class c, FP_c represents samples incorrectly predicted as class c, and FN_c represents samples of class $c$incorrectly assigned to other classes. Therefore, the class-wise precision, recall, and F1-score are computed as:

$$Precision_c = \frac{TP_c}{TP_c + FP_c}$$

$$Recall_c = \frac{TP_c}{TP_c + FN_c}$$

$$F1_c = \frac{2 \times Precision_c \times Recall_c}{Precision_c + Recall_c}$$

For a dataset with K classes, the macro-average metric is calculated by giving equal importance to each class, regardless of its sample size. The macro-average F1-score is defined as:

$$Macro\text{-}F1 = \frac{1}{K}\sum_{c=1}^{K} F\,1_c$$

Similarly, macro-average precision and macro-average recall are calculated as:

$$Macro\text{-}Precision = \frac{1}{K}\sum_{c=1}^{K} Precisio\,n_c$$

$$Macro\text{-}Recall = \frac{1}{K}\sum_{c=1}^{K} Recal\,l_c$$

The weighted-average metric accounts for class support, where n_c is the number of samples belonging to class c, and N is the total number of samples. The weighted-average F1-score is computed as:

$$Weighted\text{-}F1 = \sum_{c=1}^{K} \frac{n_c}{N} F1_c$$

Weighted-average precision and recall are computed as:

$$Weighted\text{-}Precision = \sum_{c=1}^{K} \frac{n_c}{N} Precision_c$$

$$Weighted\text{-}Recall = \sum_{c=1}^{K} \frac{n_c}{N} Recall_c$$

The overall multi-class accuracy is computed from the confusion matrix as:

$$Accuracy = \frac{\sum_{c=1}^{K} T\,P_c}{N}$$

where, $\sum_{c=1}^{K} T\,P_c$ represents the total number of correctly classified samples across all classes.

In this study, macro-average metrics are used to evaluate whether the model performs consistently across all classes, including minority classes. Weighted-average metrics are also reported to reflect performance while considering the number of samples in each class. This distinction is important because a model may achieve a high weighted-average score by performing well on majority classes while still showing weaker performance on minority classes.

To improve the reliability of model evaluation, stratified k-fold cross-validation was used during model training and hyperparameter tuning [58]. In stratified k -fold cross-validation, the training dataset is divided into k folds while preserving the class distribution in each fold. During each iteration, $k - 1$ folds are used for training, and the remaining fold is used for validation. This process is repeated k times so that each fold is used once as the validation set. The final validation performance is obtained by averaging the metric values across all folds:

$$CV\text{-}Score = \frac{1}{k}\sum_{i=1}^{k} Scor\,e_i$$

where, $Score_i$ represents the validation score obtained in the i^{th} fold.

4. RESULTS AND DISCUSSION

In this section, the classification performance on the four models and five tuning techniques for binary and multiclass classifications are presented. The goal is to examine the predictive accuracy, precision, and recall of distinct models and optimism methods for hyperparameters in attaining optimal F1-scores. Metrics are shown in tables to give a clear understanding and to indicate the efficiency of the classifiers concerning the benchmark.

4.1 Binary classification results

The experiments carried out with samples that are aimed at binary classification provided high accuracy after the samples were classified into two classes for several ML models, especially after optimizing the hyperparameters of the models. Among the RF, XGBoost, and Gradient Boost models, all performed well on the accuracy criterion as well as reasonable on the other criteria.

Figures 6-11 provide a detailed view of the classification behavior of the evaluated models. In the binary classification task, RF and XGBoost show strong detection ability for Class 1, but both models produce different types of errors. RF generates a larger number of false positives, while XGBoost substantially reduces false negatives. GB provides a more balanced confusion matrix, although it misclassifies more Class 1 samples than XGBoost. The optimized DT achieves the best binary classification behavior, with only a small

number of misclassified samples, confirming the positive effect of HO.

For the multi-class task, ET shows stronger diagonal dominance than XGBoost, indicating more stable classification across the nine classes. ET performs particularly well for Classes 0, 4, and 7, but shows some confusion between Classes 5 and 0, Classes 6 and 8, and Classes 3 and 1. In contrast, XGBoost shows larger off-diagonal errors, especially for Class 5 predicted as Class 1, Class 8 predicted as Class 3, and Class 6 predicted as Class 1. These results show that overall accuracy alone is insufficient to evaluate multi-class ID performance, since different models may make errors on different traffic categories.

Table 3. Random forest performance

Metric	Class 0	Class 1	Macro Average	Weighted Average
Precision	0.99	0.85	0.92	0.92
Recall	0.82	0.99	0.90	0.90
F1-Score	0.90	0.91	0.90	0.90
Accuracy			0.90	

As demonstrated by Table 3, RF achieved an accuracy of 0.90, with a precision of 0.99 for Class 0 and 0.85 for Class 1. The high precision for Class 0 indicates that samples predicted as Class 0 were mostly correct. However, the recall for Class 0 was only 0.82, meaning that a considerable number of actual Class 0 samples were incorrectly classified as Class 1. This is also shown in Figure 6, where 5,769 Class 0 samples were misclassified as Class 1. In contrast, the recall for Class 1 reached 0.99, indicating that RF was highly effective in detecting Class 1 samples. Therefore, RF favors the detection of malicious traffic but at the cost of producing a higher number of false alarms.

Table 4. XGBoost performance

Metric	Class 0	Class 1	Macro Average	Weighted Average
Precision	1.00	0.88	0.94	0.94
Recall	0.87	1.00	0.93	0.93
F1-Score	0.93	0.94	0.93	0.93
Accuracy			0.93	

Table 5. Gradient Boosting performance

Metric	Class 0	Class 1	Macro Average	Weighted Average
Precision	0.91	0.93	0.92	0.92
Recall	0.93	0.91	0.92	0.92
F1-Score	0.92	0.92	0.92	0.92
Accuracy			0.92	

Overall, the XGBoost classifier led to an increased performance over the RF classifier as reported in Table 4. It obtained an accuracy of 0.93, macro-average precision, recall and F1-score values of 0.94, 0.93 and 0.93 respectively. Looking at the class-wise analysis, the classifier attained almost perfect precision for Class 0(1.00) and perfect recall for Class 1 (1.00) indicating that it rarely misclassified Class 1 samples as Class 0. This trend is clearly shown in Figure 7, where only 7 Class 1 samples were predicted as Class 0. However, 4,285 Class 0 samples were still misclassified as Class 1, indicating that XGBoost, like RF, still produces some false alarms. Compared with RF in Table 3 and Figure 6, XGBoost reduces false negatives substantially while also improving overall accuracy.

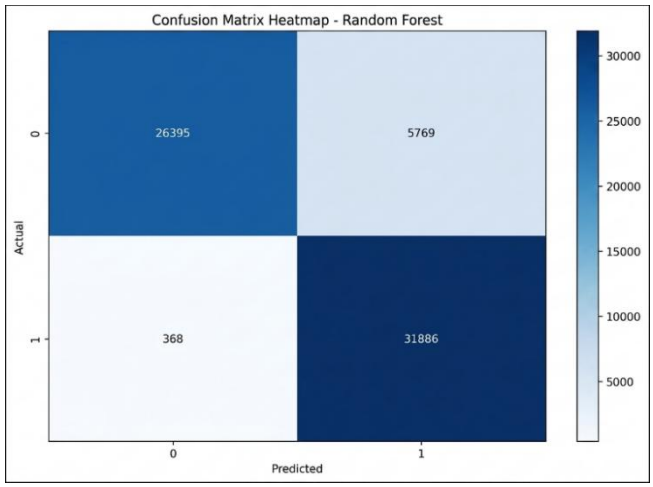

Figure 6. Confusion matrix of the Random Forest binary classifier

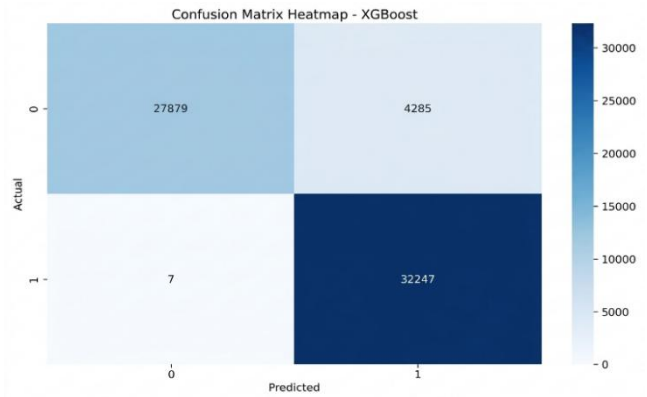

Figure 7. Confusion matrix of the XGBoost binary classifier

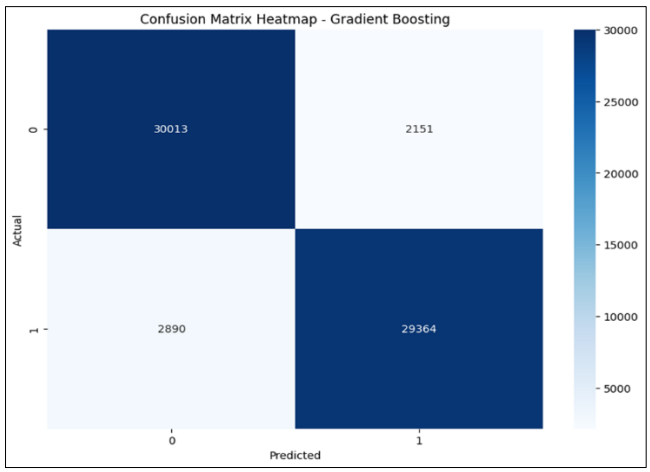

Figure 8. Gradient Boosting binary classification confusion matrix

The class-wise behaviour of the GB classifier is more balanced (a typical result as this is a very flexible in terms of classification models), as summarized by the macro-average scores in Table 5. There is a classification accuracy of 0.92 with macro-average precision, recall and F1-score values close to that of 0.92. Unlike the RF and XGBoost, this proposed model appears to be more symmetrical in its classification

performance, having Class 0 precision and recall of 0.91 and 0.93 respectively, whilst Class 1 precision and recall of 0.93 and 0.91 respectively. By inspection of the confusion matrix (Figure 8), this interpretation appears to be correct; the model correctly classifies 30013 of Class 0 and 29364 of Class 1. However, it misclassifies 2151 of Class 0 as Class 1 and 2890 of Class 1 as Class 0 – reducing the false positive result against the RF and XGBoost, but increasing the false negative result against XGBoost. This may be a relevant trade-off in ID to miss more malicious traffic.

Table 6. Optimized Decision Tree (DT) performance

Metric	Class 0	Class 1	Macro Average	Weighted Average
Precision	1.00	0.99	**0.99**	**0.99**
Recall	0.99	1.00	**0.99**	**0.99**
F1-Score	0.99	0.99	**0.99**	**0.99**
Accuracy			**0.99**	

Optimization proved quite effective with the tuned hyperparameter DT model, recording 0.99 accuracy, which is an indication of highly effective optimization.

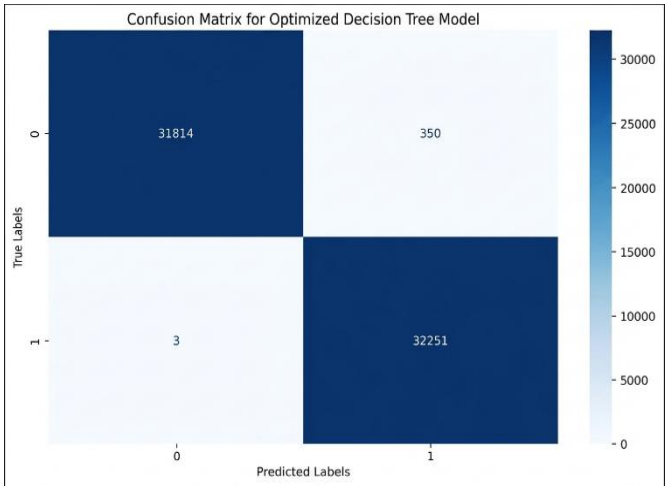

Figure 9. Confusion matrix of the optimized Decision Tree binary classifier

The optimized DT performed the best in binary classification as indicated in Table 6 and Figure 9. The accuracy, macro-average and weighted-average precision, recall and F1-score after HO are all close to 0.99. The confusion matrix of Figure 9 shows a very strong diagonal dominance, 31814 Class 0 and 32251 Class 1 samples were correctly classified, and only 350 Class 0 and 3 Class 1 samples were incorrectly classified. Compared with Tables 1-3 and Figures 6-8 for the previous models with no HO, the optimized DT substantially decreased the number of false positives and false negatives, reflecting the positive impact of HO on binary ID.

Overall on the binary classification, the major differences in performance of the models are how they deal with the false positives and false negatives. Both RF and XGBoost were very sensitive to Class 1, but both mistakenly made false positives with some Class 0 samples in Class 1 (Figures 6 and 7). While GB was a little more balanced in its classification, it missed more Class 1 samples (Figure 8). The best possible DT constructed through hyperparameter tuning was the best overall at minimizing false positives and false negatives (Figure 9). These results seem to show that HO had a definite beneficial effect on binary classification, especially in terms of precision and recall.

4.2 Multiclass classification results

The multiclass classification experiments investigate the proficiency of the models to classify traffic into 9 classes. Class-wise results for ET and XGBoosts are listed in Table 5 and 6 respectively; the confusion matrices for ET and XGBoosts are shown in Figures 10 and 11 respectively. With multiclass classification task, it is more difficult than the binary classification task as the model has to not only detect abnormal traffic but also put the sample into correct traffic category or attack category.

As indicated in Table 7, the ET classifier achieved an accuracy of 0.94, macro-average and weighted-average precision, recall and F1-score values of 0.94. The macro-average and weighted-average were largely similar, indicating that, compared to being possibly overfitted to the major classes, the model was capable of capturing common clues to distinguish classes in general. In other words, the model was able to identify classes with relative stability rather than proximity within the three classes. Looking at class-wise, ET performed really well for Class 4 and Class 7, both achieved very good F1 score values of 0.99. Class 2 obtained very good F1 score value of 0.99, indicating class 2 was highly separable from others. Highest F1 score among Class 5 is 0.84 for recall and Class 8 is 0.88 for precision.

Figure 10 shows a confusion matrix for the ET case that shows class-level behaviors. It can be seen that there are a few very strong diagonal classes, but there are also some class confusion lines. For instance, some of the Class 5 samples were predicted as class 0, some of the Class 6 samples were predicted as class 8 and some of the Class 3 samples were predicted as class 1. These lines indicate that traffic classes may sometimes have the same feature patterns. Therefore, although ET achieved strong overall performance in Table 7, Figure 10 shows that class-specific overlap still exists.

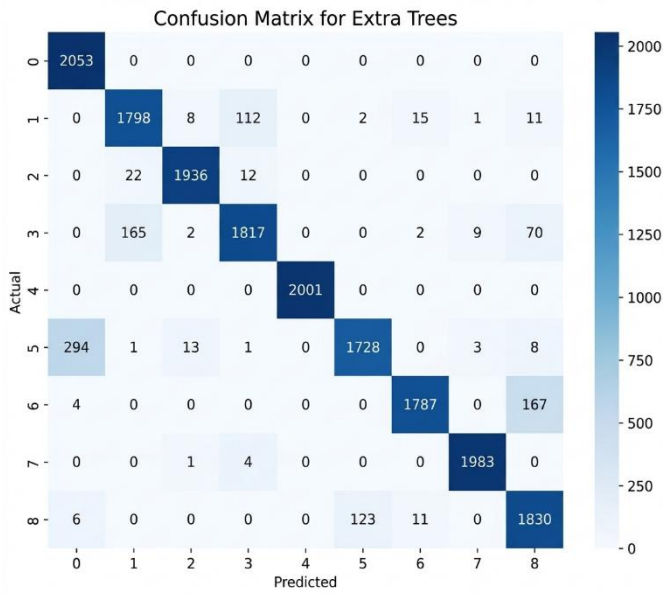

Figure 10. Confusion matrix of the extra trees multi-class classifier

Table 7. Extra trees performance

Metric	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Macro Average	Weighted Average
Precision	0.87	0.91	0.99	0.93	1.00	0.93	0.98	0.99	0.88	**0.94**	**0.94**
Recall	1.00	0.92	0.98	0.88	1.00	0.84	0.91	1.00	0.93	**0.94**	**0.94**
F1-Score	0.93	0.91	0.99	0.91	1.00	0.89	0.95	1.00	0.90	**0.94**	**0.94**
Accuracy							**0.94**				

Table 8. XGBoost performance

Metric	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Macro Average	Weighted Average
Precision	1.00	0.60	1.00	0.77	1.00	0.96	0.96	1.00	0.98	**0.92**	**0.92**
Recall	0.99	0.92	0.91	0.91	1.00	0.74	0.84	1.00	0.74	**0.89**	**0.89**
F1-Score	1.00	0.73	0.95	0.83	1.00	0.84	0.90	1.00	0.84	**0.90**	**0.90**
Accuracy							**0.89**				

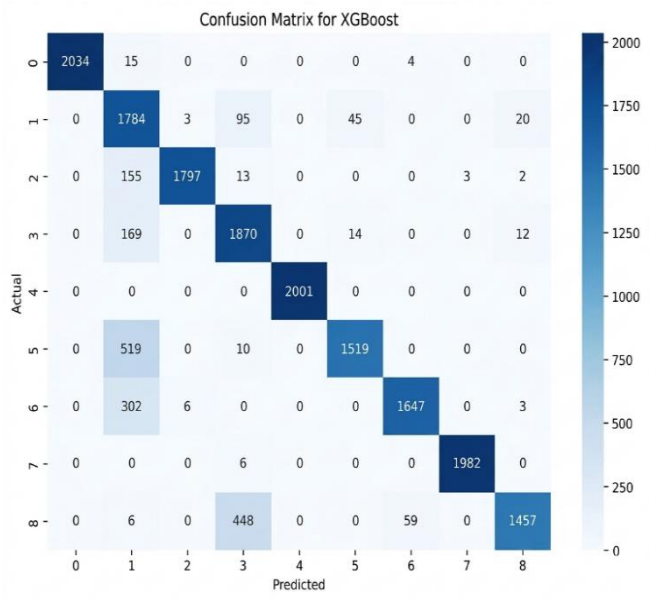

Figure 11. XGBoost multi-class classification confusion matrix

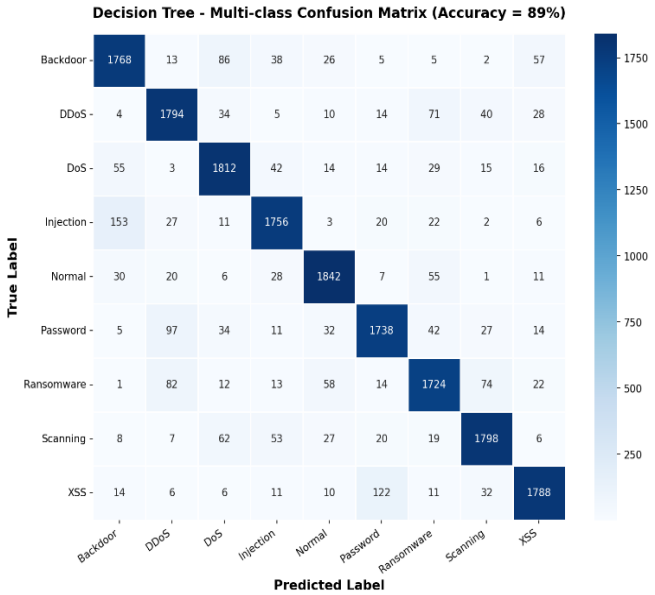

Figure 12. Confusion matrix of the decision tree multi-class classifier

Table 8 and Figure 11 show the results of the XGBoost multiclass model. The overall accuracy of XGBoost was 0.89, while the macro-average precision, recall, and F1-score were 0.92, 0.89, and 0.90, respectively. The results for each class demonstrate that XGBoost performed very well for Class 0, Class 4 and Class 7. It did not perform as well, however, in certain classes, notably Class 1, Class 5 and Class 8. The precision of Class 1 was particularly low (0.60), meaning that many samples from other classes were misclassified as Class 1. The recall value of Class 5 and Class 8 were also relatively low, with 0.74 samples from each class being assigned to other classes.

The confusion matrix in Figure 11 provides an explanation for these lower class-wise scores. Many Class 5 samples were predicted as Class 1 and many Class 8 samples were predicted as Class 3. In addition, several Class 6 samples were also predicted as Class 1. The misclassification patterns indicate that XGBoost had difficulties with certain class boundaries, especially in regions of feature patterns shared between traffic categories. However, the overall performance of XGBoost was competitive, and the class-wise performance in Table 8 and the error distribution in Figure 11 shows that its ability to separate the classes was less than that of ET.

Multi-class Performance of the DT classifier is given in Table 9 and the confusion matrix in Figure 12. Overall accuracy was 0.89 with macro-average and weighted-average precision, recall and F1-score all 0.89. As each class has 2,000 test samples, equality between the macro and weighted averages indicates that the performance that is reported is not due to class-frequency variations in the test set. As can be seen in the per-class results, the DT achieved its highest performance on the Normal class, with an F1-score of 0.92 followed by XSS and Scanning with F1-scores of 0.91 and 0.90 respectively. DT performed slightly worse on Ransomware, Backdoor and Password with all F1-scores being in the range 0.87-0.88.

The detailed class-wise difference can be further understood from the true and predicted class relationship in the DT Confusion Matrix (Figure 12). Although most samples are correctly predicted in the true class diagonal, Class-wise observation of misclassification is also interesting. In total, 153 Injection flows are predicted as Backdoor, 122 XSS are predicted as Password and 97 Password are predicted as DDoS. Other prediction examples include the confusion of Ransomware with DDoS, Normal and Scanning. These errors imply some classes of traffic probably have similar features which cannot be distinguished by a single tree classifier such

as the DT.

The results of the RF classifier are displayed in Table 10 and the confusion matrix is displayed in Figure 13. RF had an overall accuracy of 0.93, which was about 4 percentage points higher than that of DT. It also obtained macro-average and weighted-average precision, recall, and F1-score values of 0.93, which shows that it performs well in all nine classes. The highest class-wise F1 scores were 0.95 for both Backdoor and Injection, with most other classes scoring between 0.92 and 0.93. XSS had the lowest F1 score of 0.91, but this is still a good classification performance.

Table 9. Decision Tree (DT) multi-class per-class performance

Metric	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Macro Average	Weighted Average
Precision	0.87	0.88	0.88	0.90	0.91	0.89	0.87	0.90	0.92	**0.89**	**0.89**
Recall	0.88	0.90	0.91	0.88	0.92	0.87	0.86	0.90	0.89	**0.89**	**0.89**
F1-Score	0.88	0.89	0.89	0.89	0.92	0.88	0.87	0.90	0.91	**0.89**	**0.89**
Accuracy							**0.89**				

Table 10. Random Forest (RF) multi-class per-class performance

Metric	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Macro Average	Weighted Average
Precision	0.97	0.92	0.91	0.97	0.90	0.94	0.92	0.93	0.90	**0.93**	**0.93**
Recall	0.93	0.94	0.94	0.92	0.95	0.92	0.91	0.94	0.93	**0.93**	**0.93**
F1-Score	0.95	0.93	0.93	0.95	0.93	0.93	0.92	0.93	0.91	**0.93**	**0.93**
Accuracy							**0.93**				

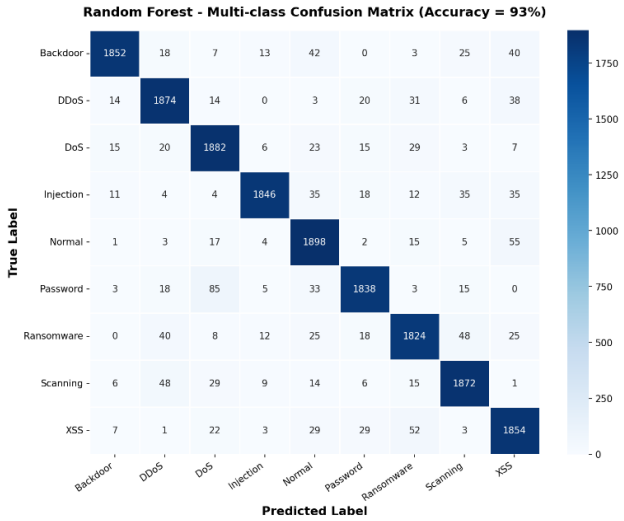

Figure 13. Confusion matrix of the Random Forest multi-class classifier

The RF confusion matrix (Figure 13) has more diagonal dominance and less severe misclassification patterns than the DT confusion matrix (Figure 12). This improvement is anticipated as the RF algorithm is a combination of several DTs and lessens the instability of individual tree decisions. RF correctly classified 1898 Normal samples, 1882 DoS samples, 1874 DDoS samples and 1872 Scanning samples out of 2000 samples per class. But there is still some confusion. For example, 85 Password samples were predicted as DoS, 55 Normal samples were predicted as XSS and 52 XSS samples were predicted as Ransomware. The residual errors show that even ensemble models can have problems if the traffic patterns of different attack categories are similar.

The summary results in Table 11 further show the differences among the multiclass models. ET achieved 0.94 accuracy and F1-score, while RF achieved 0.93 accuracy and 0.93 F1-score. DT established an accuracy of 0.89, with a more modest 0.85 F1-score, indicating less balanced class-wise behavior. The parameter tuned GB showed the weakest performance in terms of accuracy being 0.43 and an even weaker F1 of 0.42 and was deemed inappropriate for this multiclass setting given the conditions already reported in the experiment. The best reported performance enabled ET, which was tuned with the parameters set out in Table 11 to follow up with an accuracy, precision, recall and F1-score of 0.98.

This comparison of the baseline ET in Table 7 with the optimized ET in Table 11 shows the effect of HO, with accuracy rising from 0.94 to 0.98 and F1-score rising from 0.94 to 0.98. This indicates that optimization improved both correctness and class balance. In multiclass ID this is particularly important since high overall accuracy masks the fact that the model may perform poorly on minority or overlapping classes. Figures 10 and 11 (the confusion matrices for the baseline and optimal ET respectively) combined with the macro-average results in Tables 5 and 6 demonstrate that a per-class measure is necessary to judge reliability.

Table 11. Summary of results

Model	Accuracy	F1 Score	Recall	Precision	Training Time (s)
Extra Trees	0.94	0.94	0.94	0.94	2.107
Random Forest	0.93	0.93	0.93	0.94	0.468
Decision Tree (DT)	0.89	0.85	0.89	0.83	0.176
Gradient Boosting	0.43	0.42	0.43	0.68	0.702
Optimized Extra Trees	0.98	0.98	0.98	0.98	1.887

In total, the multiclass results show ET had more consistent class-wise accuracy than XGBoost, but that ET with optimization outperformed all other reported multiclass models. The results also highlight how class imbalance and

class overlap play a role; many classes were predicted with near perfect precision or recall, but others had lower metrics due to their feature-pattern being more similar to other classes. So the results support using both aggregate metrics and class-wise metrics when measuring multiclass IDSs.

4.3 Sensitivity and ablation analysis

To verify the stability of the proposed models and to identify the influence of the primary pipeline components, three sensitivity and ablation experiments were undertaken. An analysis was performed on the binary classification task with the use of the SSO-optimized DT (due to its ranking as the model with the highest binary classification accuracy). Other classifiers (RF, XGBoost and GB) were compared as well to ascertain if the aforesaid outcomes would be specific to the classifier.

4.3.1 Effect of training set size

The SSO-optimized DT was trained with increasing portions of the binary training set (20%, 40%, 60%, 80%, and 100%) to analyze learning behavior and data efficiency (shown in Table 12). Training was repeated using one independent testing set for all the test.

The SSO-optimized DT showed the promising data efficiency with 96.21% as the final test accuracy using only 20% of training samples. All the classifiers experienced the improvement when more training-set was provided, which indicated the increase in training samples improved the generalization performance. The improvement was gradually less from 80% of training samples because the feature space was well sampled in that area. The optimized DT performed consistently better than the other two classifiers using all fraction of training samples, which supported the effectiveness of the SSO-based tuning strategy.

4.3.2 Effect of class imbalance ratio

Table 13 shows the change in optimized DT with respect to changes in class imbalance. It can be seen that as the benign sample ratio grows higher, recall begins dropping off much steeper than precision, showing the trend that under imbalanced training data, the model will have increasingly more difficulty in missing malicious samples.

It is seen from the figures that the class balancing significantly enhances the quality of the SSO-optimized DT. While the precision holds up reasonably well across various ratios, recall drops with increasing imbalance. It confirms that the models continuing to be accurate in predicting positive samples, but it becomes less capable of identifying all malicious samples as the benign class constitute the majority of training set. This finding supports the use of down sampling as a preprocessing step because recall is especially important in ID, where missed attacks may have serious security consequences.

Table 12. Model accuracy vs. training set size

Training Fraction	Training Samples	Decision Tree (SSO)	Random Forest	XGBoost	Gradient Boosting
20%	51,534	96.21%	86.13%	88.74%	87.52%
40%	103,067	97.85%	88.41%	90.87%	89.63%
60%	154,601	98.73%	89.25%	91.94%	91.08%
80%	206,134	99.12%	89.78%	92.61%	91.72%
100%	257,668	99.45%	90.00%	93.00%	92.00%

Note: SSO: Salp Swarm Optimization.

Table 13. Effect of class imbalance ratio on Salp Swarm Optimization (SSO)-Decision Tree (DT) performance

Ratio (Malicious: Benign)	Malicious Training Samples	Benign Training Samples	Total Training Samples	Accuracy	Precision	Recall	F1-Score
1:1 balanced	128,834	128,834	257,668	99.45%	0.99	0.99	0.99
1:1.25	128,834	161,043	289,877	98.87%	0.99	0.97	0.98
1:1.50	128,834	193,251	322,085	97.94%	0.98	0.95	0.96
1:1.75	128,834	225,460	354,294	97.38%	0.98	0.94	0.96
Original 1:1.86	128,834	240,000	368,834	96.52%	0.98	0.91	0.94

5. CONCLUSION

This study evaluated the effect of hyperparameter tuning on tree-based ML models for ID under binary and multi-class classification settings. The evaluated models included DT, RF, XGBoost, GB, and ET. The results indicate that the performance of the models differed for the classification task and the evaluation metric used, such as accuracy, precision, recall and F1-score.

The optimized DT model showed the best performance with an accuracy of 99.45% and balanced precision, recall and F1-score values between the two classes for binary classification. The result shows that the HO enhanced the model's capability of differentiating between benign and malicious network traffic in the dataset used in this study. The optimized model also outperformed the non-optimized classifiers in terms of misclassification, highlighting the need for systematic parameter tuning in ID problems.

The best overall performance was obtained by the optimized ET classifier for multi-class classification with 98.41% accuracy. This finding indicates that ensemble-based tree models can be effective classifiers for multiple traffic or attack categories, if the hyperparameters are optimized. The results also indicate that the performance of the models varies among classifiers. For instance, XGBoost and GB were found to be helpful for prediction, but they came with different compromises in terms of accuracy, class-wise performance and computational cost.

In conclusion, the results show that HO can enhance the effectiveness of tree-based ML models in IoT IDS. Optimized DT and ET models were the best performing models in the binary and multi-class experiments, respectively. The results

presented in this study are valid for the data set, the preprocessing approach, the classifiers selected and the optimization parameters. Hence, the results cannot be extrapolated to other application areas without further testing.

The proposed optimization framework should be tested on other datasets of ID with various traffic distributions, attack types, and class-imbalance levels in future work. Other investigations could also examine the robustness with respect to different dataset sizes, different feature selection and sampling techniques. For the future work, the authors can explore the comparison between SSO-based tuning and some other meta-heurstic or Bayesian optimization approaches to examine the trade-off in predictive performance versus computational cost.

REFERENCES

[1] Sharief, F., Ijaz, H., Shojafar, M., Naeem, M.A. (2024). Multi-class imbalanced data handling with concept drift in fog computing: A taxonomy, review, and future directions. ACM Computing Surveys, 57(1): 1-48. https://doi.org/10.1145/3689627

[2] Alex, C., Creado, G., Almobaideen, W., Alghanam, O.A., Saadeh, M. (2023). A comprehensive survey for IoT security datasets taxonomy, classification and machine learning mechanisms. Computers & Security, 132: 103283. https://doi.org/10.1016/j.cose.2023.103283

[3] Aboubakar, M., Kellil, M., Roux, P. (2022). A review of IoT network management: Current status and perspectives. Journal of King Saud University-Computer and Information Sciences, 34(7): 4163-4176. https://doi.org/10.1016/j.jksuci.2021.03.006

[4] Gržinić, T., González, E.B. (2022). Methods for automatic malware analysis and classification: A survey. International Journal of Information and Computer Security, 17(1-2): 179-203. https://doi.org/10.1504/IJICS.2022.121297

[5] Felix, E.A., Lee, S.P. (2019). Systematic literature review of preprocessing techniques for imbalanced data. Iet Software, 13(6): 479-496. https://doi.org/10.1049/iet-sen.2018.5193

[6] Akinola, O.O., Ezugwu, A.E., Agushaka, J.O., Zitar, R.A., Abualigah, L. (2022). Multiclass feature selection with metaheuristic optimization algorithms: A review. Neural Computing and Applications, 34(22): 19751-19790. https://doi.org/10.1007/s00521-022-07705-4

[7] Naidu, G., Zuva, T., Sibanda, E.M. (2023). A review of evaluation metrics in machine learning algorithms. In Computer Science On-Line Conference, pp. 15-25. https://doi.org/10.1007/978-3-031-35314-7_2

[8] Maxwell, A.E., Warner, T.A., Guillén, L.A. (2021). Accuracy assessment in convolutional neural network-based deep learning remote sensing studies—Part 1: Literature review. Remote Sensing, 13(13): 2450. https://doi.org/10.3390/rs13132450

[9] Raschka, S. (2014). An overview of general performance metrics of binary classifier systems. arXiv preprint arXiv:1410.5330. https://doi.org/10.48550/arXiv.1410.5330

[10] Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., Faruki, P. (2019). Network intrusion detection for IoT security based on learning techniques. IEEE Communications Surveys & Tutorials, 21(3): 2671-2701. https://doi.org/10.1109/COMST.2019.2896380

[11] Waghmode, P., Kanumuri, M., El-Ocla, H., Boyle, T. (2025). Intrusion detection system based on machine learning using least square support vector machine. Scientific Reports, 15(1): 12066. https://doi.org/10.1038/s41598-025-95621-7

[12] Poornima, P., Kanimozhi, J.K., Loganathan, E., Kandasamy, C.A., Yoganathan, A. (2025). Advanced intrusion detection system leveraging support vector machine algorithm. In 2025 5th International Conference on Pervasive Computing and Social Networking (ICPCSN), Salem, India, pp. 955-959. https://doi.org/10.1109/ICPCSN65854.2025.11035630

[13] Tong, J., Zhang, Y. (2024). A real-time label-free self-supervised deep learning intrusion detection for handling new type and few-shot attacks in IoT networks. IEEE Internet of Things Journal, 11(19): 30769-30786. https://doi.org/10.1109/JIOT.2024.3414492

[14] Ajitha, I., Devi, A. (2025). Optimizing deep feedforward neural networks for effective classification and prediction of cyber incidents. In 2025 1st International Conference on Secure IoT, Assured and Trusted Computing (SATC), Dayton, OH, USA, pp. 1-9. https://doi.org/10.1109/SATC65530.2025.11137131

[15] Al Hanif, A., Ilyas, M. (2024). Effective feature engineering framework for securing MQTT protocol in IoT environments. Sensors, 24(6): 1782. https://doi.org/10.3390/s24061782

[16] Qazi, E.U.H., Faheem, M.H., Zia, T. (2023). HDLNIDS: Hybrid deep-learning-based network intrusion detection system. Applied Sciences, 13(8): 4921. https://doi.org/10.3390/app13084921

[17] Vaccari, I., Chiola, G., Aiello, M., Mongelli, M., Cambiaso, E. (2020). MQTTset, a new dataset for machine learning techniques on MQTT. Sensors, 20(22): 6578. https://doi.org/10.3390/s20226578

[18] Koroniotis, N., Moustafa, N., Sitnikova, E., Turnbull, B. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset. Future Generation Computer Systems, 100: 779-796. https://doi.org/10.1016/j.future.2019.05.041

[19] Soe, Y.N., Feng, Y., Santosa, P.I., Hartanto, R., Sakurai, K. (2019). Rule generation for signature based detection systems of cyber attacks in IoT environments. Bulletin of Networking, Computing, Systems, and Software, 8(2): 93-97. http://bncss.org/index.php/bncss/article/view/113.

[20] Baig, Z.A., Sanguanpong, S., Firdous, S.N., Vo, V.N., Nguyen, T.G., So-In, C. (2020). Averaged dependence estimators for DoS attack detection in IoT networks. Future Generation Computer Systems, 102: 198-209. https://doi.org/10.1016/j.future.2019.08.007

[21] Akuthota, U.C., Bhargava, L. (2025). Transformer-based intrusion detection for IoT networks. IEEE Internet of Things Journal, 12(5): 6062-6067. https://doi.org/10.1109/JIOT.2025.3525494

[22] Hnamte, V., Najar, A.A., Laldinsanga, C., Hussain, J., Hmingliana, L. (2025). A lightweight intrusion detection system using deep convolutional neural network. Computers and Electrical Engineering, 127: 110561. https://doi.org/10.1016/j.compeleceng.2025.110561

[23] Farhan, M., Waheed Ud Din, H., Ullah, S., Hussain,

M.S., et al. (2025). Network-based intrusion detection using deep learning technique. Scientific Reports, 15(1): 25550. https://doi.org/10.1038/s41598-025-08770-0

[24] Doost, P.A., Moghadam, S.S., Khezri, E., Basem, A., Trik, M. (2025). A new intrusion detection method using ensemble classification and feature selection. Scientific Reports, 15(1): 13642. https://doi.org/10.1038/s41598-025-98604-w

[25] Alsubaei, F.S. (2025). Smart deep learning model for enhanced IoT intrusion detection. Scientific Reports, 15(1): 20577. https://doi.org/10.1038/s41598-025-06363-5

[26] Susan, S., Kumar, A. (2021). The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent state of the art. Engineering Reports, 3(4): e12298. https://doi.org/10.1002/eng2.12298

[27] Alzaabi, F.R., Mehmood, A. (2024). A review of recent advances, challenges, and opportunities in malicious insider threat detection using machine learning methods. IEEE Access, 12: 30907-30927. https://doi.org/10.1109/ACCESS.2024.3369906

[28] Macas, M., Wu, C., Fuertes, W. (2022). A survey on deep learning for cybersecurity: Progress, challenges, and opportunities. Computer Networks, 212: 109032. https://doi.org/10.1016/j.comnet.2022.109032

[29] Alshdaifat, E.A., Alshdaifat, D.A., Alsarhan, A., Hussein, F., El-Salhi, S.M.D.F.S. (2021). The effect of preprocessing techniques, applied to numeric features, on classification algorithms' performance. Data, 6(2): 11. https://doi.org/10.3390/data6020011

[30] Santos, K.C., Miani, R.S., de Oliveira Silva, F. (2024). Evaluating the impact of data preprocessing techniques on the performance of intrusion detection systems. Journal of Network and Systems Management, 32(2): 36. https://doi.org/10.1007/s10922-024-09813-z

[31] Alghamdi, T.A., Javaid, N. (2022). A survey of preprocessing methods used for analysis of big data originated from smart grids. IEEE Access, 10: 29149-29171. https://doi.org/10.1109/ACCESS.2022.3157941

[32] Raparthi, M., Gayam, S.R., Kasaraneni, B.P., Kondapaka, K.K., et al. (2024). Exploratory data analysis techniques—A comprehensive review: Reviewing various exploratory data analysis techniques and their applications in uncovering insights from raw data. Australian Journal of Machine Learning Research & Applications, 4(1): 215-225.

[33] Oettl, F.C., Oeding, J.F., Feldt, R., Ley, C., Hirschmann, M.T., Samuelsson, K., ESSKA Artificial Intelligence Working Group. (2024). The artificial intelligence advantage: Supercharging exploratory data analysis. Knee Surgery, Sports Traumatology, Arthroscopy, 32(11): 3039-3042. https://doi.org/10.1002/ksa.12389

[34] Pinto, A.S., Pato, M., Datia, N. (2024). Enhancing drug reviews insights through exploratory data analysis and sentiment analysis. In 2024 28th International Conference Information Visualisation (IV), Coimbra, Portugal, pp. 190-195. https://doi.org/10.1109/IV64223.2024.00042

[35] Paz-Robles, M., Gomez-Santillan, C., Rangel-Valdez, N., Morales-Rodriguez, M.L., Castillo-Valdez, G. (2024). Analysis of accuracy on data visualization techniques for multi-objective algorithm performance based on convergence and diversity towards the pareto frontier. Artificial Intelligence in Prescriptive Analytics: Innovations in Decision Analysis, 260: 245-274. https://doi.org/10.1007/978-3-031-66731-2_10

[36] Mahboubi, A., Luong, K., Aboutorab, H., Bui, H.T., et al. (2024). Evolving techniques in cyber threat hunting: A systematic review. Journal of Network and Computer Applications, 232: 104004. https://doi.org/10.1016/j.jnca.2024.104004

[37] Hayashi, T., Cimr, D., Fujita, H., Cimler, R. (2026). Critical review for one-class classification: Recent advances and reality behind them. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 16(1): e70058. https://doi.org/10.1002/widm.70058

[38] Agarwal, P., Wazid, M., Singh, V.K., Srivastava, A., Singh, A.R., Mittal, S., Das, A.K. (2025). BCIDS-IoT: A binary classification intrusion detection scheme for internet of things communication. Security and Privacy, 8(2): e475. https://doi.org/10.1002/spy2.475

[39] Jiao, M. (2025). Application of random forest algorithm in network intrusion detection of government affairs departments. International Journal of Computational Intelligence and Applications, 24(4): 2342003. https://doi.org/10.1142/S1469026823420038

[40] Iranzad, R., Liu, X. (2025). A review of random forest-based feature selection methods for data science education and applications. International Journal of Data Science and Analytics, 20(2): 197-211. https://doi.org/10.1007/s41060-024-00509-w

[41] Shaik, A.B., Srinivasan, S. (2018). A brief survey on random forest ensembles in classification model. In International Conference on Innovative Computing and Communications: Proceedings of ICICC 2018, Volume 2, pp. 253-260. https://doi.org/10.1007/978-981-13-2354-6_27

[42] Li, H., Song, J., Xue, M., Zhang, H., Song, M. (2024). A survey of neural trees: Co-evolving neural networks and decision trees. IEEE Transactions on Neural Networks and Learning Systems, 36(7): 11718-11737. https://doi.org/10.1109/TNNLS.2024.3446891

[43] Kingsford, C., Salzberg, S.L. (2008). What are decision trees? Nature Biotechnology, 26(9): 1011-1013. https://doi.org/10.1038/nbt0908-1011

[44] Kotsiantis, S.B. (2013). Decision trees: A recent overview. Artificial Intelligence Review, 39(4): 261-283. https://doi.org/10.1007/s10462-011-9272-4

[45] Mayr, A., Binder, H., Gefeller, O., Schmid, M. (2014). The evolution of boosting algorithms. Methods of Information in Medicine, 53(6): 419-427. https://doi.org/10.3414/ME13-01-0122

[46] Natekin, A., Knoll, A. (2013). Gradient boosting machines, a tutorial. Frontiers in Neurorobotics, 7: 63623. https://doi.org/10.3389/fnbot.2013.00021

[47] Azmi, S.S., Baliga, S. (2020). An overview of boosting decision tree algorithms utilizing AdaBoost and XGBoost boosting strategies. International Research Journal of Engineering and Technology, 7(5): 6867-6870.

[48] Sarker, I.H., Kayes, A.S.M., Badsha, S., Alqahtani, H., Watters, P., Ng, A. (2020). Cybersecurity data science: An overview from machine learning perspective. Journal of Big Data, 7(1): 41. https://doi.org/10.1186/s40537-020-00318-5

[49] Ali, Y.A., Awwad, E.M., Al-Razgan, M., Maarouf, A. (2023). Hyperparameter search for machine learning

algorithms for optimizing the computational complexity. Processes, 11(2): 349. https://doi.org/10.3390/pr11020349

[50] Yang, L., Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415: 295-316. https://doi.org/10.1016/j.neucom.2020.07.061

[51] Hanifi, S., Cammarono, A., Zare-Behtash, H. (2024). Advanced hyperparameter optimization of deep learning models for wind power prediction. Renewable Energy, 221: 119700. https://doi.org/10.1016/j.renene.2023.119700

[52] Castelli, M., Manzoni, L., Mariot, L., Nobile, M.S., Tangherloni, A. (2022). Salp swarm optimization: A critical review. Expert Systems with Applications, 189: 116029. https://doi.org/10.1016/j.eswa.2021.116029

[53] Abualigah, L., Shehab, M., Alshinwan, M., Alabool, H. (2020). SALP swarm algorithm: A comprehensive survey. Neural Computing and Applications, 32(15): 11195-11215. https://doi.org/10.1007/s00521-019-04629-4

[54] May, T.M., Zainudin, Z., Muslim, N., Jamil, N.S., Jan, N.A.M., Ibrahim, N., Sabri, N.A.B. (2024). Intrusion detection system (IDS) classifications using hyperparameter tuning for machine learning and deep learning. In 2024 5th International Conference on Artificial Intelligence and Data Sciences (AiDAS), Bangkok, Thailand, pp. 344-349. https://doi.org/10.1109/AiDAS63860.2024.10730038

[55] Madwanna, Y., Annappa, B., Sneha, H.R. (2023). YARS-IDS: A novel IDS for multi-class classification. In 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), Lonavla, India, pp. 1-6. https://doi.org/10.1109/I2CT57861.2023.10126301

[56] Sinha, S., Degadwala, S. (2023). A comprehensive review on multi-class DDoS attack classification in IoT. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 9(10): 313-318. https://doi.org/10.32628/CSEIT2361053

[57] Khan, A.R., Kashif, M., Jhaveri, R.H., Raut, R., Saba, T., Bahaj, S.A. (2022). Deep learning for intrusion detection and security of Internet of things (IoT): Current analysis, challenges, and possible solutions. Security and Communication Networks, 2022(1): 4016073. https://doi.org/10.1155/2022/4016073

[58] Lumumba, V.W., Kiprotich, D., Lemasulani Mpaine, M., Grace Makena, N., Daniel Kavita, M. (2024). Comparative analysis of cross-validation techniques: LOOCV, K-folds cross-validation, and repeated K-folds cross-validation in machine learning models. American Journal of Theoretical and Applied Statistics, 13(5): 127-137. https://doi.org/10.11648/j.ajtas.20241305.13