# Intrusion Detection in Wireless Sensor Networks: A Lightweight Scheme

Yixian Liu, Feng Ni

*Abstract*— **Wireless Sensor Networks (WSNs) represent a critical technology of the 21st century. However, their inherent characteristics make them vulnerable to various network attacks. Meanwhile, the limited node resources complicate the implementation of effective intrusion detection. In response to these challenges, this study proposes a lightweight intrusion detection scheme. The process begins by applying RobustScaler to reduce the influence of outliers. Then, Incremental Principal Component Analysis (IPCA) extracts 15 principal components from the CICIDS2017 dataset, significantly reducing computational demands. Harris Hawks Optimization (HHO) is employed to further select the optimal subset of components. Finally, a Decision Tree based on information gain detects attacks on the lower-dimensional data representation. Experimental results show that, compared to other classifiers and studies, the proposed method maintains minimal processing overhead on both the laptop and Raspberry Pi 5, while delivering competitive performance metrics. It achieves 99.65% accuracy, 99.64% precision, 99.65% recall, 99.64% F1-score, a false positive rate of 0.25%, and a false negative rate of 0.51%. These outcomes demonstrate the method's effectiveness and suitability for resource-constrained WSNs environment.**

*Index Terms*—**WSNs; intrusion detection; RobustScaler; IPCA; HHO; Decision Tree**

## I. Introduction

THE proliferation and widespread application of the Internet of Things (IoT) have improved the quality of life and transformed various aspects of work. Estimates suggest that by 2025, the number of IoT devices will reach 100 billion [1]. Wireless Sensor Networks (WSNs), a key component of IoT infrastructure, have broad applications in areas such as environmental monitoring, disaster alert systems, agriculture, and smart cities [2]. Typically, WSNs comprise hundreds or thousands of sensor nodes, as illustrated in Fig. 1. Each node collects environmental data — such as air quality, soil moisture, and water quality — and transmits it via wireless channels.

Manuscript received February 23, 2025; revised July 5, 2025.

Yixian Liu is the Director of the Information Security and Information Warfare Experimental Teaching Center, Xi'an University of Posts and Telecommunications, Xi'an, China (e-mail: liu-yi-xian@xupt.edu.cn).

Feng Ni is a Postgraduate Student of the School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an, China (e-mail: nifenggg@gmail.com).

The base station collects data from nodes, processes it, and forwards it to a control center over the internet, which typically includes databases, cloud servers, and terminal devices. In essence, sensor nodes are small-scale computers equipped with embedded operating systems such as TinyOS and Contiki. They are responsible for sensing the environment, gathering data, and transmitting it. Designed for low cost and power efficiency, these devices have limited bandwidth and computational capabilities. Moreover, sensor nodes are often densely deployed, which necessitates a compact and lightweight design. As a result, their limited memory and battery capacity must be efficiently managed to maintain stable operations. These constraints are prevalent among IoT devices, which are inherently constrained in both processing capacity and power supply [3]. Besides, outdoor deployment makes nodes vulnerable to physical damage. Wireless communication also exposes WSNs to threats like eavesdropping, data tampering, and signal interference. Such threats can have severe consequences, especially in military or medical applications. Numerous studies regard traditional public key encryption technology and identity authentication as the first line of defense for WSNs security [4], [5]. However, these methods demand significant computational resources [6], [7], making them impractical for resource-constrained sensor nodes. Recent advances in lightweight encryption have improved their feasibility for low-power devices [8], [9]. It is precisely because of the aforementioned reasons that a series of challenges persist in addressing security issues within WSNs.

To counter various attack behaviors, the security requirements of WSNs generally include data integrity, availability, and confidentiality [10], [11]. Numerous studies have classified malicious attacks against WSNs based on different criteria [12]-[14]. One common approach distinguishes between active and passive attacks. Active attacks involve direct interference with normal communication, such as modifying data or injecting fraudulent packets — examples include man-in-the-middle and energy depletion attacks. In contrast, passive attacks rely on eavesdropping, where adversaries use sniffers to monitor data transmissions between nodes and extract sensitive information. Attacks can also be categorized by their origin: internal attacks, launched by compromised or impersonated nodes, are generally more difficult to detect and prevent than external attacks. Furthermore, attacks tar geting different layers of the OSI model have been systematically classified. Table I shows the classification of common attacks on each

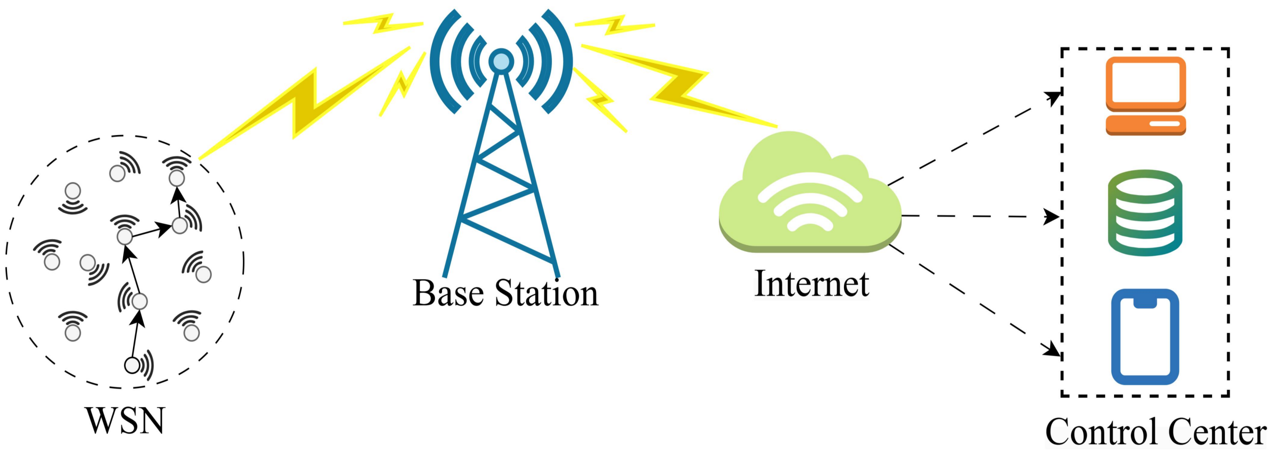

Fig. 1 WSNs Architecture.

layer of WSNs. Among the various types of attacks targeting WSNs, denial-of-service (DoS) attacks are among the most prevalent and can occur at every layer of the network [15]. This is primarily due to two characteristics of WSNs: the limited resources of sensor nodes, which cannot support large-scale data requests, and the distributed architecture of the network, which complicates attacker identification. The primary objective of DoS attacks is to disrupt communication between sensor nodes, preventing the network from functioning properly and eventually causing node failure and energy depletion.

TABLE I
ATTACKS AT EACH LAYER OF WSNS.

| Protocol Layer | Attack Type |
|---|---|
| Physical Layer | Jamming, Dos, Tampering |
| Data Link Layer | Collision, Unfairness, Exhaustion, Dos |
| Network Layer | Dos, Selective Forwarding, Spoofing, Sybil, Hole Attack |
| Transport Layer | Flooding, Dos, Desynchronization |
| Application Layer | Repudiation, Dos |

With the evolution of attack techniques, the first line of defense is no longer adequate to address security challenges in WSNs. Consequently, intrusion detection has been widely recognized as the second line of defense. In recent years, the continuous advancement of hardware technology and the growth of data volume have accelerated the application of machine learning (ML) and deep learning (DL) in WSN intrusion detection [16]. Nevertheless, resource constraints remain a significant challenge for the design and implementation of intrusion detection systems in WSNs. Moreover, ML- and DL-based intrusion detection systems rely on traffic data collected from the network, which often contains redundant features, noise, and high dimensionality. These factors increase storage requirements and computational overhead, potentially impairing model performance. Therefore, developing lightweight intrusion detection methods is essential to reduce computational complexity and storage costs, while maintaining a balance between security and energy consumption [17].

The remaining sections of this paper are as follows. Section II reviews recent developments in lightweight intrusion detection. Section III introduces the proposed framework and describes its implementation process. Section IV analyzes the experimental results and compares the proposed method with conventional ML models and recent studies. Finally, Section V summarizes the paper and outlines directions for future research.

## II. Related Work

The application of ML and DL-based intrusion detection in WSNs has attracted widespread attention and a large amount of research [18]. Compared to traditional methods, they have shown great potential and advantages, opening up new avenues to address security issues in WSNs. Chandre et al. [19] compared the performance of different ML techniques applied to intrusion detection in WSNs. The authors used convolutional neural networks as classifiers in this study and tested on WSN-DS. Based on their findings, DL exhibits better detection results than ML. Chaurasiya et al. [20] proposed a DL-based intrusion detection method for WSNs, namely a dense artificial neural network, which achieved an accuracy of 96.45% on the NSL-KDD dataset, slightly better than the DT and SVM. The drawback is that the authors did not consider the additional impact of computational costs and model complexity. Although in numerous instances, DL can offer enhanced accuracy for the purpose of intrusion detection, they typically necessitate considerable computational power, which can prove challenging in contexts with restricted resources. Wazirali and Ahmad [21] compared the performance of different ML algorithms for detecting DoS attacks in WSNs, they partitioned the WSN-DS into various data subsets and conducted their work on these differently-sized partitions. The results indicate that statistical and logic-based ML models perform best on numerical statistical datasets. Additionally, they noted that using DL algorithms on sensor nodes may be overly demanding, as DL requires extensive training to achieve high accuracy. Similarly, Ahmad et al. [22] conducted a series of comparisons on network intrusion detection systems based on ML and DL, with approximately 80% of the proposed solutions being based on DL. Whereas the implementation of these solutions is highly complex, requiring significant storage and computational resources, and these deficiencies must be further addressed. Otoum et al. [23] introduced a DL-based intrusion detection system called RBC-IDS (Restricted Boltzmann Clustering), this was compared with an adaptive ML-based IDS (ASCH-IDS). The authors found that both systems had similar accuracy and detection rates, however, the detection time for RBC-IDS was nearly twice that of ASCH-IDS, indicating that DL increases the computational burden to some extent.

An increasing number of researchers are integrating feature engineering techniques with ML algorithms. One reason is that the continuous evolution of network attack techniques renders single detection techniques insufficient for effective defense. Another reason is to diminish the complexity of models, thereby reducing system load. Abdulhammed et al.[24] proposed a PCA-based network intrusion detection method, reducing the CICIDS2017 dataset to 10 dimensions. The Random Forest algorithm demonstrated superior classification performance in both binary and multi-class scenarios. Roy et al. [25] introduced a novel Stacking-based lightweight intrusion detection system tailored for IoT environments. This study first identifies multicollinearity among dataset features using the Variance Inflation Factor and groups them accordingly. PCA is then employed for dimensionality reduction, eliminating multicollinearity and reducing the feature space. However, both studies rely on fixed results obtained after dimensionality reduction, which lack flexibility and fail to account for the interactions between principal components and attack variables. Al-Yaseen [26] employed the Firefly Algorithm for feature subset selection and utilized a Support Vector Machine as the base classifier. Experimental results demonstrated that this approach not only reduced feature dimensionality but also enhanced detection accuracy while lowering the false alarm rate. However, the NSL-KDD dataset is outdated and inadequate for representing modern network environments and emerging threats. Elsadig [27] developed a lightweight ML method based on the Decision Tree and Gini feature selection method. This method has been tested on WSN-DS, and compared with other traditional ML classifiers, achieving an accuracy of 99.5% with a processing time of only 0.13 s. The author further emphasizes that DL does not serve as a suitable solution for WSNs. However, this study is restricted to DDoS attacks and cannot be applied to comprehensive intrusion detection. He et al. [28] proposed a lightweight intrusion detection method tailored for IoT, leveraging raw PCAP files to design a rapid protocol parsing and feature grouping mechanism. Experimental results indicate that RF outperforms various other ML algorithms and significantly surpasses CNN in terms of processing time and memory consumption.

## III. Proposed Approach

This section proposes a lightweight intrusion detection scheme based on Incremental Principal Component Analysis (IPCA) and the Harris Hawks Optimization (HHO) algorithm. The IPCA was applied for feature extraction, mapping the data into a lower-dimensional space. Next, considering that IPCA does not maximize the relationship between features and attack variables, each principal component retains part of the original information but does not contribute equally to attack detection in the Decision Tree. The HHO algorithm was introduced to focus on the principal components with the strongest attack discriminatory power. The Decision Tree using information gain as the splitting criterion was employed to classify network traffic on the new data representation. Finally, common metrics, including accuracy, precision, recall, F1 score, false alarm rate, and false negative rate were used for validation. The detailed framework of the model is shown in Fig. 2.

### *A. Dataset*

The CICIDS2017 dataset, which has now been widely employed to assess the effectiveness of intrusion detection algorithms in IoT environments [29]. It was developed by the Canadian Cybersecurity Research Institute and the University of New Brunswick. It contains 78 feature columns and 1 label column. The specific descriptions of each feature and category in the dataset can be found in [30].

### *B. Data Cleaning*

Data cleaning is an integral component of data preprocessing. Typically, raw datasets contain various issues such as missing values and duplicate records, which can affect the reliability of the data. Data analysis revealed that the CICIDS2017 dataset contains duplicate samples. These duplicate samples can diminish the model's generalization capability and necessitate their removal. Additionally, the "Flow Bytes/s" column in the dataset contains missing and infinite values, and the "Flow Packets/s" column also contains infinite values. Consistent with the duplicate removal approach, data in these columns was removed to ensure integrity. Table II presents the CICIDS2017 distribution after cleaning.

TABLE II
Distribution of CICIDS2017 After Preprocessing.

| Classes | Number of samples |
|---|---|
| Benign | 795104 |
| Dos | 193745 |
| DDos | 128014 |
| FTP-Patator | 5931 |
| SSH-Patator | 3219 |
| PortScan | 90694 |
| Bot | 1948 |
| Infiltration | 36 |
| Heartbleed | 11 |
| Web Attack | 2143 |
| Total | 1220845 |

### *C. Label Encoding*

Most ML algorithms can only process numerical data, making feature encoding an essential step in ML. Label encoding maps each category to an integer, starting from 0 and incrementing by 1. This process converts non-numerical features in the raw data into numerical features.

### *D. Standardization*

Proper standardization of data before performing Incremental Principal Component Analysis (IPCA) ensures that each feature contributes more evenly to IPCA. This study used the RobustScaler for data standardization. One of the advantages of the RobustScaler is its ability to effectively $X_{\text{batch, centered}}$ is the centered data matrix of the current input

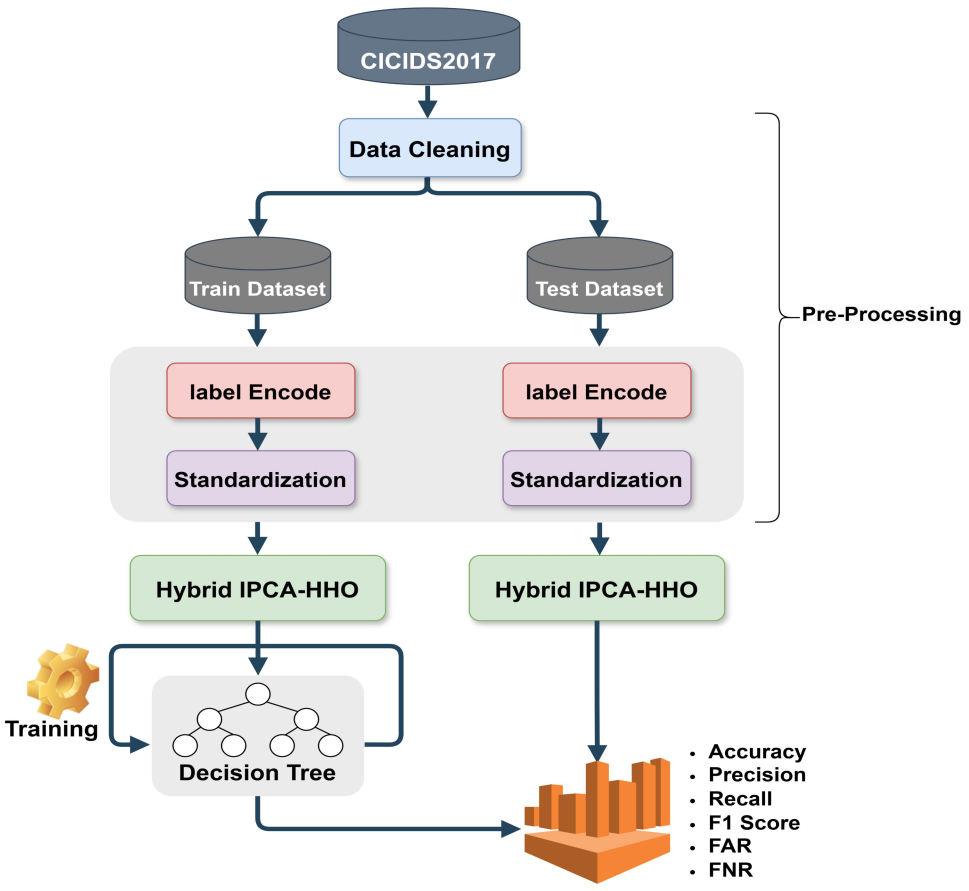

Fig. 2 Core structure of the proposed approach.

reduce the impact of outliers on the scaling results, making the scaled data more robust [31]. RobustScaler employs the median and the interquartile range (IQR) for feature scaling, where IQR denotes the difference between the first quartile (Q1) and the third quartile (Q3). Specifically, for each feature, the median and IQR are calculated first. The data is then scaled using IQR, thus compressing it into a smaller range. The following formula is commonly used for standardization :

$$x_{scaled} = \frac{x_i - \text{median}(x)}{\text{IQR}} \tag{1}$$

here, $x_{scaled}$ is the standardization value of feature $x$, $x_i$ is the original feature value, median($x$) is the median of feature $x$.

## E. Incremental Principal Component Analysis

As a widely utilized linear dimensionality reduction technique, Principal Component Analysis (PCA) can significantly reduce the dimensionality of the data while preserving crucial information, achieving this by distilling the main elements of data known as principal components by identifying the directions of greatest variance [32]. By retaining the most representative principal components, transforming the original high-dimensional data into a set of new independent variables. Incremental Principal Component Analysis (IPCA), a variant of PCA, extracts principal components based on singular value decomposition (SVD) [33], [34], expressed as:

$$X = U\Sigma V^T \tag{2}$$

here, $U$ denotes the left singular matrix, with each column constituting the left singular vectors of $X$, $\Sigma$ represents a diagonal matrix whose diagonal elements are the singular values of $X$, and $V^T$ is the transpose of the matrix composed of the right singular vectors of $X$. Unlike conventional PCA, which requires loading the entire dataset into memory for processing, IPCA operates by partitioning the original dataset into multiple fixed-size batches:

$$X_{\text{batch}} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{md} \end{bmatrix} \tag{3}$$

where $X_{\text{batch}}$ is the batch data matrix, the variable $d$ denotes the feature count in the dataset, while $m$ refers to the sample size in the current batch. It iteratively updates the global mean by incorporating data from the current batch and samples from previous batches:

$$\mu_{\text{new}} = \frac{s\mu_{\text{old}} + m\mu_{\text{batch}}}{s+m} \tag{4}$$

$s$ represents the amount of data processed, $\mu_{old}$ is the previous global mean, and $\mu_{\text{batch}}$ is the mean of the current batch. Subsequently, a new matrix is formulated:

$$X_{\text{new}} = \begin{bmatrix} \Sigma_{\text{old}} \cdot V_{\text{old}}^T \\ X_{\text{batch, centered}} \\ \sqrt{\frac{s}{s+m}} \cdot m(\mu_{\text{old}} - \mu_{\text{batch}}) \end{bmatrix} \tag{5}$$

batch size. A new SVD is then performed to update the singular values and singular vectors. By incrementally computing each batch and continuously updating the principal components until the entire dataset is processed, principal components were extracted from the final right singular matrix $V$ . Subsequently, the original dataset was projected onto these principal components:

$$X_{\text{reduced}} = X_{\text{centered}} V_k \tag{6}$$

Where $X_{\text{reduced}}$is the new low-dimensional data representation, $X_{\text{centered}}$ is the centered data matrix of the original data and the $V_k$ represents the first $k$ vectors of the matrix $V$ where $k$ ($k \leq d$) is the number of principal components selected. IPCA reduces memory consumption, offering greater flexibility and efficiency, particularly when handling large-scale datasets.

### *F. Harris Hawks Optimization*

Harris Hawks Optimization (HHO), conceptualized by Heidari et al. in 2019 [35], represents a meta-heuristic algorithm derived from the predatory strategies exhibited by Harris Hawks. HHO is renowned for its simplicity and its robust capabilities in both global and local search, making it highly versatile for solving optimization problems across various domains. In feature selection tasks, the initial population of solutions is typically represented as binary vectors, where the length of each vector corresponds to the number of features in the dataset. A value of 1 in the vector indicates the selection of a feature, while 0 denotes its exclusion. HHO initializes the population by randomly generating multiple binary candidate solution vectors. The update of candidate positions within the population—whether through global exploration or local exploitation—is determined by the prey's escape energy and specific random conditions, allowing for dynamic adjustments in the positions and directions of solutions. Global exploration incorporates strategies such as random tall trees and family member strategies, aimed at expanding the search space. In contrast, local exploitation focuses on fine-tuning solution positions, utilizing strategies like soft besiege, hard besiege, soft besiege with rapid dives, and hard besiege with rapid dives.

This study sought to enhance classification performance within the principal component space by leveraging a fitness function. The fitness function was defined as:

$$Fitness(solution) = 1 - Accuracy \tag{7}$$

which represents the complement of the attack detection accuracy of the Decision Tree. To prevent the selection of an empty subset of principal components, the fitness function returns a value of 1 as a penalty. The corresponding pseudocode is presented in Algorithm 1.

### *G. Decision Tree*

A tree-structured ML algorithm under supervised learning, the Decision Tree (DT), is straightforward to implement and exhibits low model complexity, which has been proven to result in high cost-efficiency for applications in the IoT domain, while meeting real-time response requirements [36]-[38]. DT consists of multiple nodes and directed edges. Nodes are classified as internal or leaf nodes. Internal nodes serve as decision points, where the dataset is partitioned according to specific features, acting as criteria for branching. Leaf nodes, on the other hand, represent the final outcomes of the decision process [39]. In this study, the information gain method was selected as the feature selection strategy for the experiment. Information gain measures the reduction in uncertainty of the original dataset due to a particular feature :

$$Gain(D, F) = H(D) - H(D|F) \tag{8}$$

$Gain(D, F)$ represents the information gain, where $D$ is the training dataset and $F$ is the feature. $H(D)$ is the empirical entropy of the dataset, $H(D|F)$ is the conditional entropy of $D$ given feature $F$. Node splitting typically aims to select the feature with the highest information gain, signifying a greater impact on reducing entropy. Entropy quantifies the level of uncertainty and is calculated as:

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i \tag{9}$$

$X$ is a discrete random variable that can take on a limited set o f values, with its probability distribution expressed as $P(X = x_i) = p_i, i = 1,2, \ldots, n$. Conditional entropy represents the unc ertainty of a random variable given known conditions:

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i) \tag{10}$$

let Y be a random variable, and let X be the given condition with $P(X = x_i) = p_i, i = 1,2, \ldots, n$.

**Algorithm 1** Harris Hawks Optimization(HHO)

**Input**: Principal component space $K$, population size $N$, maximum number of iterations $M$
**Output**: Optimal principal component subset $X_{best}$, classification accuracy

```
1: Initialize the population: randomly generate $N$ binary vectors $X_i$
2: Compute the fitness value for each $X_i$, record the best solution $X_{best}$
3: for iterations = 1 to $M$ do
4:     Calculate escape energy $E$:
           $E = 2 \times (1 - \text{iterations}/M)$
5:     for each individual $X_i$ in the population do
6:         if $|E| \geq 1$ and random $> 0.5$ then
7:             # Exploration: Random tall tree
8:         else if $|E| \geq 1$ and random $\leq 0.5$ then
9:             # Exploration: Family members mean
10:        else if $0.5 \leq |E| < 1$ and random $> 0.5$ then
11:            # Exploitation: Soft besiege
12:        else if $|E| < 0.5$ and random $> 0.5$ then
13:            # Exploitation: Hard besiege
14:        else if $0.5 \leq |E| < 1$ and random $\leq 0.5$ then
15:            # Exploitation: Soft besiege with progressive dives
16:        else
17:            # Exploitation: Hard besiege with progressive dives
18:        end if
19:        Repair $X_i$: Ensure binary encoding
20:        Evaluate new fitness for $X_i$ and update $X_{best}$ if better
21:    end for
22: end for
23: return Final optimal subset $X_{best}$ and accuracy
```

## IV. Experiments and Evaluation

### A. Experimental Environment

The experimental platform was based on the Windows 11 operating system. PyCharm served as the development tool, and model construction was performed using the Scikit-learn library in Python version 3.8. Moreover, due to its low power consumption, portability, and affordability, the Raspberry Pi has become a popular choice for IoT devices [40]. Consequently, this study employed Raspberry Pi 5 to further validate the scalability of the proposed method and to assess its effectiveness in resource-constrained environments. This is critical for the practical deployment of WSNs applications. Table III outlines the specific devices and their corresponding hardware specifications.

TABLE III
DEVICES IN THE STUDY.

| Devices | Components |
|---|---|
| HP Laptop | Intel Core i5-13500H CPU; 16 GB RAM |
| Raspberry Pi 5 | Broadcom BCM2712 SoC (Quad-core ARM Cortex-A76); 8 GB RAM |

.

### B. Performance Metrics

In the domain of ML, evaluation of classification models frequently involves metrics such as accuracy, precision, recall, and the F1 score. Accuracy reflects the overall effectiveness of an intrusion detection system by quantifying the proportion of correctly classified instances relative to the total number of instances. Precision measures the proportion of correctly identified instances of a particular attack category among all instances predicted to belong to that category. Recall, also referred to as the detection rate, evaluates the model's capability to accurately identify attack instances by calculating the proportion of actual attacks that are correctly detected. The F1 score integrates both precision and recall into a single measure, typically expressed as their harmonic mean. The formal definitions of these four widely used performance metrics are provided below:

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{11}$$

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$F1\ Score = 2 * \frac{precision * recall}{precision + recall} \tag{14}$$

What's more, within the realm of intrusion detection, the False Alarm Rate (FAR) and the False Negative Rate (FNR) serve as critical metrics for evaluating the effectiveness of detection models. FAR typically refers to the proportion of normal traffic that a detection system incorrectly classifies as attacks. An excessively high FAR can not only disrupt legitimate activities within the network but also impair the efficiency of actual attack detection. Conversely, the FNR quantifies the proportion of actual attacks that the model fails to detect, representing the fraction of malicious traffic erroneously classified as benign. A high FNR indicates that a substantial number of attacks go undetected, thereby compromising the effectiveness of the detection system. The FAR and FNR are defined as follows:

$$FAR = \frac{FP}{FP + TN} \tag{15}$$

$$FNR = \frac{FN}{FN + TP} \tag{16}$$

Notably, under resource-constrained conditions, execution time is a crucial metric for the comprehensive evaluation of algorithmic performance. In this study, the processing time is the average result of ten executions under the same conditions.

### C. Analysis and Discussion

This experiment initially aimed to investigate the impact of different standardization methods on feature extraction. The updated dataset spanned a 70-dimensional feature space. As shown in Fig. 3, the cumulative explained variance ratio for CICIDS2017 revealed that reducing the number of principal components to fewer than 10 retained less than 70% of the original variance. This resulted in a significant loss of valuable information, constraining the learning capability of intrusion detection models.

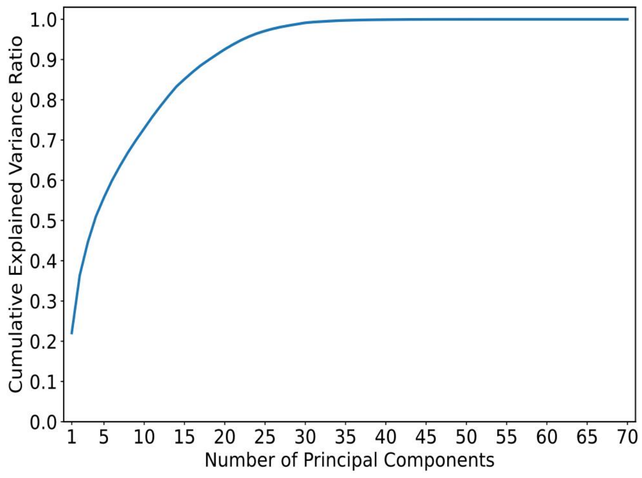

Fig. 3 Cumulative Explained Variance Ratio on CICIDS2017.

Consequently, the feature dimensionality was progressively reduced from 70 to 10 in steps of 5. Fig. 4 presents the comparison of detection performance after applying IPCA based on different standardization methods. Among them, RobustScaler consistently outperformed the others. When the CICIDS2017 dataset was standardized using RobustScaler, the DT maintained high detection accuracy across different numbers of principal components. In contrast, principal components from conventionally scaled data restricted DT capability and exhibited instability in identical dimensional spaces. These discrepancies were primarily ascribed to the intrinsic characteristics of the original data distribution. Potential outliers and noise in the raw data can degrade the effectiveness of traditional scaling methods, whereas RobustScaler is less sensitive to such issues, thereby reducing their impact and allowing the DT to detect a variety of attacks in low-dimensional spaces. Given the high dimensionality of the original features, selecting an appropriate number of principal components was essential. As shown in Figs. 3 and 4, the first 30 principal components captured nearly all the useful information, indicating significant redundancy. When the number of components reached 15, the accuracy plateaued, and additional components yielded marginal gains. Therefore, retaining 15 principal components allowed the DT to maintain high classification performance while avoiding unnecessary computational complexity. Meanwhile, the effect of varying IPCA batch sizes was examined, as depicted in Fig. 5. Very small batch sizes led to frequent updates of the principal components, hindering convergence. Conversely, excessively large batches offered negligible improvements in detection capability but increased memory usage. Therefore, a batch size of 10,000 samples was adopted as a balanced configuration.

Table IV compares the performance of DT trained with Information Gain (IG-DT) and Gini Index (Gini-DT) after dimensionality reduction via PCA and IPCA. Gini-DT struggled with this task, requiring more computational time to achieve accuracy comparable to IG-DT, while also exhibiting higher FAR and FNR. In contrast, IG-DT more effectively captured the correlation between principal components and different attack types. On the other hand, PCA and IPCA exerted a similar impact on the final classification performance of DTs. However, the batch processing mechanism demonstrated significantly higher computational efficiency. As Fig. 6 shows, IPCA reduced execution time and peak memory consumption across all dimensional settings.

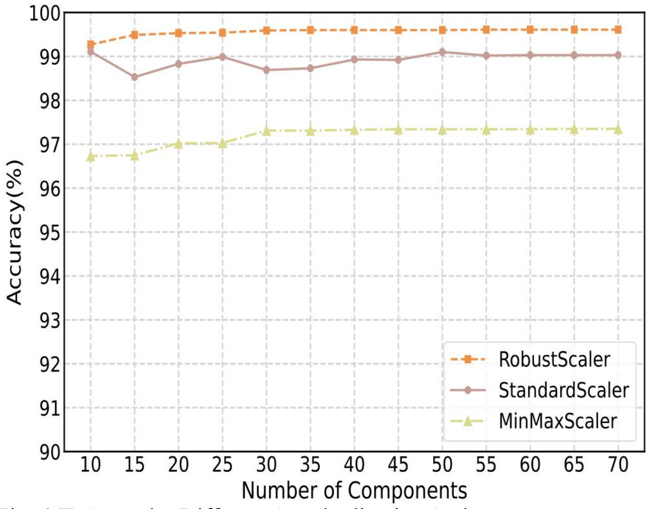

Fig. 4 IPCA under Different Standardization Styles.

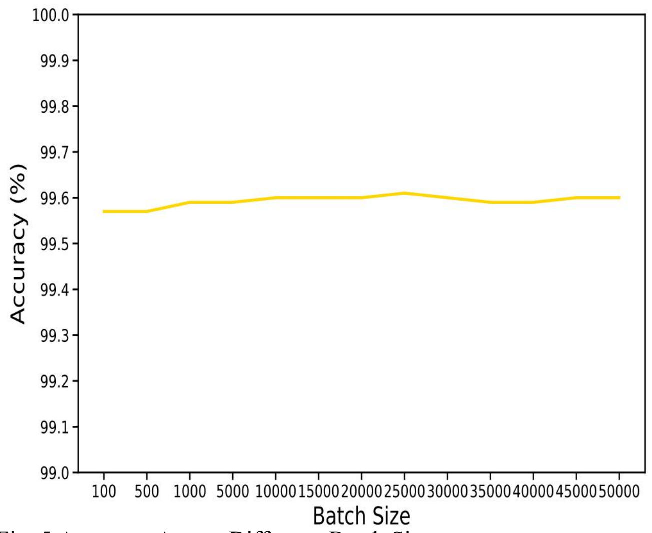

Fig. 5 Accuracy Across Different Batch Size.

TABLE IV
COMPARISON OF MODEL PERFORMANCE.

| Classifiers | Accuracy (%) | FAR (%) | FNR (%) | Processing Time(s) |
|---|---|---|---|---|
| PCA + Gini- DT | 99.57 | 0.29 | 0.70 | 6.65 |
| PCA + IG-DT | 99.60 | 0.28 | 0.57 | 4.52 |
| IPCA+ Gini-DT | 99.58 | 0.28 | 0.69 | 6.63 |
| IPCA+ IG-DT | 99.60 | 0.28 | 0.56 | 4.44 |

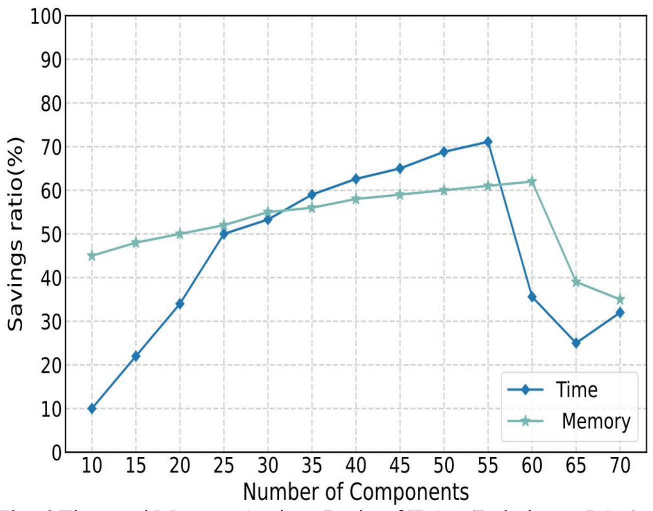

Fig. 6 Time and Memory Savings Ratio of IPCA (Relative to PCA).

After transforming the dataset using IPCA, the resulting 15 principal components were used as input to the HHO algorithm to eliminate redundant components and enhance detection performance. This dimensionality reduction significantly narrowed the search space of the HHO compared to the original dataset. Specific parameters are detailed in Table V. As the hawk swarm size increased, both the selected subset and the number of chosen principal components varied, as shown in Table VI.

TABLE V
PARAMETER OF HHO.

| Parameters | Values |
|---|---|
| Population Size | 10, 20, 30, 40, 50 |
| Maximum Iteration | 100 |
| Levy | 0.01 |

TABLE VI
SELECTED PRINCIPAL COMPONENTS SUBSETS.

| Hawks Size | Selected Principal Components | Accuracy (%) |
|---|---|---|
| 10 | 1,4,5,6,8,9,10,11,12,13,14,15 | 99.65 |
| 20 | 1,3,4,5,8,9,10,11,12,13,14,15 | 99.64 |
| **30** | **5,6,9,10,11,12,13,14,15** | **99.65** |
| 40 | 5,6,9,11,12,13,14,15 | 99.63 |
| 50 | 6,7,9,10,11,12,13,14,15 | 99.64 |

Components 9, 11–15 were consistently chosen across all subsets, suggesting their higher relevance to attack detection. In experiments with a hawk swarm size of 20 or fewer, the HHO tended to select a larger number of features. However, with a hawk swarm size of 30, the DT achieved optimal accuracy using fewer principal components. Based on this result, the subset of principal components selected under a hawk swarm size of 30 was retained as the final set of principal components. Figs. 7 and 8 show 2D visualizations of the original feature space versus the reduced space with nine selected components. In the original dataset, significant overlap was observed between different classes, and some benign traffic samples appeared as extreme outliers. In contrast, the IPCA-HHO-transformed data showed a more compact distribution, with certain classes forming distinct clusters. However, some degree of overlap remained, suggesting that complete separation of various attacks and normal traffic might have required representation in a higher-dimensional feature space or could have been attributed to the class imbalance in the dataset. Table VII presents a performance comparison of IG-DT in principal component spaces selected by different supervised feature selection methods, including Fisher Score, Analysis of Variance (ANOVA), Mutual Information (MI), and Lasso regression. The results indicated that the proposed method outperformed all other approaches across all evaluation metrics. HHO mitigated the information loss caused by IPCA by prioritizing principal components that contributed significantly to attack classification. This synergy enhanced the predictive reliability of DT and effectively reduced both the FAR and FNR.

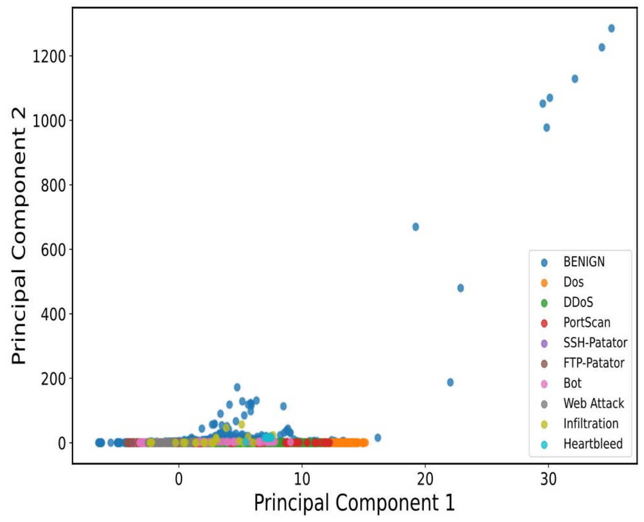

Fig. 7 2D Projection on CICIDS2017.

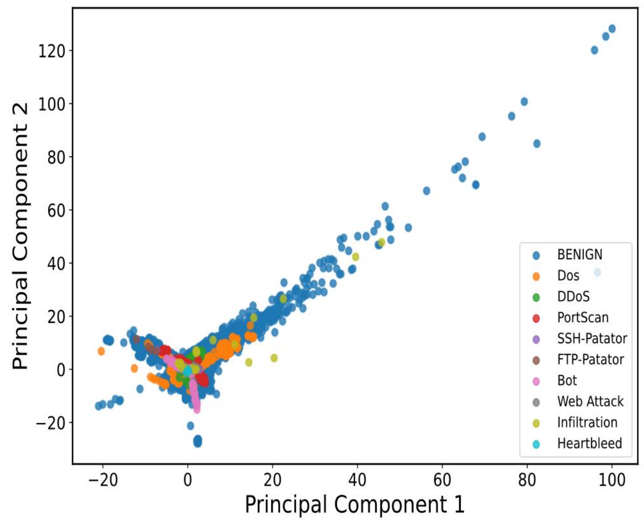

Fig. 8 2D Projection After IPCA-HHO on CICIDS2017.

TABLE VII
EVALUATION OF HYBRID DIMENSIONALITY REDUCTION APPROACHES.

| Hybrid Methods | Accuracy (%) | FAR(%) | FNR(%) |
|---|---|---|---|
| IPCA + Fisher Score | 99.57 | 0.27 | 0.60 |
| IPCA +ANOVA | 99.57 | 0.27 | 0.60 |
| IPCA + MI | 99.58 | 0.30 | 0.57 |
| IPCA+Lasso | 99.59 | 0.29 | 0.58 |
| **Proposed** | **99.65** | **0.25** | **0.51** |

TABLE VIII
PARAMETERS OF EACH CLASSIFIER.

| Classifiers | Parameters |
|---|---|
| DT | Max Depth=30, Classification criterion= Information Gain |
| RF | Number of trees =74, Classification criterion= Information Gain |
| KNN | Neighbors=3 |
| LR | Regularization parameter =0.7, Max Inter=600 |
| GBDT | Number of trees =68, Learning Rate=0.1 |

To further evaluate the effectiveness and rationality of the proposed method, it was compared against five benchmark ML classifiers—DT, Random Forest (RF), K-Nearest Neighbors (KNN), Logistic Regression (LR), and Gradient Boosting Decision Tree (GBDT)—which were trained and tested on the original dataset. The key parameters of each model are summarized in Table VIII. Performance metrics and average processing times on a laptop are reflected in Figs. 9 and 10, respectively, while Fig. 11 shows the elapsed times when deployed on Raspberry Pi 5. Training and inference times on both platforms are reported in Tables IX and X.

As shown in Fig. 9, RF performed the best across all evaluation metrics, achieving 99.80% in each. It also boasted the lowest FAR and FNR, as shown in Table XI. However, RF incurred high computational costs, with average processing times of 125.39 s on the laptop and 315.31 s on Raspberry Pi 5. This overhead stems primarily from the ensemble nature of RF, which involves training and aggregating multiple decision trees. The complexity is mainly driven by the number and depth of trees, and the training process requires substantial parallel computation. On resource-constrained devices, such complexity may limit its practical applicability.

Meanwhile, KNN and GBDT also performed well, each exceeding 99% in all metrics and maintaining FAR and FNR below 1%. However, they demonstrated longer classification times than RF, particularly on Raspberry Pi 5. The proposed method, by comparison, exhibited negligible processing time. Given that KNN is an instance-based learning method, it identifies the *n* nearest neighbors for each instance by computing the distances and making decisions based on their labels. This means that for each sample, KNN must traverse the entire dataset, significantly increasing processing time, especially with large-scale datasets. The construction of each tree in GBDT depends on the gradients of the preceding tree, resulting in a sequential update and optimization process. Unlike RF's parallelism, this approach is significantly more computationally intensive. Although LR executed faster than KNN and GBDT (137.71 s on the laptop and 540.9 s on Raspberry Pi 5), its detection performance was inferior, with FAR and FNR of 5.15% and 1.98%, respectively, and other metrics below 97%. While tuning hyperparameters such as the number of iterations may enhance its performance, it would also increase computational complexity.

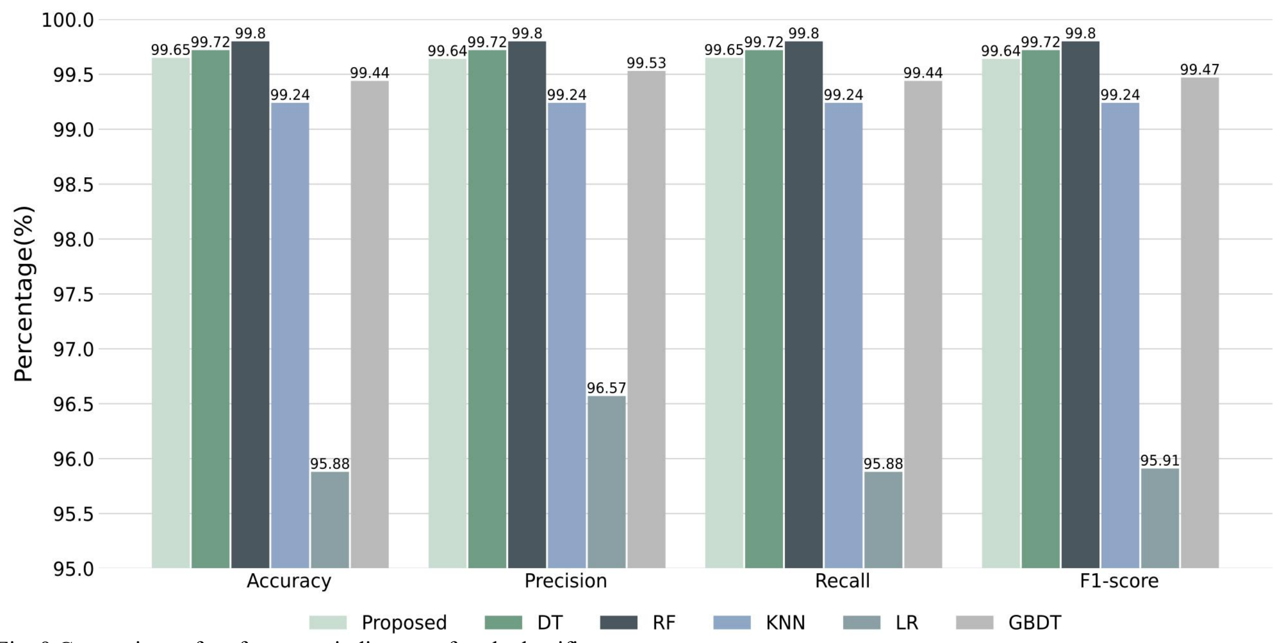

Fig. 9 Comparison of performance indicators of each classifier.

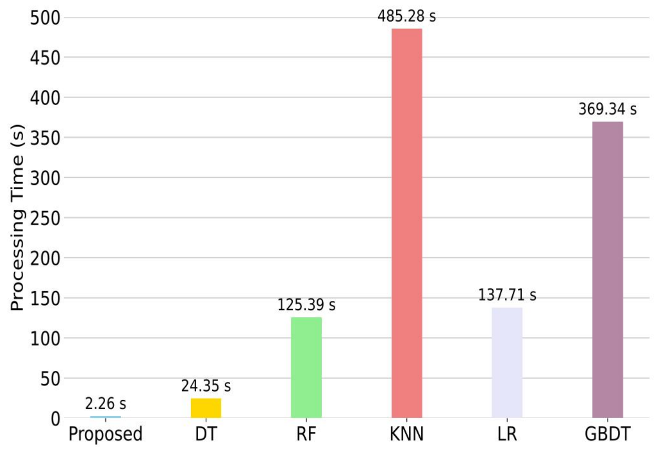

Fig. 10 Processing duration on laptop.

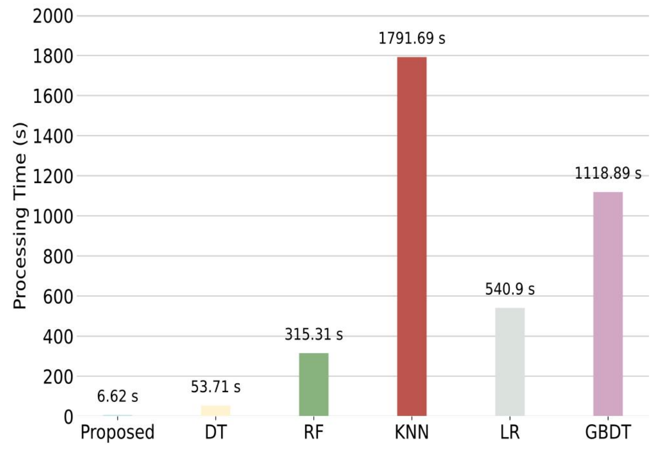

Fig. 11 Processing duration on Raspberry Pi 5.

Among the classifiers trained on the original dataset, DT demonstrated the highest computational efficiency, with 99.72% in accuracy, precision, recall, and F1 score, with FAR of 0.20% and FNR of 0.45%. In comparison, the proposed method achieved slightly lower performance across all six metrics than the conventional DT, with a marginal decline of less than 0.1%. Specifically, it attained 99.65% accuracy and recall, 99.64% precision and F1 score, an FAR of 0.25% and an FNR of 0.51%. It is noteworthy that the proposed method demonstrated a time cost of 2.26 s on a laptop, and an average elapsed time of only 6.62 s on Raspberry Pi 5, which faces performance bottlenecks. This significantly reduced computational overhead while ensuring a rapid response to network attacks. This slight reduction is primarily attributed to the inevitable loss of certain original information during the IPCA process.

TABLE IX
TRAINING AND INFERENCE TIME FOR DIFFERENT CLASSIFIERS ON LAPTOP.

| Classifiers | Training Time(s) | Inference Time(s) |
|---|---|---|
| DT | 24.23 | 0.12 |
| RF | 123.49 | 1.90 |
| KNN | 0.21 | 485.07 |
| LR | 111.31 | 0.09 |
| GBDT | 363.19 | 5.15 |
| **Proposed** | **2.23** | **0.03** |

TABLE X
TRAINING AND INFERENCE TIME FOR DIFFERENT CLASSIFIERS ON RASPBERRY PI 5.

| Classifiers | Training Time(s) | Inference Time(s) |
|---|---|---|
| DT | 53.40 | 0.31 |
| RF | 306.71 | 8.60 |
| KNN | 0.60 | 1791.09 |
| LR | 539.9 | 0.31 |
| GBDT | 1100.34 | 18.55 |
| **Proposed** | **6.56** | **0.06** |

TABLE XI
FAR AND FNR FOR EACH CLASSIFIER.

| Classifiers | FAR (%) | FNR (%) |
|---|---|---|
| DT | 0.20 | 0.45 |
| RF | 0.17 | 0.30 |
| KNN | 0.70 | 0.90 |
| LR | 5.15 | 1.98 |
| GBDT | 0.28 | 0.97 |
| **Proposed** | **0.25** | **0.51** |

Furthermore, this study measured the energy consumption and average processor power (using Intel Power Gadget) during the operation of various classifiers on a laptop, as well as the peak memory usage percentage of their respective processes. These results are presented in Table XII and Fig. 12. The findings further validate the effectiveness of the proposed approach in meeting low-power and low-memory requirements while maintaining high detection performance. This lightweight design is particularly suitable for WSN devices. Although a slight reduction in detection capability was observed, the loss remained within acceptable limits and was offset by a substantial reduction in computational and energy demands, thereby enhancing deployment cost-efficiency. In the context of WSNs, achieving a balance between computational efficiency and security is essential [41]. Moreover, the proposed method exhibits scalability when applied to larger datasets. Dynamically adjusting the IPCA batch size effectively reduces memory consumption. As the number of features increases, HHO algorithm requires more iterations to converge, increasing computational complexity. However, IPCA-based dimensionality reduction significantly reduces the HHO search space, partially offsetting the additional overhead. The selection of principal component subsets can also be optimized by modifying the fitness function to meet varying security metric requirements. Nevertheless, due to differences in data distributions across datasets, fine-tuning—such as adjusting standardization techniques, the number of principal components, and the population size—is necessary to balance detection performance and computational cost.

TABLE XII
RESOURCE CONSUMPTION.

| Classifiers | Energy Consumption (J) | Processor Power (W) |
|---|---|---|
| DT | 544.08 | 23 |
| RF | 3085.88 | 23 |
| KNN | 22757.22 | 433 |
| LR | 7627.60 | 48 |
| GBDT | 8052.43 | 22 |
| **Proposed** | **45.89** | **19** |

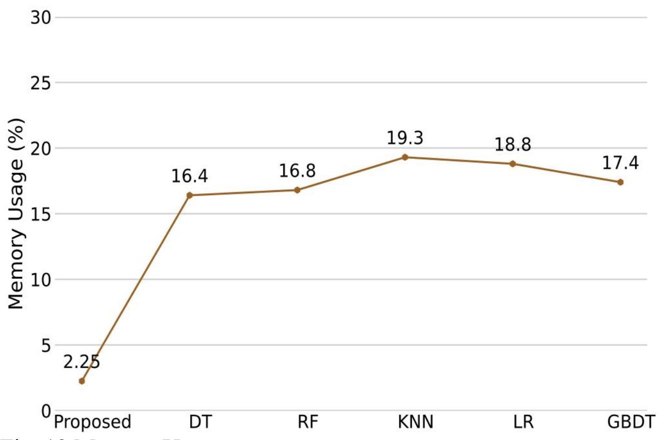

Fig. 12 Memory Usage.

Table XIII illustrates the recognition capabilities of the proposed method across various attacks. With only nine principal components, the DT maintained strong predictive capabilities for most attacks. However, its performance was relatively weaker for bot and infiltration attacks, likely due to both the limited number of training instances and overlapping feature boundaries. As illustrated by the confusion matrix in Fig. 13, considerable overlap was observed between the feature distributions of bot and benign traffic, as well as infiltration and benign traffic, leading to frequent misclassifications of these attacks as normal behavior. Additionally, Fig. 14 shows the Receiver Operating Characteristic (ROC) curve, while Fig. 15 employs SHAP values [42] to quantify the contribution of each principal component to prediction outcomes for different attack types, further enhancing the interpretability of the DT in detecting malicious activity.

TABLE XIII
DETECTION METRICS OF THE PROPOSED METHOD FOR DIFFERENT ATTACKS.

| Attacks | Precision(%) | Recall (%) | F1 Score (%) |
|---|---|---|---|
| Benign | 99.73 | 99.75 | 99.74 |
| Dos | 99.64 | 99.45 | 99.54 |
| DDos | 99.93 | 99.91 | 99.92 |
| FTP-Patator | 99.60 | 99.21 | 99.40 |
| SSH-Patator | 98.69 | 99.29 | 98.99 |
| PortScan | 98.96 | 99.93 | 99.44 |
| Bot | 80.10 | 53.49 | 64.15 |
| Infiltration | 100.00 | 61.54 | 76.19 |
| Heartbleed | 100.00 | 100.00 | 100.00 |
| Web Attack | 97.48 | 97.79 | 97.64 |

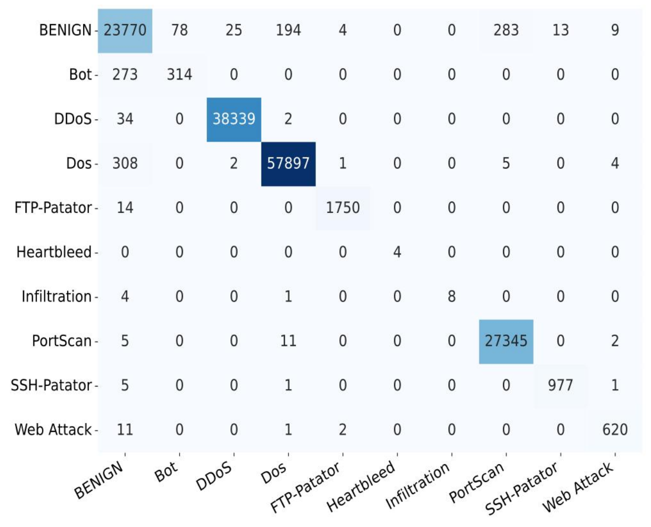

Fig. 13 Confusion Matrix.

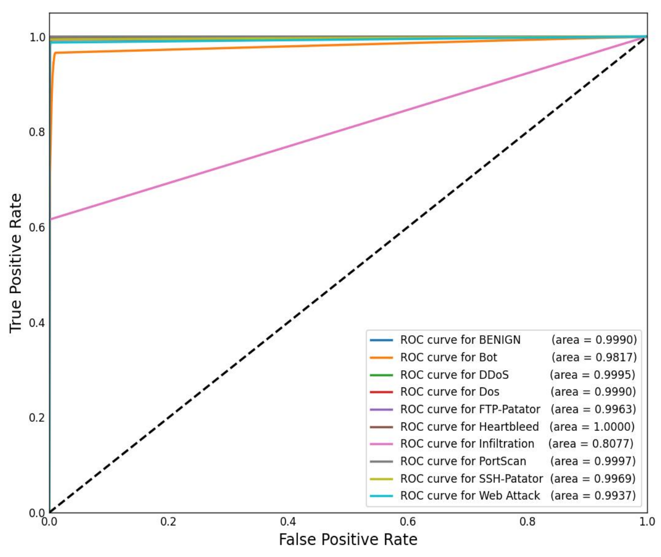

Fig. 14 Roc Curve.

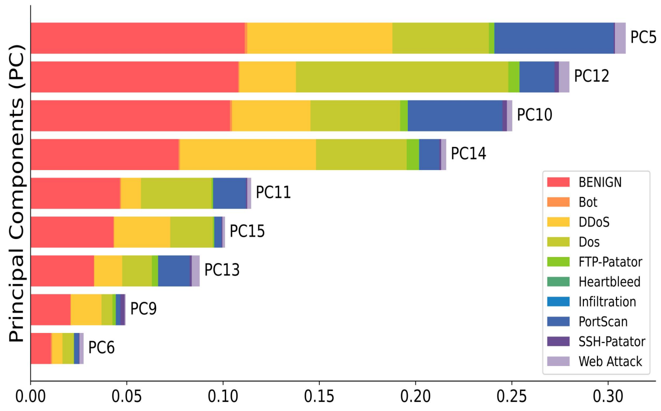

Fig. 15 Average Impact on Model Output Magnitude.

To ensure greater fairness and representativeness of the results, this study conducted an additional 10 rounds of hold-out validation. Figs. 16, 17, 18, and 19 illustrate the variations in accuracy, precision, recall, and F1 score for the 6 classifiers across different runs. As shown in the figures, each model exhibited only minor fluctuations within a narrow range across all experiments, indicating overall stability. The average values of these metrics for each classifier are summarized in Table XIV. Notably, the metrics of LR and GBDT showed greater variability, likely due to the randomness introduced by a single data split. However, the repeated hold-out method helped mitigate such inconsistencies. In contrast, other classifiers, including the proposed method, produced average results that closely aligned with those obtained from a single split. This consistency suggests that the proposed approach demonstrates robust performance across varying data distributions.

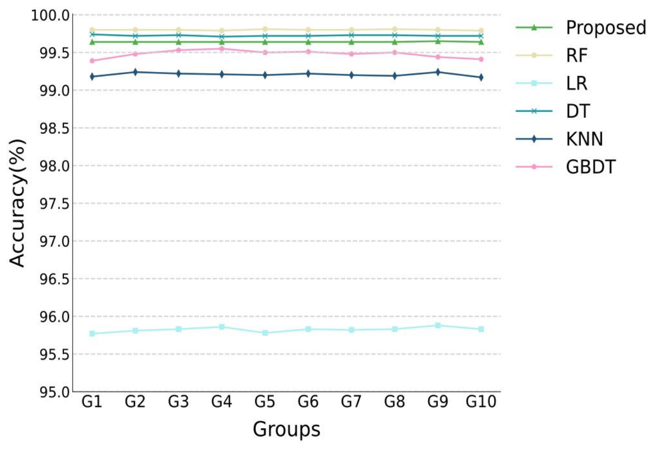

Fig. 16 Accuracy Across Hold-Out Methods.

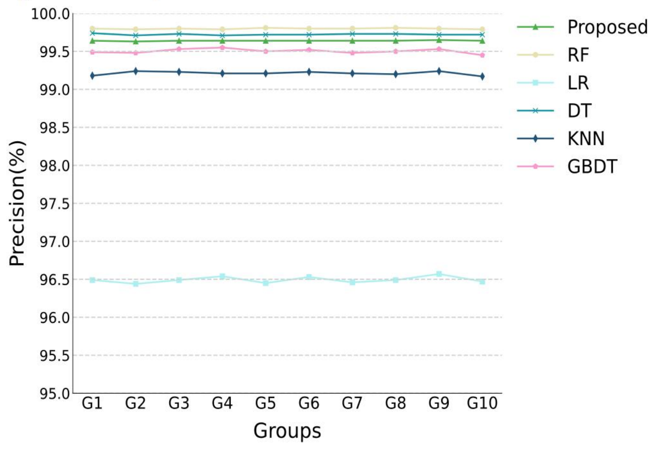

Fig. 17 Precision Across Hold-Out Methods.

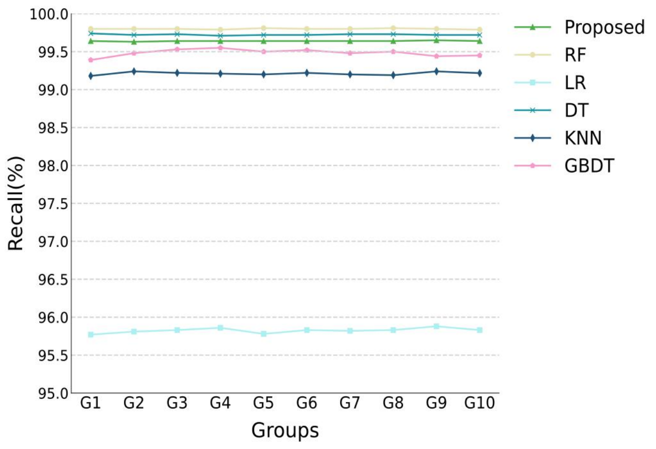

Fig. 18 Recall Across Hold-Out Methods.

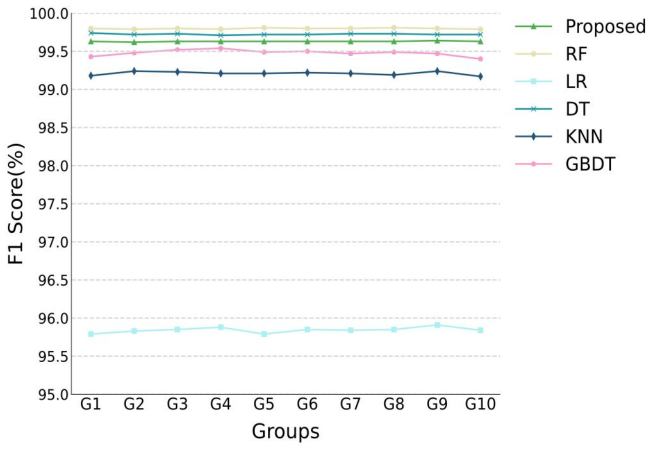

Fig. 19 F1 Score Across Hold-Out Methods.

TABLE XIV
AVERAGE METRICS OF DIFFERENT CLASSIFIERS UNDER THE HOLDOUT METHOD.

| Classifiers | Accuracy (%) | Precision (%) | Recall (%) | F1 Score(%) |
|---|---|---|---|---|
| DT | 99.71 | 99.71 | 99.71 | 99.71 |
| RF | 99.80 | 99.80 | 99.80 | 99.80 |
| KNN | 99.21 | 99.21 | 99.21 | 99.21 |
| LR | 95.82 | 96.49 | 95.82 | 95.84 |
| GBDT | 99.48 | 99.50 | 99.48 | 99.48 |
| **Proposed** | **99.65** | **99.64** | **99.65** | **99.64** |

### *D. Computational complexity*

The computational complexity of IPCA is predominantly governed by SVD operations executed across distinct data batches, formally expressed as $O(\mathrm{md}^2)$. Here, *m* corresponds to the sample count within each batch, while *d* signifies the feature dimensionality. Regarding the HHO algorithm, its computational overhead is principally dictated by three factors: the hawk population size ($N$), the dimensionality of the principal component subspace ($K$), and the maximum iteration count ($M$), yielding a composite complexity of $O(N{\cdot}K{\cdot}M)$. For the DT, complexity analysis segregates into two primary stages: training and inference. The training phase exhibits $O(n{\cdot}p{\cdot}\log(p))$, complexity, with *n* representing training instances and *p* the optimized principal components. During inference, operational complexity is determined by tree depth, conventionally characterized as $O(\log(n))$.

### *E. Comparison with Current Researches*

This subsection presents a comparative analysis of the proposed method with existing research on network intrusion detection algorithms based on the CICIDS2017 dataset. These studies have employed various techniques to simplify the dataset and reduce computational load. Table XV provides a detailed comparison of the data. [43] proposed a binary manta ray foraging (MRF) optimization algorithm to select the optimal feature subset. Although the accuracy of this method is comparable to that of this study, precision, recall, and F1 score show varying degrees of decline. [44] employed a deep neural network (DNN) and introduced a feature selection scheme that integrates standard deviation with the difference between the mean and the median. Although the approach exhibits near-flawless results across multiple metrics, it comes at the cost of high computational complexity. The hybrid CNN and LSTM models proposed by [45] achieved an accuracy of 98.99% in the context of intrusion detection in IoT. However, the lack of additional overall metrics makes it difficult to fully assess the model's performance across different aspects. Moreover, the hybrid DL may not be appropriate for implementation on constrained IoT devices. The reason for this is that these DL algorithms involve a multitude of parameters and intricate computations, which collectively result in an increased demand for computational resources and storage space. In contrast, DT might be a more practical solution in such environments. Similarly, [46] does not mention computational efficiency, which is a critical factor in WSNs. They employed various feature extraction techniques and eventually fed the reduced dataset into a feed-forward neural network.

Table XV
COMPARISON BETWEEN THE PREVIOUS STUDIES.

| Classifiers | Features | Accuracy (%) | Precision (%) | Recall (%) | F1 Score (%) | Time(s) |
|---|---|---|---|---|---|---|
| [24] | 10 | 99.6 | 96.5 | 99.6 | 98.03 | 544.47 |
| [25] | 15 | 99.11 | 99.08 | 99.11 | 99.08 | N/A |
| [26] | 3 | 98.03 | N/A | N/A | N/A | 211.19 |
| [43] | 38 | 99.6 | 94.3 | 96.9 | 99.3 | 455.317 |
| [44] | 64 | 99.80 | 99.85 | 99.94 | 99.89 | 27719.36 |
| [45] | N/A | 98.99 | N/A | N/A | N/A | N/A |
| [46] | 39 | 99.8 | 98.7 | 97.7 | 98.7 | N/A |
| **Proposed** | **9** | **99.65** | **99.64** | **99.65** | **99.64** | **2.26** |

## V. CONCLUSION AND FUTURE WORK

The implementation of intelligent intrusion detection in resource-constrained WSNs presents significant technical challenges and necessitates a low-power detection model capable of operating efficiently under such limitations. From a system perspective, this also helps extend the lifespan of sensor nodes [47]. To balance security and power consumption, this study employed IPCA to transform the original data space into a set of linearly independent vectors. To preserve the overall data structure, the optimal number of principal components and batch size were determined. Subsequently, HHO was applied to select the most informative components, resulting in an 87% reduction in data dimensionality. This substantially alleviated the computational and memory burden imposed by the detection algorithm. The proposed method also demonstrated strong detection performance and practical applicability, as validated through deployment and testing on Raspberry Pi 5.

The imbalance in the CICIDS2017 dataset may adversely affect the model's performance [48]. Future work will investigate techniques such as SMOTE [49] to address this issue. Moreover, as attack behaviors in the Internet of Things (IoT) continue to evolve, models trained offline may become less effective. To address this, future research will explore the potential of online learning. IPCA inherently supports online learning by incrementally updating principal components through sliding-window-based batch processing. However, the traditional HHO algorithm is not natively compatible with dynamic data streams. One promising direction is to adapt the optimization strategy across successive data windows to reflect evolving data distributions. For instance, HHO could inherit the optimal component subset from the previous window to avoid redundant computations. Additionally, a memory mechanism could be introduced to retain promising solutions from prior iterations as initial candidates, while randomly reinitializing the rest [50], thereby mitigating the risk of local optima. Furthermore, the search intensity, population size, and iteration count could be adaptively tuned to minimize latency. During the detection phase, online models such as Hoeffding Trees may replace traditional decision trees. A concept drift detection module [51] can also be integrated to monitor changes in data distribution and trigger reinitialization of IPCA and HHO, followed by retraining or structural updates to the detection model.

## REFERENCES

[1] K. Rose, S. Eldridge, and L. Chapin, ‘The Internet of Things: An Overview’, vol. 80, no. 15, pp. 1–53, 2015.

[2] M. A. Jamshed, K. Ali, Q. H. Abbasi, M. A. Imran, and M. Ur-Rehman, ‘Challenges, Applications, and Future of Wireless Sensors in Internet of Things: A Review’, *IEEE Sensors J,* vol. 22, no. 6, pp. 5482–5494, Mar. 2022.

[3] M. A. Khan and K. Salah, ‘IoT security: Review, blockchain solutions, and open challenges’, *Future Generation Computer Systems*, vol. 82, pp. 395–411, May. 2018.

[4] M. Abedini and I. Al-Anbagi, "Active eavesdroppers detection system in multi-hop wireless sensor networks," *2022 IEEE Symposium on Computers and Communications (ISCC)*, Rhodes, Greece, 2022, pp. 1–6.

[5] Y. Kumar and V. Kumar, ‘A Systematic Review on Intrusion Detection System in Wireless Networks: Variants, Attacks, and Applications’, *Wireless Pers Commun*, vol. 133, no. 1, pp. 395–452, Nov. 2023.

[6] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, ‘Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions’, *J Ambient Intell Human Comput*, vol. 15, no. 2, pp. 1625–1642, Feb. 2024.

[7] Z. Huanan, X. Suping, and W. Jiannan, ‘Security and application of wireless sensor network’, *Procedia Computer Science*, vol. 183, pp. 486–492, 2021.

[8] B. Kaur, M. Rakhra, D. Singh, A. Singh, and Shruti, “Advancements in lightweight cryptography: Secure solutions for resource-constrained environments in IoT, WSNs, and CPS,” *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, 2024, pp. 1–7.

[9] L. Harn, C.-F. Hsu, Z. Xia, and Z. He, ‘Lightweight Aggregated Data Encryption for Wireless Sensor Networks (WSNs)’, *IEEE Sens. Lett.*, vol. 5, no. 4, pp. 1–4, Apr. 2021,.

[10] P. Dewal, G. S. Narula, V. Jain, and A. Baliyan, ‘Security Attacks in Wireless Sensor Networks: A Survey’, in Cyber Security, vol. 729, M. U. Bokhari, N. Agrawal, and D. Saini, Eds., in Advances in Intelligent Systems and Computing, vol. 729. , Singapore: Springer Singapore, 2018, pp. 47–58.

[11] J.-Y. Yu, E. Lee, S.-R. Oh, Y.-D. Seo, and Y.-G. Kim, ‘A Survey on Security Requirements for WSNs: Focusing on the Characteristics Related to Security’, *IEEE Access*, vol. 8, pp. 45304–45324, 2020.

[12] M. Faris, M. N. Mahmud, M. F. M. Salleh, and A. Alnoor, ‘Wireless sensor network security: A recent review based on state-of-the-art works’, *International Journal of Engineering Business Management*, vol. 15, p. 18479790231157220, Feb. 2023.

[13] I. Tomic and J. A. McCann, ‘A Survey of Potential Security Issues in Existing Wireless Sensor Network Protocols’, *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1910–1923, Dec. 2017.

[14] D. E. Boubiche, S. Athmani, S. Boubiche, and H. Toral-Cruz, ‘Cybersecurity Issues in Wireless Sensor Networks: Current Challenges and Solutions’, *Wireless Pers Commun*, vol. 117, no. 1, pp. 177–213, Mar. 2021.

[15] O. A. Osanaiye, A. S. Alfa, and G. P. Hancke, ‘Denial of Service Defence for Resource Availability in Wireless Sensor Networks’, *IEEE Access*, vol. 6, pp. 6975–7004, 2018.

[16] S. Ismail, D. W. Dawoud, and H. Reza, ‘Securing Wireless Sensor Networks Using Machine Learning and Blockchain: A Review’, *Future Internet*, vol. 15, no. 6, p. 200, May. 2023.

[17] R. Ahmad, R. Wazirali, and T. Abu-Ain, ‘Machine Learning for Wireless Sensor Networks Security: An Overview of Challenges and Issues’, *Sensors*, vol. 22, no. 13, Art. no. 13, Jan. 2022.

[18] M. Mamdouh, M. A. I. Elrukhsi, and A. Khattab, ‘Securing the Internet of Things and Wireless Sensor Networks via Machine Learning: A Survey’, in *2018 International Conference on Computer and Applications (ICCA)*, Beirut: IEEE, Aug. 2018, pp. 215–218.

[19] P. R. Chandre, P. N. Mahalle, and G. R. Shinde, ‘Deep Learning and Machine Learning Techniques for Intrusion Detection and Prevention in Wireless Sensor Networks: Comparative Study and Performance Analysis’, in *Design Frameworks for Wireless Networks*, Singapore: Springer Singapore, 2020, pp. 95–120.

[20] A. Goyal, S. Mishra, and V. K. Chaurasiya, ‘Intrusion Detection in Wireless Sensor Networks Using Deep Learning’, in *2023 4th International Conference for Emerging Technology (INCET)*, Belgaum, India: IEEE, May. 2023, pp. 1–13.

[21] R. Wazirali and R. Ahmad, ‘Machine Learning Approaches to Detect DoS and Their Effect on WSNs Lifetime’, *Computers, Materials & Continua*, vol. 70, no. 3, pp. 4922–4946, 2022.

[22] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, ‘Network intrusion detection system: A systematic study of machine learning and deep learning approaches’, *Trans Emerging Tel Tech*, vol. 32, no. 1, p. e4150, Jan. 2021.

[23] S. Otoum, B. Kantarci, and H. T. Mouftah, ‘On the Feasibility of Deep Learning in Sensor Network Intrusion Detection’, *IEEE Netw. Lett.*, vol. 1, no. 2, pp. 68–71, Jun. 2019.

[24] R. Abdulhammed, M. Faezipour, H. Musafer, and A. Abuzneid, ‘Efficient Network Intrusion Detection Using PCA-Based Dimensionality Reduction of Features’, in *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, Istanbul, Turkey: IEEE, Jun. 2019, pp. 1–6.

[25] S. Roy, J. Li, B.-J. Choi, and Y. Bai, ‘A lightweight supervised intrusion detection mechanism for IoT networks’, *Future Generation Computer Systems*, vol. 127, pp. 276–285, Feb. 2022.

[26] W. L. Al-Yaseen, ‘Improving intrusion detection system by developing feature selection model based on firefly algorithm and support vector machine,’ *IAENG Int. J. Comput. Sci.*, vol. 46, no. 4, pp. 534–540, 2019.

[27] M. A. Elsadig, ‘Detection of Denial-of-Service Attack in Wireless Sensor Networks: A Lightweight Machine Learning Approach’, *IEEE Access*, vol. 11, pp. 83537–83552, 2023.

[28] M. He, Y. Huang, X. Wang, P. Wei, and X. Wang, ‘A Lightweight and Efficient IoT Intrusion Detection Method Based on Feature Grouping’, *IEEE Internet Things J.*, vol. 11, no. 2, pp. 2935–2949, Jan. 2024.

[29] G. A. Mukhaini, M. Anbar, S. Manickam, T. A. Al-Amiedy, and A. A. Momani, ‘A systematic literature review of recent lightweight detection approaches leveraging machine and deep learning mechanisms in Internet of Things networks’, *Journal of King Saud University - Computer and Information Sciences*, vol. 36, no. 1, p. 101866, Jan. 2024.

[30] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, ‘Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization’:, in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116.

[31] V. N. G. Raju, K. P. Lakshmi, V. M. Jain, A. Kalidindi, and V. Padma, ‘Study the Influence of Normalization/Transformation process on the Accuracy of Supervised Classification’, in *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, Tirunelveli, India: IEEE, Aug. 2020, pp. 729–735.

[32] H. Abdi, L.J. Williams, Principal Component Analysis, Wiley interdisciplinary reviews: computational statistics. 2 (2010) 433-459.

[33] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, ‘Incremental Learning for Robust Visual Tracking’, *Int J Comput Vis*, vol. 77, no. 1–3, pp. 125–141, May 2008.

[34] A. Levy and M. Lindenbaum, ‘Sequential Karhunen–Loeve Basis Extraction and its Application to Images’, *IEEE Transactions on Image Processing*, vol. 9, no. 8, 2000.

[35] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, ‘Harris hawks optimization: Algorithm and applications’, *Future Generation Computer Systems*, vol. 97, pp. 849–872, Aug. 2019,

[36] R. F. Bikmukhamedov and A. F. Nadeev, ‘Lightweight Machine Learning Classifiers of IoT Traffic Flows’, in *2019 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, Russia: IEEE, Jul. 2019, pp. 1–5.

[37] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, ‘Practical real-time intrusion detection using machine learning approaches’, *Computer Communications*, vol. 34, no. 18, pp. 2227–2235, Dec. 2011.

[38] N. Tekin, A. Acar, A. Aris, A. S. Uluagac, and V. C. Gungor, ‘Energy consumption of on-device machine learning models for IoT intrusion detection’, *Internet of Things,* vol. 21, p. 100670, Apr. 2023

[39] K. Das and R. N. Behera, ‘A survey on machine learning: concept, algorithms and applications’, *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 2, pp. 1301–1309, 2017.

[40] M. Hosny, K. Magdi, A. Salah, O. El-Komy, and N. A. Lashin, "Internet of Things Applications Using Raspberry-Pi: A Survey," *Int. J. Electr. Comput. Eng.*, vol. 13, no. 1, pp. 902-910, 2023.

[41] T. Kim, L. F. Vecchietti, K. Choi, S. Lee, and D. Har, ‘Machine Learning for Advanced Wireless Sensor Networks: A Review’, *IEEE Sensors J.*, vol. 21, no. 11, pp. 12379–12397, Jun. 2021.

[42] S. M. Lundberg and S. I. Lee, ‘A unified approach to interpreting model predictions’, *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[43] I. H. Hassan, M. Abdullahi, M. M. Aliyu, S. A. Yusuf, and A. Abdulrahim, ‘An improved binary manta ray foraging optimization algorithm based feature selection and random forest classifier for network intrusion detection’, *Intelligent Systems with Applications*, vol. 16, p. 200114, Nov. 2022.

[44] A. Thakkar and R. Lohiya, ‘Fusion of statistical importance for feature selection in Deep Neural Network-based Intrusion Detection System’, *Information Fusion*, vol. 90, pp. 353–363, Feb. 2023.

[45] A. Nazir, J. He, N. Zhu, S.S. Qureshi, S.U. Qureshi, F. Ullah, A. Wajahat, M.S. Pathan, ‘A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem’, *Ain Shams Engineering Journal*, vol. 15, no. 7, p. 102777, Jul. 2024.

[46] M. H. Behiry and M. Aly, ‘Cyberattack detection in wireless sensor networks using a hybrid feature reduction technique with AI and machine learning methods’, *J Big Data*, vol. 11, no. 1, p. 16, Jan. 2024.

[47] S. Rajasegarar, C. Leckie, and M. Palaniswami, ‘Anomaly Detection in Wireless Sensor Networks’, *IEEE Wireless Communications*, 2008.

[48] J. Liu, Y. Gao, and F. Hu, ‘A fast network intrusion detection system using adaptive synthetic oversampling and LightGBM’, *Computers & Security*, vol. 106, p. 102289, Jul. 2021.

[49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, ‘SMOTE: Synthetic Minority Over-sampling Technique’, *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002.

[50] X.-F. Liu, Z.-H. Zhan, and J. Zhang, ‘Incremental particle swarm optimization for large-scale dynamic optimization with changing variable interactions’, *Applied Soft Computing*, vol. 141, p. 110320, Jul. 2023.

[51] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, ‘A survey on concept drift adaptation’, *ACM Comput. Surv*., vol. 46, no. 4, pp. 1 – 37, Apr. 2014.