Towards Reliable IoT Security: A Deterministic Arithmetic Optimization Algorithm for Wrapper-Based Feature Selection in Intrusion Detection Systems

Taha M.O. Alakhras

P5189@pps.umt.edu.my

Universiti Malaysia Terengganu

Waheed A. H. M Ghanem

Universiti Malaysia Terengganu

Farizah Yunus

Universiti Malaysia Terengganu

Sanaa A. A Ghaleb

Universiti Sultan Zainal Abidin, Besut, Terengganu, Malaysia

Mohammed Otair

Middle East University, Amman, Jordan

Research Article

Keywords: arithmetic optimization algorithm, feature selection, binary optimization, classification

Posted Date: December 29th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-8305469/v1

Additional Declarations: No competing interests reported.

Towards Reliable IoT Security: A Deterministic Arithmetic Optimization Algorithm for Wrapper-Based Feature Selection in Intrusion Detection Systems

Taha M.O. Alakhras[1], Waheed A. H. M. Ghanem[1,3,4], Farizah Yunus[1], Sanaa A. A. Ghaleb[2,5], Prof. Mohammed Otair[5]

Abstract

Today use computer networks all the time for everything—our phones, computers, internet of thing (IoT), and cloud services. Because of this, networks often get attacked by things like denial of service (DoS), user to remote attack (U2R) We try to stop these attacks with Intrusion Detection Systems (IDSs). However, today's IDSs struggle to find brand-new types of attacks. To make them work better, we first need to pick out only the most useful features of information before the system runs. This paper introduces a Deterministic version of the Arithmetic Optimization Algorithm (DAOA) to solve the feature selection problem in classification. The classifier employs K-Nearest Neighbors (KNN) using a wrapper-based approach. to find the optimal solutions. In contrast, all previous studies have introduced a probabilistic version of the Arithmetic Optimization Algorithm (BAOA). This study uses NF-UNSW-NB15-V2 dataset as benchmark datasets from the collection by the university of Queensland The results demonstrate that DAOA outperformed the Binary Arithmetic Optimization Algorithm(BAOA), Binary Grey Wolf Optimizer (GWO), Binary Particle Swarm Optimization (BPSO), and Binary Harmony Search optimization (HS),Binary Ant Colony Optimization for Real-valued domains (ACOR), when various performance metrics were used, including classification accuracy, selected features, The tested algorithms were ranked using the Friedman Test, and pairwise comparisons were performed using the Wilcoxon Signed-Rank Test. After running the algorithms for 30 iterations and 20 epochs, the results showed that the DAOA achieved the highest classification accuracy while selecting the smallest feature set compared to all other tested algorithms.

Keywords: arithmetic optimization algorithm, feature selection, binary optimization, classification.

1. Introduction

The volume of data generated across modern digital resources has been increasing rapidly in recent years. [1], Data mining is a knowledge-driven analytical process that encompasses a variety of methods drawn from machine learning, statistics, and database systems [2]. In recent years, metaheuristic algorithms have attracted significant research interest due to their strong potential for solving complex problems across a wide range of domains [3], High dimensional challenges are becoming increasingly common as the number of variables and the complexity of modern problems continue to grow. Recently, several metaheuristic algorithms, such as the Dipper Throated Optimization (DTO), Battle Royale Optimization (BRO), Waterwheel Plant Algorithm (WWPA), have been proposed to address these issues. [3], [4] [5]. In machine learning , dimensionality reduction trims the feature space and reshapes high-dimensional data into more compact, information-rich representations[6],[7], In this context, metaheuristic algorithms stand out for their strong global search ability and adaptive behavior, making them a key approach for tackling complex global optimization problems[8], Although optimization algorithms are frequently treated as flexible black-box tools capable of handling diverse and complex problems, this flexibility does not ensure that every algorithm described in the literature can reliably reach the global optimum across all optimization tasks, Although optimization algorithms are frequently treated as flexible black-box tools capable of handling diverse and complex problems, this flexibility does not ensure that every algorithm described in the literature can reliably reach the global optimum across all optimization tasks [9], The AOA offers a straightforward

Taha M.O. Alakhras1
P5189@pps.umt.edu.my
Waheed A. H. M. Ghanem[1,3,4]
waheedghanem@umt.edu.my
Farizah Yunus1
farizah.yunus@umt.edu.my
Sanaa A. A. Ghaleb[2,5]
sanaaabduljabbar@unisza.edu.my.
Prof. Mohammed Otair[5]
m.otair@meu.edu.jo
[1]Faculty of Computer Science and Mathematics, Universiti Malaysia Terengganu, Terengganu, Malaysia.
[2]Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Besut, Terengganu, Malaysia
[3]Faculty of Engineering, University of Aden, Aden, Yemen
[4]Faculty of Science and Education, University of Lahej, Lahej, Yemen.
[5]Faculty of Education, University of Aden, Yemen
[5]Department of Computer Science, Faculty of Information Technology, Middle East University, Amman, Jordan

structure, adaptable parameter settings, and effective global search behavior. Yet its exploitation capability remains limited, especially when tackling complex high-dimensional or multi-modal problems, where it can easily become trapped in local optima and exhibit slow convergence, The AOA's position-update mechanism does not introduce enough randomness into the movement of search agents, which increases the likelihood of getting stuck in local optima, Moreover, the AOA's limited exploitation capability reduces its consistency in achieving optimal parameter settings when applied to Proportional-Integral-Derivative (PID) controller optimization, The AOA has been effectively applied across multiple domains, including data clustering, image segmentation, energy storage grid management, feature recognition, neural network modeling, and engineering design[10]. Some AOA variants have also been employed to address multi-objective optimization problems [11]. Figure 1 provides a concise view of how feature selection works. It starts with a dataset containing all available features, which is then processed by a feature selection algorithm to produce a candidate subset. This subset is evaluated against predefined selection criteria. If the subset does not meet the criteria, the process loops back to refine the selection. If it does meet the criteria, the validated results are accepted as the final output. This study introduces a deterministic variant of the Arithmetic Optimization Algorithm (DAOA) to overcome the instability and inconsistent search behavior of the original probabilistic AOA in feature selection. By eliminating random fluctuations in position updates, DAOA delivers more reliable convergence, enhanced global search, and smaller feature subsets for IoT intrusion detection. DAOA is benchmarked against five established binary metaheuristics (BAOA, BPSO, GWO, BHS, ACOR) on the NF-UNSW-NB15-V2 dataset, using a KNN wrapper, 30 independent runs, and multiple metrics. Statistical significance is confirmed via Friedman ranking, Wilcoxon signed-rank, and Nemenyi post-hoc tests. The results demonstrate that the deterministic AOA formulation provides a more stable, accurate, and feature-efficient solution for IoT intrusion detection. Accordingly, the primary contributions and novelty of this study can be summarized as follows:

1. The proposed DAOA is a fully deterministic reformulation of the Arithmetic Optimization Algorithm, eliminating all sources of randomness and achieving more stable behavior, reliable convergence, and enhanced global exploration capability.
2. A KNN-based wrapper feature-selection framework is established using the proposed DAOA to reliably assess classification performance over compact feature subsets.
3. Performed a rigorous benchmark evaluation against five leading binary metaheuristics (BAOA, BPSO, GWO, BHS, and ACOR) on the NF-UNSW-NB15-V2 dataset across 30 independent runs.
4. Applied a rigorous statistical validation pipeline including the Friedman ranking test, pairwise Wilcoxon signed-rank test, and Nemenyi post-hoc analysis to ensure that all observed performance improvements are statistically significant.
5. Demonstrated that DAOA consistently achieves the highest accuracy, the smallest feature subsets, and the most stable performance, positioning it as a highly effective solution for resource-constrained IoT intrusion detection.

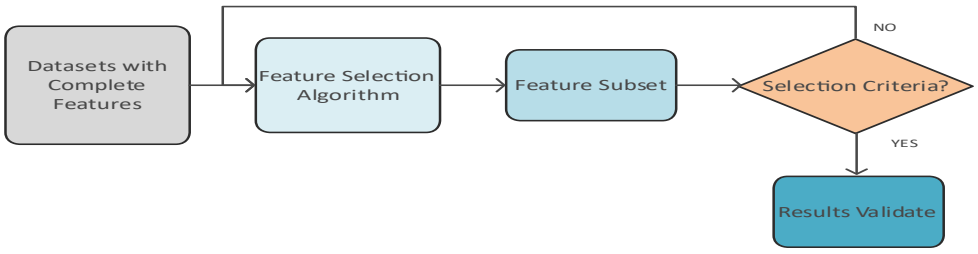

Figure 1:Feature Selection process Flowchart

Section 2 reviews the Related Works, followed by Section 3, which presents the proposed feature selection method based Arithmetic Optimization Algorithm. then outline the Experiment Setup in Section 4 and the Evaluation Metrics in Section 5. Section 6 covers the Results and Discussion, and the paper ends with the Conclusions and Future Work in Section 7.

2. Related Work

This section reviews several modified AOA implementations that have been successfully applied to feature-selection problems. Although optimization algorithms are often treated as flexible black-box solvers for

complex problems, this does not guarantee that the methods reported in the literature will consistently reach the global optimum across different optimization [9], in [11] the authors proposed two binary variants of AOA-BAOA-V and BAOA-S for feature selection on high-resolution medical images for tumor detection. BAOA-V employs a hyperbolic tangent transfer function, whereas BAOA-S uses a sigmoid function to convert the standard AOA into a binary form. Between the two, BAOA-S achieved superior performance by selecting smaller and more relevant feature subsets compared to BAOA-V. Zakeri and Hakimabad introduced a feature-selection approach inspired by an analytical cooperation model among grasshoppers during food-source exploration. The effectiveness of the method was demonstrated through comparisons with several well-established feature-selection [12]. Ghanam et al in 2021 Developed a metaheuristic IDS framework employing a multi-objective feature-selection strategy that enhances detection performance while simultaneously reducing feature dimensionality [8]. Seghir Fateh et al proposed a hyperlearning binary dragonfly algorithm to address the feature-selection problem in COVID-19 diagnosis. The method was further evaluated on 21 UCI datasets, demonstrating improved classification accuracy along with a reduction in the number of selected features [13], Zhou Shengchao et al proposed an Adaptive Differential Evolutionary Algorithm to solve the problem of assigning jobs to batches without breaking the machine capacity constraints, and then sequencing the batches to minimize the completion time[14]. Yi Jiao-Hong et all proposed an improved version of the NSGA-III algorithm (INSGA-III) by introducing the Stud concept and incorporating enhanced crossover operators-SBX, UC, and SI-which collectively reduced the computational cost of solving large-scale optimization [15]. In another study Pashaei & Pashaei propose AOA was hybridized with Simulated Annealing (SA) and integrated with a filter-based method for feature selection in high-dimensional cancer gene-expression data. A crossover mechanism was additionally introduced to strengthen the exploratory capability of the hybrid approach. The method was evaluated on ten gene-expression datasets to assess its overall performance [16]. In Zivkovic Miodrag, the authors proposed a k-NN-AOA hybrid method for detecting fake news during the COVID-19 pandemic by enhancing k-NN classification accuracy through the selection of relevant feature subsets. The approach was applied to the real-world Koirala dataset and compared against several feature-selection techniques combined with the k-NN classifier, demonstrating superior performance [17], Feature-selection research has utilized a broad spectrum of metaheuristic families. Swarm-intelligence algorithms include Grey Wolf Optimizer [18],[19], Competitive Swarm Optimizer [20], Dipper Throated [21], Cat Swarm [22]], Chaotic Dragonfly [23], Krill Herd Optimizer [24], Whale Optimization Algorithm, and Harris Hawks [25]. Evolutionary-based methods comprise Genetic [26] and Bat Algorithm [27], whereas physics-inspired and mathematical models include Gradient-Based Optimizer [28]. Stochastic Fractal Search [29], Sine–Cosine Optimizer [30], Multi-Verse Optimizer [31] Firefly [32], and Moth-Flame Optimization [33]. Collectively, these algorithms represent the most widely adopted optimization approaches for tackling feature-selection tasks across diverse application domains.

3. Methodology

The original AOA, BAOA and the proposed DAOA are briefly discussed in this section.

3.1 Arithmetic Optimization Algorithm (AOA)

The Arithmetic Optimization Algorithm (AOA) is a recently introduced metaheuristic algorithm by [34]. It relies on fundamental arithmetic operators, including addition, subtraction, multiplication, and division. By applying these operators to a set of solutions, the algorithm aims to derive the optimal element through mathematical optimization. Exploration is facilitated through multiplication and division, allowing for significant changes. However, as these operators exhibit high dispersion and are unsuitable for local search, the algorithm incorporates addition and subtraction operators for exploitation or local search. AOA is a population-based algorithm wherein initial solutions are represented as $X_i = [X_i^1, X_i^2, \ldots, X_1^d]$,are randomly generated over a d-dimensional search space using Eq. 1

$$X_i^j = X_{min}^j + r\left(X_{max}^j - X_{min}^j\right) \; i = \{1,2,3,\ldots,N\}, \; j = \{1,2,3,\ldots,d\} \qquad (1)$$

Where N is population size, Xi represented the j[th] solution, $\boldsymbol{X_i^j}$ represented the j[th] dimension of the i[th] solution, X_{min}^j *and* X_{max}^j the uber and lower bound in the search space for jth dimension and r is random number between 0 to 1, and the first solution of X represented by matrix as show:

$$\boldsymbol{X} = \begin{bmatrix} \mathrm{X}_1^1 & \cdots & \mathrm{X}_1^{\mathrm{d}} \\ \vdots & \ddots & \vdots \\ \mathrm{X}_{\mathrm{N}}^1 & \cdots & \mathrm{X}_{\mathrm{N}}^{\mathrm{d}} \end{bmatrix} \quad (2)$$

A fitness function is established to assess the quality of each solution within the population during an iteration. The candidate solution with the highest fitness value in each iteration is regarded as the most optimal solution identified thus far. The decision on whether to prioritize exploration or exploitation is determined by the Math Optimizer Accelerated (MOA) function, calculated as shown in Eq. (3). This function yields a coefficient based on the current iteration C_{Iter}, which is utilized in the search phases.

$$MOA(C_{Iter}) = \mathrm{Min} + C_{Iter} * \left(\frac{Max - Min}{Max_{Iter}} \right) \quad (3)$$

Where $\boldsymbol{C_{Iter}}$ represents the current iteration, $\boldsymbol{Max_{Iter}}$ denotes the maximum number of iterations, $\boldsymbol{Max}$ and $\boldsymbol{Min}$ are constants indicating the maximum and minimum possible values of MOA respectively. The MOA is formulated to favor exploration in the initial stages and exploitation in the later iterations. A random number, r1, within the range [0, 1] is generated, and its value is compared with MOA. If r1 > MOA, exploration is performed; otherwise, exploitation takes place. **Exploration Phase**: During this phase, the solution space undergoes exploration utilizing division and multiplication operators. For exploration, either the division or multiplication operator is randomly selected with equal probabilities. The calculation for the new solution is represented by Eq. (4)

$$\mathrm{X_i^j}(\mathrm{C_{Iter}}+1) \begin{cases} \mathrm{best(X_J)} \div (\mathrm{MOP}+\epsilon) \times ((\mathrm{X_{max}^j} - \mathrm{X_{min}^j}) \times \mu + \mathrm{X_{min}^j}) \times \mu + \mathrm{X_{min}^j}), \mathrm{r2} < 0.5 \\ \mathrm{best(X_j)} \times \mathrm{MOP} \times ((\mathrm{X_{max}^j} - \mathrm{X_{min}^j}) \times \mu + \mathrm{X_{min}^j}) \times \mu + \mathrm{X_{min}^j}), \mathrm{otherwise} \end{cases} \quad (4)$$

Where, $X_i^j(\boldsymbol{C_{Iter}} + \mathbf{1})$ represents the j[th] dimension of the ith solution in the next iteration, $\boldsymbol{best(X_J)}$ represents the j[th] dimension in the current best solution, $\boldsymbol{\epsilon}$ is a small non-zero number, μ is a control parameter to adjust the search process set to 0.5 as authors, r2 is a random number between [0,1], and MOP is a Math Optimizer Function calculated in each iteration using Eq. 5.

$$\mathrm{MOP}(\mathrm{C_{Iter}}) = 1 - \frac{{\mathrm{C_{Iter}}}^{1/\propto}}{{\mathrm{M_{Iter}}}^{1/\propto}} \quad (5)$$

Where, ∝ is the sensitivity parameter set to a value of 5. **Exploitation Phase**: In this phase, an in-depth exploration of solutions occurs, aiming to find the **optimal** solution in the vicinity of the best solution. The operators utilized in this phase are addition and subtraction. Similar to the exploration phase, the probability of selecting operators during exploitation is also equal. The new solutions are calculated as shown in Eq. 6.

$$\mathrm{X_i^j}(\mathbf{C_{Iter}} + \mathbf{1}) = \begin{cases} \mathrm{best(X_j} - \mathrm{MOP} \times ((\mathrm{X_{max}^j} - \mathrm{X_{min}^j}) \times \mu + \mathrm{X_{min}^j}, & \mathrm{r3} < 0.5 \\ \mathrm{best}\left(\mathrm{X_j}\right) + \mathrm{MOP} \times \left(\left(\mathrm{X_{max}^j} - \mathrm{X_{min}^j}\right) \times \mu + \mathrm{X_{min}^j}\right), & \mathrm{otherwise} \end{cases} \quad (6)$$

Where, r3 is a random number between [0,1]. Complete information on the AOA algorithm's inspiration and mathematical model is available in[34].

3.2 Binary Arithmetic Optimization Algorithm (BAOA).

Feature selection is inherently a discrete binary problem, Thus the original AOA described in Section (A), cannot direct be utilized to Address such problems [2], These studies employed four common families of binary transfer functions (BTFs), including the S-shaped, V-shaped, Z-shaped, and U-shaped families [35].

Algorithm 1: Pseudo-Code of the Standard BAOA

1	Start.
2	Data Preprocessing: Load Dataset -> Balance Classes -> Encode Categorical Data -> Split (Train/Test).
3	Initialize Population: Generate random positions (X) in continuous space [0, 1].
4	Evaluate Fitness: Calculate Accuracy using KNN classifier.
5	Start Main Loop (while t < Max_Iter):
6	Update MOP and MOA parameters.
7	FOR each solution (i):
8	IF r1 > MOA (Exploration Mode):
9	Update position using Division or Multiplication operators.
10	ELSE (Exploitation Mode):
11	Update position using Subtraction or Addition operators.
12	Apply common families of binary transfer functions (BTFs) Binarize: = S(X)
13	If rand < S(x) then 1, else 0.
14	Evaluate Fitness of the new binary solutions.
15	Update Global Best Solution (X_best).
16	t = t + 1.
17	End Main Loop.
18	Output Results: Best Accuracy, Selected Features.
19	End.

As we show in the preview's pseudo code Algorithm 1 the main step to convert function from AOA to BAOA that step 12 and 13, So These studies employed four common families of binary transfer functions (BTFs), including the S-shaped, V-shaped, Z-shaped, and U-shaped families AS illustrate in table 1 [35].

Table 1:common families of binary transfer functions

s-shaped	v- shaped	z- shaped	u- shaped
$S_1(x) = \frac{1}{1+e^{-2x}}$	$V_1(x) = \left\lvert \operatorname{erf}\left(\frac{\sqrt{\pi}}{2}x\right)\right\rvert$	$Z_1(x) = \lvert\sqrt{1-2^x}\rvert$	$u_1(x) = \lvert x\rvert^{1.5}$
$S_2(x) = \frac{1}{1+e^{-x}}$	$V_2(x) = \lvert\tan(x)\rvert$	$Z_1(x) = \lvert\sqrt{1-5^x}\rvert$	$u_2(x) = \lvert x\rvert^2$
$S_3(x) = \frac{1}{1+e^{-2}}$	$V_3(x) = \left\lvert \frac{x}{\sqrt{1+x^2}}\right\rvert$	$Z_1(x) = \lvert\sqrt{1-8^x}\rvert$	$u_3(x) = \lvert x\rvert^3$
$S_4(x) = \frac{1}{1+e^{-3}}$	$V_2(x) = \left\lvert \frac{2}{\pi}\arctan\left(\frac{\pi}{2}x\right)\right\rvert$	$Z_1(x) = \lvert\sqrt{1-20^x}\rvert$	$u_4(x) = \lvert x\rvert^4$

3.3 Proposed Deterministic Arithmetic Optimization Algorithm (DAOA)

This study introduces the Deterministic Arithmetic Optimization Algorithm for Feature Selection (DAOA-FS), a groundbreaking paradigm shifts from the conventional probabilistic binarization approach in AOA-based feature selection to a fully deterministic binarization strategy. Unlike existing methods that rely on stochastic transfer

functions (e.g., Sigmoid, V-shaped, S-shaped, etc.) followed by a random threshold (r_4), which introduce uncontrolled variability and reduce solution consistency across independent runs, DAOA-FS eliminates randomness in the binary conversion stage entirely. Instead, the binarization is performed deterministically using a simple rounding operation (round(x)) applied externally within the objective function. This deterministic rounding ensures that any solution value ≥ 0.5 is consistently mapped to 1 (feature selected) and < 0.5 to 0 (feature discarded), regardless of the run. Experimental results on the NF-UNSW-NB15-V2 dataset demonstrate that this deterministic approach significantly enhances solution stability, reproducibility, and efficiency, achieving comparable or superior classification accuracy with substantially fewer selected features and near-zero variance across multiple runs. Figure 2 illustrates the flowchart of the proposed Deterministic Arithmetic Optimization Algorithm (DAOA), The process begins by loading and processing a pre-set IoT intrusion detection dataset, followed by random initialization of a continuous population then in each iteration the MOA and MOP parameters are updated to control the exploration and exploitation phases. Candidate solutions are updated in the continuous space using arithmetic operators (division, multiplication, subtraction, or addition) according to the current MOA value, differently from traditional probabilistic binarization DAOA applies a fully deterministic transformation by clipping positions to [0, 1] and directly rounding them to binary values (0 or 1), eliminating stochastic noise and enhancing stability. Following the transformation, the binary subsets are evaluated by a KNN classifier, and the global best solution is updated if a candidate solution yields better fitness, The optimization loop concludes once the maximum number of iterations is reached, returning the optimal feature subset along with its corresponding classification performance.

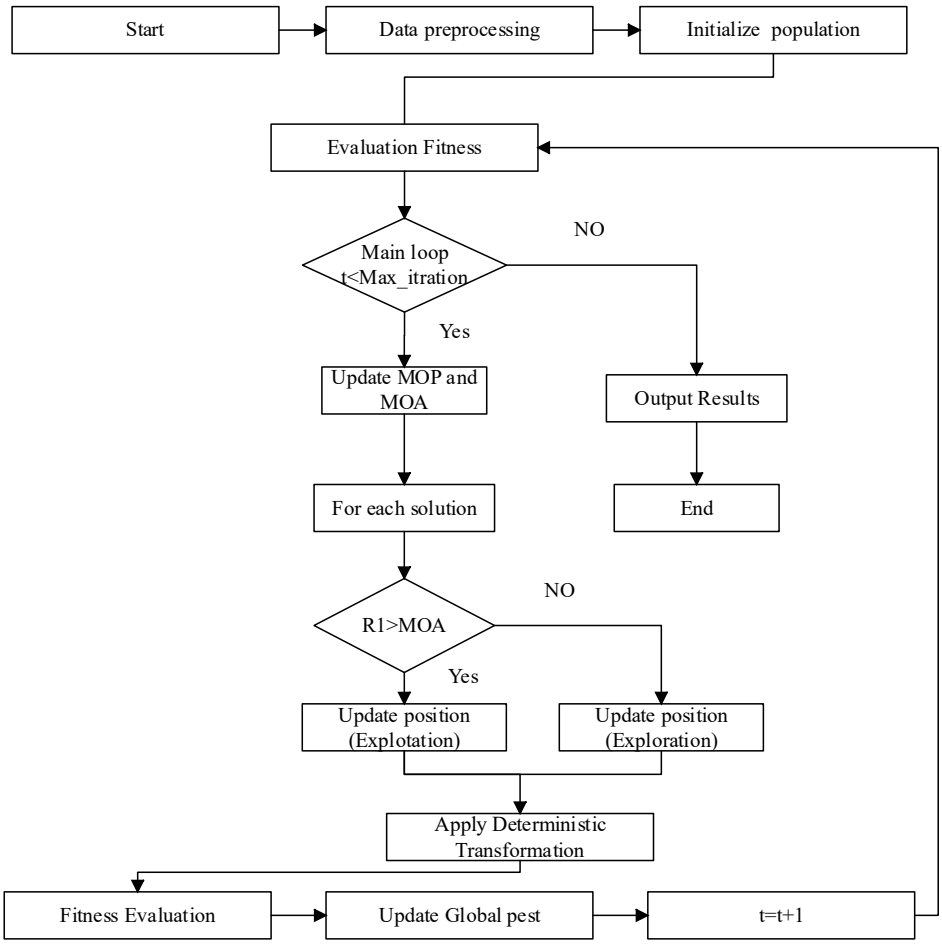

Figure 2:flowchart of the proposed Deterministic Arithmetic Optimization Algorithm (DAOA).

The algorithm 2 begins by preprocessing the dataset and initializing a continuous population $X \in [0,1]^d$ where each dimension encodes a feature-selection likelihood. During optimization, AOA's exploration and exploitation operators update the continuous positions while keeping them within the valid domain using clipping. A key contribution of the proposed method is the introduction of a deterministic transformation that replaces stochastic binarization as in Eq.7 his ensures a stable and reproducible mapping from continuous to binary space without random fluctuations.

$$X_{\text{bin}}(j) = \begin{cases} 1, & \text{if } X_{\text{cont}}(j) \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

The deterministic binarization rule in Eq. 6 operates on the continuous position vector X_{cont}, where each dimension represents the likelihood of selecting a specific feature. For clarity, the variables are defined as follows:

- $X_{\text{cont}}(j)$: the continuous value of the j-th feature in the range $[0,1]$produced by the DAOA update equations.
- $X_{\text{bin}}(j)$: the binary decision indicating whether the j-th feature is selected (1) or discarded (0).
- Threshold 0.5: a deterministic cutoff that replaces all probabilistic transfer functions used in BAOA and other binary metaheuristics.

3.3.1 Objective Function and Fitness Evaluation

The fitness evaluation uses this binary mask to select a subset of features and compute the weighted F1-score of a classifier (e.g. KNN), while penalizing large subsets to maintain compactness as in (Eq.8)

$$\text{Fitness} = \text{F1}_{\text{weighted}} - \lambda\left(\frac{\text{selected features}}{d}\right) \quad (8)$$

Where λ denoted Penalty Strength in script λ =0.001 where the deterministic rounding and penalty formulation enable the algorithm to balance prediction capability and subset minimization effectively. Throughout iterations, the global best solution is updated based on this objective, and the final output includes the optimal feature subset.

Algorithm 2: Pseudo-Code of the Proposed Deterministic AOA for Feature Selection (DAOAFS)

1	Start.
2	Data preprocessing.
3	Initialize population (continuous solutions X in Rd).
4	Initial fitness evaluation (using deterministic rounding inside the fitness function).
5	Set iteration counter t = 1.
6	While (t ≤ Max_Iter) do
7	Update MOP and MOA parameters.
8	For each solution i in the population do
9	If (r1 > MOA) (exploration) then update position using AOA math operators.
10	Else (exploitation) update position using AOA math operators.
11	Clip each dimension of X_i to the range [0, 1].
12	Pass the continuous vector X_i (in [0,1]) to the fitness function.
13	Inside fitness:
14	Apply deterministic transformation: X_bin = Round (X_i) (≥ 0.5 → 1, otherwise 0). *Eq 7*
15	Evaluate objective value (e.g., F1-score – penalty for #features). *Eq 8*
16	End for.
17	Update global best solution X_best based on fitness.
18	Set t = t + 1.
19	End while.
20	Output final selected feature subset and corresponding performance (improved efficiency).
21	End.

4. Experimental Setup

In this section, the description of the datasets used, parameter settings, and evaluation metrics are clearly displayed.

4.1 Dataset Description

The NF-UNSW-NB15-v2 dataset used in this study is a significantly enhanced NetFlow-based extension of the widely recognized UNSW-NB15 dataset, originally developed by the Cyber Range Lab of UNSW Canberra. The original dataset was generated using the IXIA PerfectStorm tool to simulate a realistic combination of normal network traffic and nine contemporary attack categories (Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms),

capturing approximately 100 GB of raw network traffic using tcpdump making it one of the most comprehensive and feature-rich publicly available NetFlow-based datasets for network intrusion detection systems (NIDS) in IoT, It include 2,390,275 network flows 95,053 attacks (3.98%) and 2,295,222 benign records (96.02%) provided in clean Parquet format with no missing values or duplicate, Due to computational and memory limitations, processing the full dataset was not feasible in this study, Consequently a balanced and representative subset comprising 20,000 samples was meticulously constructed. To maintain the integrity of the original class distribution, stratified under sampling was applied to the majority classes, specifically benign traffic and high-frequency attacks.[36].

4.2 Parameter Settings of Compared Algorithms

To ensure a fair and reproducible assessment, all experiments utilized the following standardized hyperparameter configuration. As presented in Table 2 the experimental parameters were deliberately chosen to strike an optimal balance between computational feasibility and sufficient search capability while ensuring statistical reliability. These compact yet effective settings fully adhere to established practices in high-dimensional feature selection research, and python 3.11 is used to code the algorithms, and all datasets are run on a computer 11th Gen Intel(R) Core (TM) i7-1165G7 @ 2.80GHz (2.80 GHz),12.0 GB (11.8 GB usable) Ram, Windows 11 64-bit operating system, x64-based processor.

Table 2:experimental parameters

Parameter	Value	Description
Population Size	15	Number of candidate solutions
Maximum Iterations (Epochs)	20	Maximum number of optimization iterations
Independent Runs	30	Number of independent executions (final results)
Classifier	KNN (k=5)	Base classifier for fitness evaluation
Fitness Function	5-fold cross-validated F1-weighted + 0.01 × (features/total features) penalty	This formulation simultaneously maximizes predictive performance and minimizes the number of selected features.
Cross-Validation Folds	5	Stratified k-fold cross-validation
Train/Test Split	70%/30%	Stratified hold-out validation
Random Seed Base	42	Base seed (incremented per run)
Dataset Subset Size	20,000	Balanced representative sample

5. Evaluation Metrics

The performance of all evaluated algorithms was assessed using standard intrusion-detection metrics derived from the confusion matrix, including Accuracy, Precision, Recall, F1-Score, Specificity. And prediction time, and feature-subset size. Accuracy and F1-Score were the primary metrics for statistical comparisons while Specificity and Precision were essential for measuring false alarm behavior [37]. The efficiency of feature selection was captured through the number of selected features and the real time prediction. For statistical significance, the Friedman test [38] was applied across 30 independent runs followed by Wilcoxon signed-rank post-hoc comparisons and the Nemenyi critical-difference diagram was used to visualize ranking stability.

5.1 Confusion-Matrix Metrics (Accuracy, Precision, Recall, F1, Specificity)

A confusion matrix is a table used to evaluate the performance of a machine learning model for classification tasks. It provides a summary of the number of correct and incorrect predictions made by the model on a set of test data [37].

$$\mathrm{CM} = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{9}$$

The confusion matrix contains four elements:

- True Positive (TP): The model predicted the positive class and the prediction is correct.

• False Positive (FP): The model predicted the positive class, but the prediction is incorrect.
• True Negative (TN): The model predicted the negative class and the prediction is correct.
• False Negative (FN): The model predicted the negative class, but the prediction is incorrect.

5.1.1 Accuracy:

The following formula Eq10 is used to determine total percentage of correct classifications.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (10)$$

This is the primary indicator in evaluating intrusion detection systems and is adopted as a key criterion in the summary table and the Friedman test.

5.1.2 Precision:

It measures the model's reliability when an attack is detected as shown in Eq11.

$$precision = \frac{TP}{TP + FP} \quad (11)$$

5.1.3 Recall (Detection Rate)

This reflects the model's ability to actually detect attacks as shown in Eq 12.

$$Recall = \frac{TP}{TP + FN} \quad (12)$$

5.1.4 F1-Score (Weighted / Macro)

It is used to measure the balance between Precision and Recall, and is the determining factor within the Fitness function for feature selection as shown in Eq 13.

$$\text{F1} - \text{Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (13)$$

5.1.5 Specificity

It measures the model's ability to correctly identify normal samples, which is important for reducing false alarms as shown in Eq 14.

$$Specificity = \frac{TN}{TN + FP} \quad (14)$$

5.2 Feature-Subset Size

The number of features selected is shown, and this metric is fundamental to measuring efficiency as shown in Eq 15:

$$selected\_features = \sum_{i=1}^{d} x_i \quad (15)$$

Where $x^i \in \{0,1\}$

5.3 Prediction Time

The measurement was performed accurately by calculating the prediction time only, without any training cost, because IoT systems require low real-time response times as shown in Eq 16.

$$T_{\text{predict}} = T_{\text{after\_predict}} - T_{\text{before\_predict}} \quad (16)$$

5.4 Convergence Behavior

Convergence curves for all optimization algorithms were recorded via Mean ± Std. across 30 iterations. These curves reflect: speed of stability, volatility levels, and the quality of solutions over time, and are included in the form of a Mean ± Std. graph as Eq (17).

$$Fitness_t = F1_{\text{weighted}} - \lambda\left(\frac{selected\_features}{d}\right) \quad (17)$$

Where $Fitness_t$ the value of the fitness function in the iteration t, λ a weighting factor controls the effect of the size penalty, The $F1_{\text{weighted}}$-Score value at iteration number t, which represents the quality of the rating, d total number of features in the dataset before the selection process.

5.5 Statistical Validation Methods

To assess the statistical significance of performance differences among the evaluated algorithms, the Friedman test was applied across multiple independent runs. Pairwise comparisons were then conducted using the Wilcoxon signed-rank test. Both procedures were implemented through the friedmanchisquare, Wilcoxon, and rankdata functions provided in the SciPy (Scientific Python). stats module, ensuring robust and validated statistical analysis. Furthermore, a Nemenyi post-hoc test was performed using the posthoc-nemenyi-friedman routine from the scikit-posthocs library, which offers non-parametric multiple-comparison methods for rank-based data [39]

5.5.1 Friedman Ranking Test

To assess the reliability of differences between algorithms, the Friedman test, a non-parametric test, was adopted to compare more than one algorithm on the same datasets and the transformation was performed using Eq. 18 [40].

$$\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_{j=1}^{k} R_j^2\right) - 3N(k+1) \quad (18)$$

Where χ_F^2 Friedman test statistics, N number of runs, k number of algorithms being compared, R_j is the sum of ranks obtained by algorithm across all runs, j is the algorithm index in the comparison (from 1 to k)

5.5.2 Nemenyi Post-Hoc Analysis

To further analyze multiple algorithm comparisons following the Friedman test, the Nemenyi post-hoc procedure was applied. This non-parametric test determines whether the differences in average ranks between any pair of algorithms exceed a statistically significant critical threshold and The transformation was performed using Eq.19 and Eq.20 [41].

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \quad (19)$$

$$\left|R_i - R_j\right| > CD \quad (20)$$

Where CD the critical threshold which is determined when exceeded means that the difference between the two algorithms is statistically significant. q_α the critical value of the Studentized Range distribution (q-distribution) at a significance level of α, this value is constant and available in Nemenyi tables, N denoted the number of runs, K denoted the number of algorithms, $\left|R_i - R_j\right|$ the average rank of algorithm i and algorithm j as calculated during the Friedman test.

5.5.3 Wilcoxon Signed-Rank Test

To assess the pairwise significance of performance differences between two algorithms, the Wilcoxon signed-rank test, a non-parametric paired comparison method, was employed. This test evaluates whether the observed differences across matched samples are statistically meaningful without assuming normality the process was performed using Eq.21, Eq.22 and Eq. 23 [42].

$$W = \min(W^+, W^-) \quad (21)$$

Where W is the baseline Wilcoxon statistic, and the test is based on the smallest sum of ranks between

$$W^+ = \sum_{d_i>0} rank_i \quad W^- = \sum_{d_i<0} rank_i \quad (22)$$

Where d_i the difference between the result of algorithm A and B in sample I, $d_i > 0$ the results of algorithm A are better than those of algorithm B, $d_i < 0$ the results of algorithm **A** are better than those of algorithm A, $rank_i$ absolute value order of the difference, W^+ the sum of the ranks of the samples in which A was better than B, W^- the sum of the ranks of the samples in which B was better than A.

$$Z = \frac{W - \frac{N(N+1)}{4}}{\sqrt{\frac{N(N+1)(2N+1)}{24}}} \quad (23)$$

This equation is used when the number of samples N, is large, and to get p-value the value of W must be converted to Z, where N Number of pairs (number of times the two models are compared), $\frac{N(N+1)}{4}$ expected value of W in the absence of

differences, $\sqrt{\frac{N(N+1)(2N+1)}{24}}$ standard deviation of ranks.

6. Results and Discussion

This section outlines the outcomes of the comparative approaches based on the evaluation metrics applied.

6.1 Overall Performance Comparison

As shown in Table 3 illustrate the overall behavior of the compared algorithms across the main evaluation metrics. The proposed DAOA stands out by achieving the highest average accuracy of 0.9499 and the best F1 score of 0.9504, while relying on only eight selected features. This combination reflects a model that is both accurate and lightweight Especially in the Internet of Things with limited resources. In contrast, BAOA requires nearly double the number of features and delivers noticeably lower accuracy. The remaining algorithms, including PSO, GWO, HS, and ACOR, show competitive performance but do not surpass DAOA in the balance between precision, recall, and specificity. The comparison highlights that DAOA offers a more efficient feature-selection behavior with consistently.

Table 3:Summary Overview

NO	Algorithm	Avg Feats	Avg Acc	Avg F1	Avg Prec	Avg Rec	Avg Spec
1	BAOA (Original)	15.8	0.9351	0.9298	0.9274	0.9351	0.9796
2	DAOA (Baseline)	8.0	0.9499	0.9504	0.9519	0.9499	0.9931
3	PSO	17.1	0.9397	0.9359	0.9343	0.9397	0.9837
4	GWO	17.1	0.9423	0.9400	0.9393	0.9423	0.9865
5	HS	16.1	0.9480	0.9476	0.9481	0.9480	0.9915
6	ACOR	12.6	0.9483	0.9484	0.9498	0.9483	0.9921

The best run for each algorithm was selected based solely on the highest classification accuracy achieved across the 30 independent executions. The confusion matrix for every algorithm was then generated using the feature subset obtained in this best-performing run. To ensure a fair and consistent comparison, the same k-nearest neighbors (KNN) classifier was applied to all algorithms under identical training and testing conditions. These figure 3 confusion matrices illustrate the best-run performance of all compared algorithms and provide a detailed view of how each method classifies individual attack categories. Across the six matrices, the DAOA model shows the most balanced and accurate predictions, particularly in the dominant classes where misclassification is minimized and the diagonal cells record higher counts than those of the competing approaches. BAOA, PSO, GWO, HS, and ACOR also achieve strong results, yet each displays noticeable confusion in certain classes, with some algorithms misclassifying more samples in the minority categories. The improvement with DAOA is visible not only in the reduction of false positives but also in its ability to correctly identify smaller or harder-to-detect classes, which often represent subtle intrusion patterns. This consistency across categories highlights DAOA's enhanced robustness, better feature selection, and stronger generalization capability compared to the other algorithms.

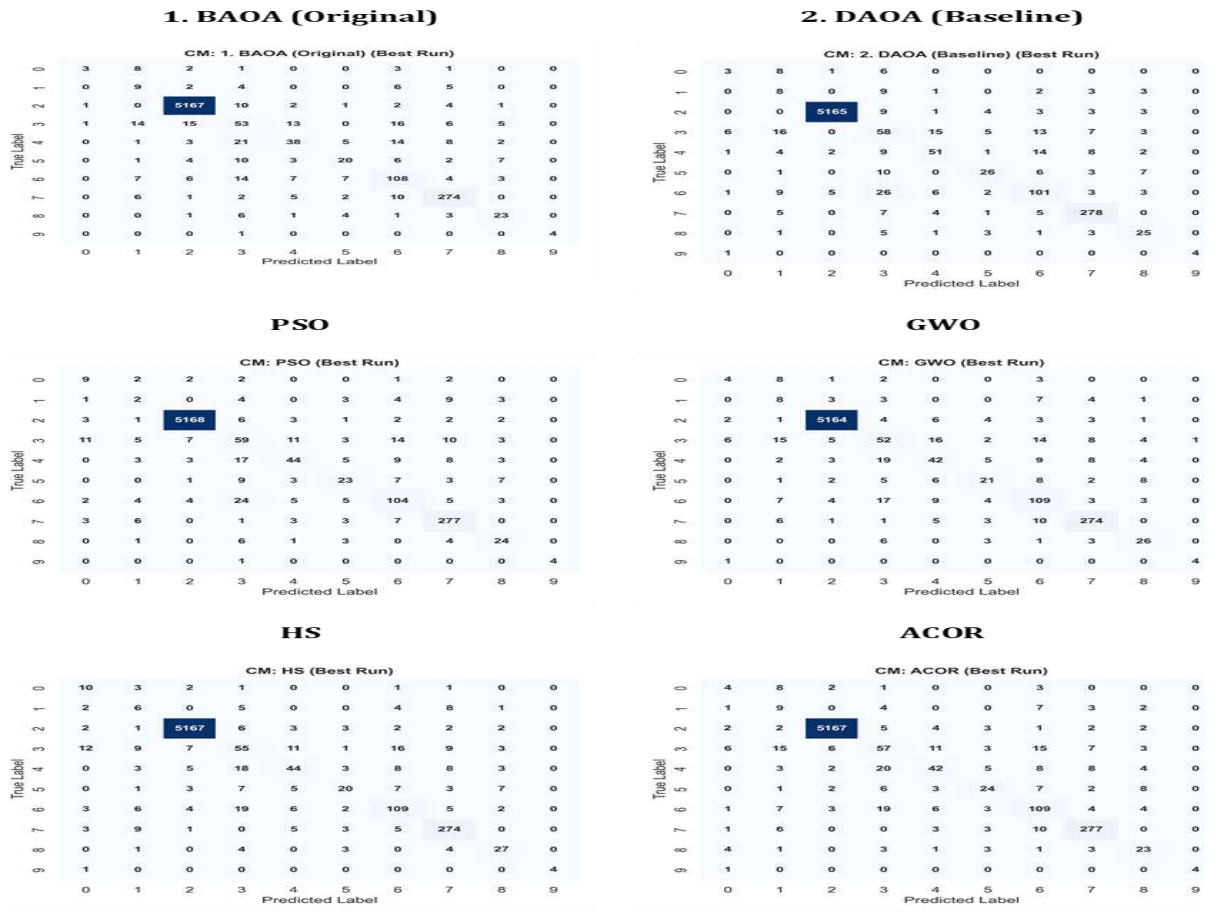

Figure 3:Best-Run Confusion Matrix Analysis for BAOA, DAOA, PSO, GWO, HS, and ACOR

Figure 4 presents compare the average performance metrics achieved by all algorithms, including accuracy, precision, recall, and specificity. DAOA demonstrates the strongest overall balance, reaching the highest accuracy and recall while also maintaining superior precision compared to the original BAOA and swarm-based methods. HS and ACOR achieve competitive results, particularly in specificity where both methods perform near the top. In contrast, BAOA, PSO, and GWO show noticeable drops in precision and slightly lower accuracy levels, indicating less consistent detection capability. The comparison highlights DAOA's advantage in achieving both high predictive performance and strong reliability across multiple evaluation criteria and it is worth noting that accuracy and recall are of almost the same degree.

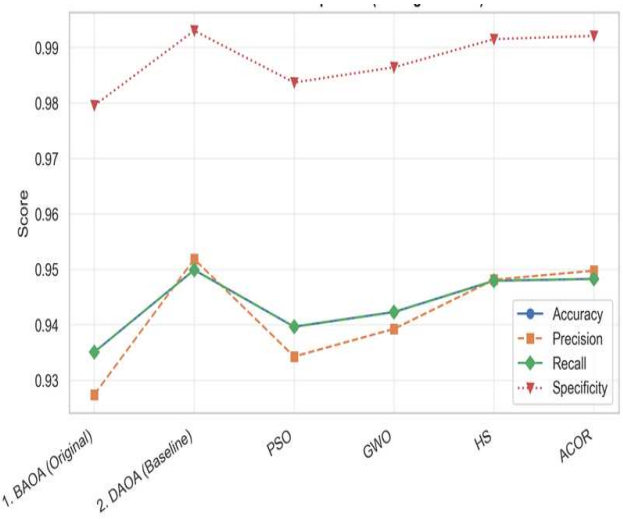

Figure 4:Average Performance Metrics Comparison Across All Algorithms

6.2 Stability Across 30 Independent Runs

To ensure that the observed performance is not the result of random fluctuations, the compared algorithms were executed over 30 independent runs and 20 epoch, each starting from a different initial population. Evaluating stability across multiple runs provides a deeper understanding of an algorithm's reliability and robustness, particularly for stochastic optimization methods that may exhibit inconsistent behavior. The results reveal clear differences in stability among the algorithms, highlighting which methods produce consistent solutions and which suffer from performance variability.

This table 4 displays the performance of the BAOA algorithm across 30 independent runs, providing a clear insight into the algorithm's stability, Reliability and behavioral variability between runs. We observe the number of selected features oscillating between 12 and 22. This fluctuation underscores the algorithm's sensitivity to initial conditions, preventing it from converging on a consistent, definitive subset of features, Accuracy drifts between 0.92 and 0.95 instead of staying fixed. This shifting behavior clearly shows that the results aren't steady, as the performance changes every time, we run the test and a similar discrepancy is also observed in the values of F1, Precision, and Recall. This indicates that the algorithm does not always maintain a consistent balance between correct detection and error reduction. In terms of time, execution often ranges between 108 and 121 seconds, which means that the computational cost of the algorithm is average but almost constant compared to the volatility of performance. This shows us that while BAOA gives decent results overall, it struggles to stay consistent each time we repeat the test.

Table 4:BAOA Algorithm Performance Across 30 Independent Runs

Run	Feats	Acc	F1	Prec	Rec	Spec	Time(s)
1	18	0.9412	0.9398	0.9399	0.9412	0.9874	99.37
2	12	0.9372	0.9331	0.9312	0.9372	0.9841	109.21
3	16	0.9273	0.9167	0.9107	0.9273	0.9698	121.80
4	18	0.9310	0.9237	0.9203	0.9310	0.9749	119.31
5	19	0.9342	0.9291	0.9268	0.9342	0.9792	118.80
6	17	0.9313	0.9254	0.9223	0.9313	0.9766	115.91
7	17	0.9233	0.9123	0.9054	0.9233	0.9680	115.63
8	13	0.9392	0.9365	0.9363	0.9392	0.9829	115.28
9	19	0.9420	0.9398	0.9400	0.9420	0.9851	114.57
10	14	0.9325	0.9266	0.9227	0.9325	0.9791	114.83
11	16	0.9273	0.9168	0.9108	0.9273	0.9698	115.33
12	12	0.9385	0.9374	0.9379	0.9385	0.9864	112.10
13	15	0.9340	0.9282	0.9257	0.9340	0.9788	109.34
14	14	0.9222	0.9125	0.9067	0.9222	0.9685	108.10
15	14	0.9453	0.9443	0.9436	0.9453	0.9902	109.04
16	13	0.9283	0.9187	0.9147	0.9283	0.9720	109.84
17	15	0.9398	0.9385	0.9391	0.9398	0.9874	109.19
18	15	0.9470	0.9468	0.9481	0.9470	0.9902	109.61
19	16	0.9313	0.9255	0.9223	0.9313	0.9770	109.70
20	14	0.9467	0.9450	0.9451	0.9467	0.9881	108.94
21	19	0.9417	0.9393	0.9384	0.9417	0.9833	109.45
22	16	0.9312	0.9230	0.9192	0.9312	0.9747	109.26
23	18	0.9498	0.9490	0.9496	0.9498	0.9913	110.30
24	17	0.9460	0.9454	0.9454	0.9460	0.9897	109.85
25	14	0.9237	0.9146	0.9100	0.9237	0.9697	109.44
26	16	0.9262	0.9159	0.9101	0.9262	0.9705	109.54
27	17	0.9267	0.9177	0.9132	0.9267	0.9715	109.79
28	14	0.9493	0.9494	0.9509	0.9493	0.9922	109.45
29	14	0.9202	0.9092	0.9028	0.9202	0.9658	109.65
30	22	0.9385	0.9348	0.9336	0.9385	0.9829	110.51

Table 5 summarizes tracks DAOA's performance over 30 separate runs, revealing a high stability that clearly stands out against the others. The algorithm keeps the results focused, selecting only between 5 and 13 features. This narrow range reflects the algorithm's ability to maintain a small, effective set of features across most runs. Accuracy remains steady in a high range between 0.9473 and 0.9532. This consistency shows that the algorithm isn't bothered by random starts, allowing it to reach high-quality solutions time after time. The same pattern appeared in the F1, Precision, Recall and Specificity metrics, where the values tended to be homogeneous and convergent, indicating a good balance between error reduction and correct detection. Even though it runs a little slower than BAOA, the time stays steady between 138 and 150 seconds. This extra time is a fair, considering the high quality of the results and the small, focused set of features it finds. This consistent performance reveals that DAOA not only achieves higher average performance, but also maintains remarkable consistency across all runs, making it a practical choice for intrusion detection models that require stable accuracy and low computational load.

Table 5:DAOA Algorithm Performance Across 30 Independent Runs

Run	Feats	Acc	F1	Prec	Rec	Spec	Time(s)
1	10	0.9487	0.9493	0.9507	0.9487	0.9930	146.03
2	5	0.9498	0.9489	0.9489	0.9498	0.9933	144.51
3	5	0.9487	0.9502	0.9531	0.9487	0.9936	149.35
4	8	0.9498	0.9506	0.9524	0.9498	0.9933	148.04
5	6	0.9490	0.9503	0.9526	0.9490	0.9932	143.77
6	8	0.9498	0.9498	0.9502	0.9498	0.9919	148.87
7	12	0.9513	0.9516	0.9530	0.9513	0.9929	144.55
8	6	0.9473	0.9475	0.9487	0.9473	0.9894	144.42
9	13	0.9532	0.9544	0.9563	0.9532	0.9944	149.82
10	10	0.9507	0.9514	0.9528	0.9507	0.9938	142.96
11	11	0.9492	0.9503	0.9525	0.9492	0.9934	144.58
12	7	0.9492	0.9483	0.9487	0.9492	0.9904	141.33
13	7	0.9478	0.9480	0.9487	0.9478	0.9931	143.74
14	6	0.9500	0.9508	0.9524	0.9500	0.9935	146.88
15	6	0.9512	0.9520	0.9536	0.9512	0.9943	141.82
16	11	0.9518	0.9514	0.9518	0.9518	0.9940	146.26
17	9	0.9492	0.9495	0.9502	0.9492	0.9929	138.59
18	9	0.9480	0.9490	0.9510	0.9480	0.9926	146.20
19	7	0.9505	0.9512	0.9526	0.9505	0.9925	146.08
20	8	0.9483	0.9487	0.9498	0.9483	0.9923	145.00
21	6	0.9505	0.9495	0.9501	0.9505	0.9935	143.94
22	5	0.9500	0.9499	0.9507	0.9500	0.9934	143.76
23	10	0.9493	0.9501	0.9520	0.9493	0.9932	150.09
24	9	0.9503	0.9512	0.9529	0.9503	0.9939	149.16
25	8	0.9503	0.9517	0.9543	0.9503	0.9936	147.60
26	7	0.9507	0.9517	0.9536	0.9507	0.9927	145.71
27	9	0.9518	0.9531	0.9551	0.9518	0.9939	146.97
28	7	0.9487	0.9491	0.9503	0.9487	0.9929	146.34
29	6	0.9508	0.9519	0.9533	0.9508	0.9936	140.76
30	8	0.9513	0.9519	0.9532	0.9513	0.9933	139.20

Table 6 presents the performance of the PSO algorithm across 30 independent runs, revealing a noticeably unstable behavior compared to the other algorithms. The number of selected features ranges from 13 to 24, a wide span that

reflects the lack of consistency in the feature-selection process and the algorithm's inability to converge toward a stable subset of attributes. Accuracy also fluctuates considerably, moving between lower values around 0.9225 and higher values approaching 0.9523, indicating that PSO is highly sensitive to initialization and the randomness inherent in its search mechanism. The same pattern appears in the F1, Precision, Recall, and Specificity metrics, where the values rise and fall irregularly, reducing the reliability of the algorithm when runs are repeated. Meanwhile, the computational time remains within a moderate and relatively stable range of 105 to 120 seconds, but this stability in runtime does not translate into stable performance. Overall, PSO is capable of producing strong results in some runs, yet it fails to maintain this level consistently across all iterations. This variability suggests that the algorithm can occasionally reach high-quality solutions, but it would require additional enhancements to reduce oscillations and improve stability, both in feature selection and in classification performance.

Table 6:PSO Algorithm Performance Across 30 Independent Runs

Run	Feats	Acc	F1	Prec	Rec	Spec	Time(s)
1	21	0.9235	0.9127	0.9059	0.9235	0.9683	105.75
2	19	0.9477	0.9469	0.9463	0.9477	0.9907	111.30
3	20	0.9493	0.9486	0.9484	0.9493	0.9914	110.29
4	15	0.9517	0.9506	0.9503	0.9517	0.9929	118.58
5	15	0.9225	0.9128	0.9071	0.9225	0.9686	117.43
6	15	0.9430	0.9431	0.9440	0.9430	0.9910	119.26
7	21	0.9488	0.9482	0.9483	0.9488	0.9920	107.79
8	19	0.9523	0.9520	0.9524	0.9523	0.9933	109.69
9	17	0.9480	0.9482	0.9493	0.9480	0.9916	107.18
10	20	0.9298	0.9208	0.9163	0.9298	0.9726	107.42
11	15	0.9265	0.9164	0.9103	0.9265	0.9704	116.75
12	18	0.9470	0.9459	0.9455	0.9470	0.9903	112.25
13	21	0.9435	0.9433	0.9447	0.9435	0.9899	108.32
14	17	0.9485	0.9477	0.9485	0.9485	0.9909	113.37
15	19	0.9272	0.9182	0.9134	0.9272	0.9715	107.77
16	15	0.9465	0.9466	0.9476	0.9465	0.9908	113.83
17	15	0.9380	0.9332	0.9308	0.9380	0.9813	119.06
18	16	0.9508	0.9516	0.9533	0.9508	0.9931	112.99
19	24	0.9268	0.9171	0.9118	0.9268	0.9709	108.58
20	16	0.9230	0.9129	0.9076	0.9230	0.9684	109.60
21	15	0.9438	0.9418	0.9412	0.9438	0.9882	111.55
22	17	0.9308	0.9227	0.9191	0.9308	0.9745	115.92
23	16	0.9492	0.9493	0.9503	0.9492	0.9926	117.60
24	18	0.9377	0.9354	0.9362	0.9377	0.9847	114.86
25	14	0.9492	0.9490	0.9496	0.9492	0.9916	107.73
26	13	0.9473	0.9465	0.9465	0.9473	0.9921	120.01
27	13	0.9433	0.9416	0.9422	0.9433	0.9898	127.32
28	19	0.9313	0.9231	0.9181	0.9313	0.9752	111.73
29	15	0.9373	0.9340	0.9331	0.9373	0.9818	115.89
30	16	0.9257	0.9158	0.9109	0.9257	0.9708	113.07

Table 7 summarizes the performance of the GWO algorithm across 30 independent runs, illustrating a pattern of noticeable fluctuation in both feature selection and predictive accuracy. The number of selected features varies between 11 and 22, a range that reflects the algorithm's inconsistent search trajectory and its difficulty in stabilizing around a compact and reliable feature subset. Accuracy values also shift substantially, moving from lower readings near 0.9287 to higher ones exceeding 0.9507. This irregular spread indicates that GWO is sensitive to initialization and may converge to different local optima across runs. A similar level of instability appears in the F1, Precision, Recall, and Specificity metrics, where the scores rise and fall without a consistent trend. Although several runs achieve strong results, others show a noticeable drop, which reduces the algorithm's overall

reliability when assessed repeatedly. Computational time, on the other hand, ranges widely-from around 70 seconds to more than 210 seconds-revealing that even runtime is not stable and can double or triple depending on the search path.

Table 7:GWO Algorithm Performance Across 30 Independent Runs

Run	Feats	Acc	F1	Prec	Rec	Spec	Time(s)
1	16	0.9272	0.9174	0.9125	0.9272	0.9701	87.97
2	14	0.9483	0.9478	0.9480	0.9483	0.9916	202.00
3	16	0.9500	0.9508	0.9525	0.9500	0.9931	102.96
4	14	0.9422	0.9407	0.9407	0.9422	0.9873	150.95
5	17	0.9450	0.9434	0.9428	0.9450	0.9883	92.49
6	15	0.9460	0.9460	0.9465	0.9460	0.9912	190.95
7	18	0.9465	0.9453	0.9451	0.9465	0.9882	98.19
8	17	0.9407	0.9397	0.9395	0.9407	0.9896	83.74
9	14	0.9488	0.9490	0.9498	0.9488	0.9923	185.43
10	21	0.9483	0.9478	0.9476	0.9483	0.9906	70.98
11	11	0.9490	0.9490	0.9498	0.9490	0.9928	210.10
12	22	0.9320	0.9279	0.9273	0.9320	0.9818	89.84
13	19	0.9298	0.9207	0.9153	0.9298	0.9727	73.31
14	16	0.9498	0.9502	0.9517	0.9498	0.9927	96.47
15	19	0.9507	0.9508	0.9515	0.9507	0.9930	85.60
16	18	0.9477	0.9477	0.9489	0.9477	0.9915	85.16
17	18	0.9348	0.9286	0.9250	0.9348	0.9777	105.87
18	12	0.9493	0.9503	0.9515	0.9493	0.9935	739.90
19	19	0.9468	0.9465	0.9468	0.9468	0.9900	75.14
20	19	0.9262	0.9166	0.9110	0.9262	0.9707	88.08
21	16	0.9443	0.9442	0.9449	0.9443	0.9916	121.10
22	20	0.9240	0.9155	0.9105	0.9240	0.9711	85.59
23	18	0.9463	0.9453	0.9451	0.9463	0.9892	110.34
24	17	0.9473	0.9464	0.9461	0.9473	0.9904	73.34
25	16	0.9462	0.9461	0.9467	0.9462	0.9909	129.97
26	15	0.9477	0.9476	0.9484	0.9477	0.9905	147.12
27	20	0.9363	0.9332	0.9330	0.9363	0.9837	79.46
28	20	0.9287	0.9207	0.9152	0.9287	0.9748	74.49
29	21	0.9497	0.9495	0.9501	0.9497	0.9913	85.46
30	15	0.9402	0.9365	0.9349	0.9402	0.9818	105.49

As shown in Table 8 the performance of the HS algorithm across 30 independent runs and reveals a relatively stable but occasionally inconsistent behavior. The number of selected features fluctuates between 12 and 21, a moderate range that indicates partial stability in feature selection, yet not enough to ensure a consistently compact subset. Accuracy values generally lie within a high band, from about 0.9432 to 0.9527, suggesting that HS is capable of maintaining strong predictive performance across most runs. However, some runs show noticeable drops, reflecting sensitivity to initialization and search-path variations. The F1, Precision, Recall, and Specificity metrics follow a similar pattern: strong values in the majority of runs but with intermittent fluctuations that signal uneven convergence. One striking observation is the large spike in computational time in run 12, which exceeds 2600 seconds, deviating sharply from the usual range of 80 to 130 seconds. This anomaly indicates that HS can sometimes fall into prolonged search cycles, significantly increasing the computational cost. that HS shows delivers good classification performance and relatively consistent metric values, but occasional instability both in feature-selection variability and rare but extreme runtime spikes limits its reliability when compared to more stable algorithms such as DAOA.

Table 8:HS Algorithm Performance Across 30 Independent Runs

Run	Feats	Acc	F1	Prec	Rec	Spec	Time(s)
1	18	0.9380	0.9366	0.9367	0.9380	0.9871	88.24
2	18	0.9472	0.9475	0.9483	0.9472	0.9913	79.57
3	14	0.9527	0.9526	0.9536	0.9527	0.9928	121.51
4	14	0.9287	0.9195	0.9151	0.9287	0.9718	114.77
5	15	0.9492	0.9496	0.9509	0.9492	0.9926	121.19
6	20	0.9505	0.9499	0.9500	0.9505	0.9927	84.91
7	19	0.9432	0.9425	0.9427	0.9432	0.9896	87.23
8	18	0.9472	0.9461	0.9457	0.9472	0.9910	85.03
9	10	0.9507	0.9501	0.9504	0.9507	0.9927	124.40
10	16	0.9490	0.9495	0.9509	0.9490	0.9930	94.83
11	15	0.9498	0.9498	0.9503	0.9498	0.9928	101.62
12	15	0.9483	0.9490	0.9508	0.9483	0.9928	2611.52
13	13	0.9503	0.9513	0.9533	0.9503	0.9933	108.05
14	19	0.9515	0.9511	0.9517	0.9515	0.9932	80.53
15	17	0.9488	0.9492	0.9503	0.9488	0.9928	93.92
16	12	0.9507	0.9511	0.9523	0.9507	0.9937	116.44
17	16	0.9492	0.9491	0.9502	0.9492	0.9919	85.12
18	17	0.9473	0.9473	0.9482	0.9473	0.9912	124.93
19	21	0.9490	0.9500	0.9519	0.9490	0.9929	92.50
20	16	0.9480	0.9480	0.9487	0.9480	0.9930	90.98
21	17	0.9495	0.9489	0.9490	0.9495	0.9923	91.94
22	14	0.9507	0.9498	0.9497	0.9507	0.9930	108.04
23	17	0.9498	0.9493	0.9495	0.9498	0.9930	82.00
24	21	0.9485	0.9477	0.9476	0.9485	0.9916	84.88
25	16	0.9488	0.9480	0.9480	0.9488	0.9917	88.28
26	17	0.9492	0.9490	0.9498	0.9492	0.9924	108.10
27	14	0.9437	0.9432	0.9434	0.9437	0.9916	105.25
28	12	0.9478	0.9479	0.9489	0.9478	0.9916	142.10
29	14	0.9512	0.9521	0.9542	0.9512	0.9939	93.96
30	18	0.9503	0.9508	0.9523	0.9503	0.9926	80.28

Table 9 reports the performance of the ACOR algorithm across 30 independent runs and highlights a mix of strong predictive accuracy alongside substantial instability in computational time. The number of selected features remains relatively stable, fluctuating between 8 and 17 features, which indicates reasonable consistency in feature-selection behavior. Accuracy values are generally high, ranging from about 0.9438 to 0.9523, and remain tightly grouped. The F1, Precision, Recall, and Specificity metrics follow the same pattern, revealing that ACOR is capable of producing solid classification performance across most runs. And the computational time shows extreme variability. While some runs finish in around 175 to 365 seconds, others spike dramatically, exceeding 450 seconds in several runs and reaching over 9500 seconds in run 1. These drastic jumps suggest that ACOR is prone to falling into prolonged or inefficient search cycles, which significantly increase the cost of execution. This inconsistency in runtime stands in sharp contrast to the steadier timing observed in other algorithms. From the above و ACOR offers strong accuracy and stable metric values, but its unpredictable and sometimes exceptionally high computational time severely limits its practicality. Even though the algorithm can reach high-quality solutions, the risk of excessive runtime makes it less suitable for real-world intrusion detection scenarios that require efficient and consistent performance.

Table 9:ACOR Algorithm Performance Across 30 Independent Runs

Run	Feats	Acc	F1	Prec	Rec	Spec	Time(s)
1	13	0.9523	0.9527	0.9537	0.9523	0.9933	9596.03
2	14	0.9503	0.9508	0.9525	0.9503	0.9928	241.27
3	12	0.9522	0.9521	0.9534	0.9522	0.9933	203.47
4	13	0.9517	0.9524	0.9537	0.9517	0.9936	453.90
5	10	0.9507	0.9499	0.9494	0.9507	0.9925	365.06
6	13	0.9500	0.9509	0.9525	0.9500	0.9932	384.84
7	13	0.9442	0.9443	0.9454	0.9442	0.9916	361.67

8	11	0.9527	0.9533	0.9546	0.9527	0.9936	229.31
9	11	0.9512	0.9519	0.9536	0.9512	0.9931	547.09
10	15	0.9295	0.9198	0.9144	0.9295	0.9720	485.75
11	11	0.9513	0.9520	0.9533	0.9513	0.9936	751.78
12	8	0.9438	0.9438	0.9445	0.9438	0.9912	263.98
13	11	0.9480	0.9478	0.9482	0.9480	0.9903	290.45
14	13	0.9518	0.9526	0.9542	0.9518	0.9933	254.48
15	15	0.9510	0.9521	0.9538	0.9510	0.9933	317.44
16	12	0.9508	0.9514	0.9531	0.9508	0.9934	175.71
17	15	0.9518	0.9525	0.9538	0.9518	0.9931	249.93
18	13	0.9165	0.9218	0.9359	0.9165	0.9885	349.03
19	11	0.9498	0.9509	0.9530	0.9498	0.9930	264.67
20	13	0.9510	0.9508	0.9513	0.9510	0.9933	212.62
21	8	0.9493	0.9501	0.9519	0.9493	0.9931	310.38
22	13	0.9515	0.9517	0.9527	0.9515	0.9930	261.19
23	15	0.9515	0.9522	0.9539	0.9515	0.9930	235.74
24	16	0.9502	0.9506	0.9522	0.9502	0.9933	168.56
25	17	0.9492	0.9491	0.9500	0.9492	0.9926	192.13
26	11	0.9500	0.9508	0.9525	0.9500	0.9931	272.81
27	8	0.9492	0.9502	0.9517	0.9492	0.9939	261.38
28	16	0.9513	0.9519	0.9532	0.9513	0.9932	205.51
29	14	0.9462	0.9404	0.9376	0.9462	0.9934	235.21
30	**13**	0.9500	0.9510	0.9535	0.9500	0.9932	261.40

Figure 5 illustrates boxplot illustrates the distribution of classification accuracy over 30 independent runs for each algorithm. DAOA shows the most stable and consistently high performance, with a narrow accuracy range and a median close to the upper bound, indicating strong robustness across runs. ACOR and HS follow closely, exhibiting relatively tight distributions but with occasional outliers reflecting less stable behavior. In contrast, BAOA, PSO, and GWO display wider variability, with BAOA showing the largest spread and the lowest median accuracy. The comparison highlights DAOA's superior reliability, as it maintains high accuracy with significantly lower variance than the other algorithms.

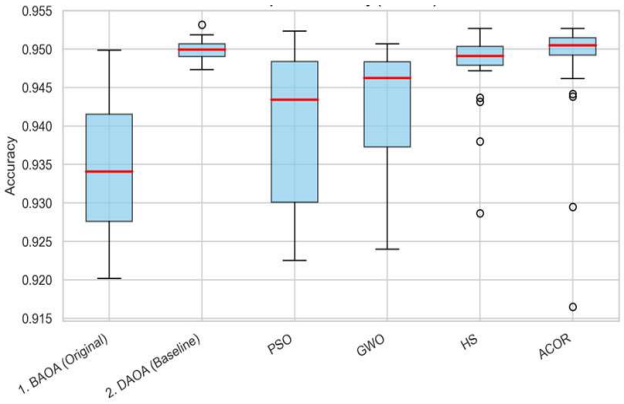

Figure 5:Accuracy Distribution Across 30 Independent Runs for All Algorithms.

6.3 Feature-Reduction Efficiency

Figure 6 illustrates the average number of selected features for all competing algorithms and highlights clear differences in their feature-reduction capabilities. DAOA achieves the most compact subset, selecting only eight features on average, which reflects a more efficient and focused search strategy. BAOA, HS, and ACOR select moderate numbers of features, while PSO and GWO consistently choose the largest subsets, exceeding sixteen features. The comparison shows that DAOA offers a stronger balance between dimensionality reduction and model efficiency, making it more suitable for intrusion detection system in IoT scenarios where lightweight feature sets are essential.

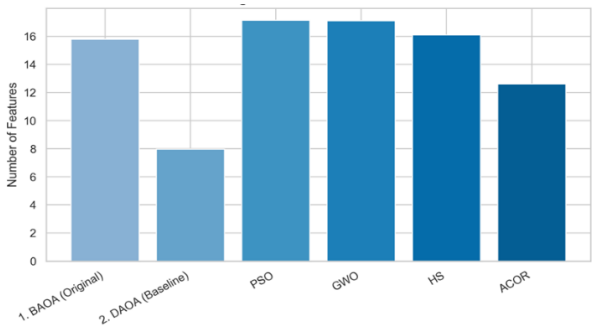

Figure 6:Average Selected Features

Figure 7 depicts bubble chart illustrates the trade-off between accuracy, the number of selected features, and computational time for all evaluated algorithms. DAOA appears at the upper left region of the plot, achieving the highest accuracy while using the fewest features, demonstrating a highly efficient and compact solution. ACOR and HS deliver competitive accuracy levels but require more features and considerably higher execution time, as reflected by their larger bubble sizes. PSO and GWO show moderate performance with higher feature counts and lower accuracy compared to the leading methods. BAOA, positioned lower with a larger feature set and reduced accuracy, reflects the limitations of the original algorithm. Overall, the visualization highlights DAOA's superior balance, offering strong predictive performance with minimal feature usage and reasonable computational cost, making it particularly suitable for IoT environments where efficiency and lightweight operation are essential.

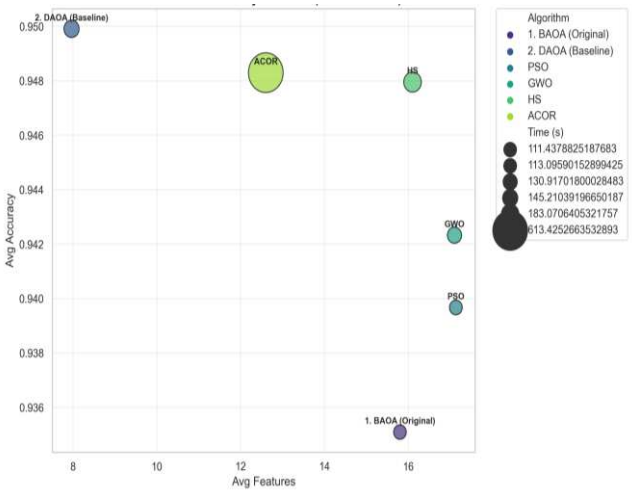

Figure 7:Performance–Efficiency Trade-off for All Algorithms (Bubble Size Represents Time)

6.4 Convergence Analysis

The figure 8 shows is convergence curve summarizes how each algorithm improves its fitness value over the 20 optimization epochs. BAOA shows the slowest and least stable progression, with a wider spread indicating higher variability across runs. In contrast, DAOA converges rapidly during the early iterations and maintains a consistently high fitness level, reflecting both stability and strong search efficiency. PSO, GWO, HS, and ACOR display closer patterns, yet DAOA preserves a slight but steady advantage throughout the optimization process. The narrow confidence band around DAOA further highlights its reliability, demonstrating that the algorithm not only reaches better solutions but does so with reduced variance compared to the other methods

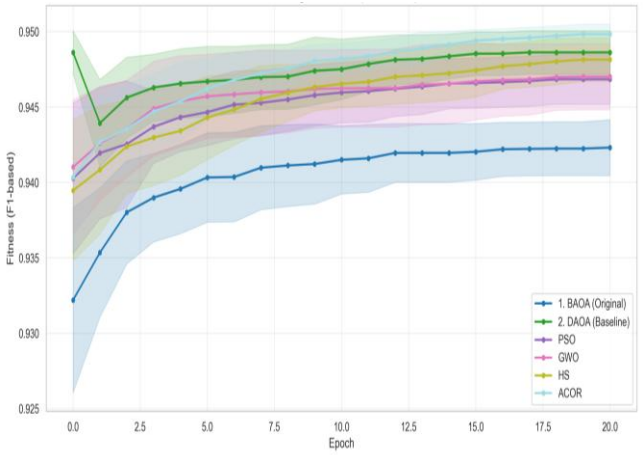

Figure 8:Convergence Curves of All Algorithms Over 20 Epochs (Mean ± Standard Deviation).

6.5 Statistical Significance Findings

Statistical tests were employed to verify whether the observed performance differences among the evaluated algorithms are meaningful rather than due to random variation. This section presents the results of the Friedman ranking test, the pairwise Wilcoxon signed-rank comparisons, and the Nemenyi post-hoc analysis, providing a rigorous assessment of the statistical significance behind DAOA.

6.5.1 Friedman Test Results:

As shown in the table 10, the Friedman test revealed highly significant differences among the evaluated algorithms, with extremely small p-values for both accuracy (5.2848×10^{-14}) and F1-score (3.9535×10^{-16}). These results confirm that the algorithms do not belong to the same statistical group. Based on the mean ranks, DAOA achieved the highest position across all metrics, establishing it as the best-performing method. ACOR ranked second and showed no statistically significant difference from DAOA, as indicated by its p-value of 1.000 for accuracy and 0.9515 for F1-score. In contrast, HS, GWO, PSO, and the original BAOA were all statistically inferior, with p-values below 0.05, demonstrating clear evidence that DAOA outperforms these algorithms. Overall, these findings validate the effectiveness and reliability of the deterministic DAOA approach in achieving superior classification performance for IoT intrusion detection.

Friedman Test Results:

A. Accuracy p-value: 5.2848×10^{-14}
B. F1-score p-value: 3.9535×10^{-16}
C. Best algorithm (by mean accuracy): DAOA (Baseline)

Table 10:Friedman Ranking and Pairwise p-Values for All Algorithms

Rank	Algorithm	Mean Acc	Mean F1	Rank (Acc)	Rank (F1)	p vs Best (Acc)	p vs Best (F1)	Statistically Superior?
1	DAOA	0.9499	0.9504	1	1	-	-	Best
2	ACOR	0.9483	0.9484	2	2	1	0.9515	No
3	HS	0.948	0.9476	3	3	0.0101**	0.0002**	No
4	GWO	0.9423	0.94	4	4	<0.0001**	<0.0001**	No
5	PSO	0.9397	0.9359	5	5	<0.0001**	<0.0001**	No
6	BAOA	0.9351	0.9298	6	6	<0.0001**	<0.0001**	No

6.5.2 Pairwise Wilcoxon Signed-Rank Test Results:

As showing in table 11 using the Wilcoxon signed-rank test for pairwise comparison with the best-performing algorithm (DAOA), the results show that BAOA, PSO, GWO, and HS all achieved very small p-values for both accuracy and F1-score ($p < 0.05$), confirming that DAOA is statistically superior to these methods. In contrast, ACOR recorded p-values of 1.0000 for accuracy and 0.9515 for F1-score, indicating no statistically significant difference between its performance and that of DAOA. Despite this similarity, DAOA maintains a decisive practical advantage: it consistently selected the smallest number of features across all experiments while still achieving the highest accuracy. This balance of efficiency and predictive strength positions DAOA as the most effective and reliable algorithm among all methods evaluated.

Table 11:Pairwise Wilcoxon Signed-Rank Test Against DAOA (Accuracy and F1-Score)

Algorithm	Compared to	p-value for accuracy	p-value for F1-scor
BAOA (Original)	DAOA (Baseline)	9.31E-09	3.73E-09
PSO	DAOA (Baseline)	2.76E-06	3.15E-07
GWO	DAOA (Baseline)	1.02E-07	3.54E-08
HS	DAOA (Baseline)	1.01E-02	2.32E-04
ACOR	DAOA (Baseline)	1.00E+00	9.52E-01

6.5.3 Namanya post-hoc test Results:

The Namanya post-hoc test compares every pair of algorithms to determine whether their convergence performance differs significantly after running the optimization for multiple epochs. The table 12 reports p-values for each pair, where values below 0.05 (marked with "*") indicate a statistically significant difference between the two algorithms. The results show that DAOA (Baseline) demonstrates significant performance differences against most competing algorithms, with p-values of 0.0000* when compared to BAOA, PSO, and GWO. This confirms that DAOA achieved superior and more stable convergence behavior. Its comparison with HS yields a p-value of 0.5841, indicating no statistically significant difference, while its comparison with ACOR ($p = 1.0000$) shows complete similarity in convergence patterns. Other algorithms demonstrate mixed statistical relationships. For example, HS and ACOR show no significant difference ($p = 0.6967$), while HS differs significantly from PSO and GWO. Overall, the table highlights that DAOA consistently forms one of the top-performing groups, with only ACOR showing statistically equivalent convergence speed and stability, further reinforcing DAOA's reliability relative to the other methods.

Table 12:Nemenyi Post-hoc Test (P-Values Matrix)

Algo vs Algo	BAOA	DAOA	PSO	GWO	HS	ACOR
BAOA	1.0000	0.0000*	0.8935	0.6299	0.0001*	0.0000*
DAOA	0.0000*	1.0000	0.0000*	0.0001*	0.5841	1.0000
PSO	0.8935	0.0000*	1.0000	0.9968	0.0074*	0.0000*
GWO	0.6299	0.0001*	0.9968	1.0000	0.0356*	0.0001*
HS	0.0001*	0.5841	0.0074*	0.0356*	1.0000	0.6967
ACOR	0.0000*	1.0000	0.0000*	0.0001*	0.6967	1.0000

Figure 9 provides an overview of the Critical Difference (CD) diagram visualizes the Nemenyi post-hoc test results based on the average ranks of the evaluated algorithms. The DAOA (Baseline) algorithm achieves the best overall rank and forms a statistically non-significant group with ACOR, indicating that both methods deliver comparable accuracy but, DAOA selects substantially fewer features than ACOR, giving it a clear practical advantage in efficiency and making it the more suitable choice for resource-constrained IoT intrusion detection environments. In contrast, the remaining algorithms HS, GWO, PSO, and BAOA are positioned beyond the CD threshold, showing that their performance is statistically inferior to DAOA. The diagram clearly highlights that DAOA outperforms all competing methods.

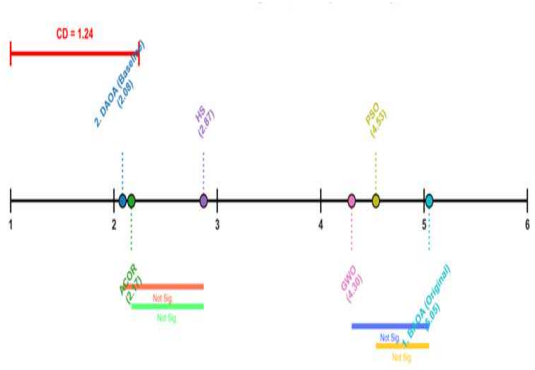

Figure 9:Critical difference diagram (Nemenyi, Accuracy).

7. Conclusion and Future Work

In this study, A deterministic version of the Arithmetic Optimization Algorithm (DAOA) was proposed to solve the feature-selection problem in classification for IoT intrusion detection. The method was evaluated against five widely used binary metaheuristics BAOA, BPSO, GWO, BHS, and ACOR using the NF-UNSW-NB15-V2 dataset. Across 30 independent runs, DAOA consistently achieved the highest classification accuracy while selecting the smallest number of features, demonstrating a strong balance between predictive performance and feature reduction efficiency. Statistical analyses using the Friedman, Wilcoxon, and Nemenyi tests confirmed that DAOA significantly outperforms BAOA, PSO, GWO, and HS and ACOR. Despite this, DAOA's ability to reach this level of accuracy with a substantially reduced feature subset offers a clear practical advantage for resource-constrained IoT environments. These findings show that DAOA effectively manages the exploration, exploitation trade-off and provides a robust and efficient wrapper-based feature-selection framework for IoT intrusion detection systems. in Future work it would be worthwhile to applying DAOA to additional real-world optimization problems, investigating alternative transfer functions, or integrating DAOA with other classifiers such as SVM and neural networks to assess whether its performance advantages generalize beyond KNN-based classification.

Data availability Data are available from the corresponding author on reasonable request

Funding Open Access NO funding provided.

Declarations

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

Authors contributions: Taha M.O. Alakhras performed the comparative analysis and prepared the manuscript. Waheed A.H.M. Ghanem, Farizah Yunus, Sanaa A.A. Ghaleb, and Mohammed Otair provided critical review, validation, and valuable insights for revision. All authors read and approved the final manuscript.

Ethical approval: This material is the authors' own original work, which has not been previously published elsewhere. The paper is not currently being considered for publication elsewhere. The paper reflects the authors' own research and analysis in a truthful and complete manner.

References

[1] Taha M.O. Alakhras, "A SURVEY OF INTRUSION DETECTION SYSTEMS IN IOT: MACHINE LEARNING AND FEATURE SELECTION APPROACHES," *ijam*, vol. 38, no. 10s, pp. 828–884, Nov. 2025, doi: 10.12732/ijam.v38i10s.1003.

[2] N. Khodadadi *et al.*, "BAOA: Binary Arithmetic Optimization Algorithm With K-Nearest Neighbor Classifier for Feature Selection," *IEEE Access*, vol. 11, pp. 94094–94115, 2023, doi: 10.1109/ACCESS.2023.3310429.

[3] A. A. Alhussan, D. S. Khafaga, E.-S. M. El-Kenawy, A. Ibrahim, M. M. Eid, and A. A. Abdelhamid, "Pothole and Plain Road Classification Using Adaptive Mutation Dipper Throated Optimization and Transfer Learning for Self Driving Cars," *IEEE Access*, vol. 10, pp. 84188–84211, 2022, doi: 10.1109/ACCESS.2022.3196660.

[4] A. A. Abdelhamid *et al.*, "Waterwheel Plant Algorithm: A Novel Metaheuristic Optimization Method," *Processes*, vol. 11, no. 5, p. 1502, May 2023, doi: 10.3390/pr11051502.

[5] T. Rahkar Farshi, "Battle royale optimization algorithm," *Neural Comput & Applic*, vol. 33, no. 4, pp. 1139–1157, Feb. 2021, doi: 10.1007/s00521-020-05004-4.

[6] W. A. H. M. Ghanem, A. Jantan, S. A. A. Ghaleb, and A. B. Nasser, "An Efficient Intrusion Detection Model Based on Hybridization of Artificial Bee Colony and Dragonfly Algorithms for Training Multilayer Perceptrons," *IEEE Access*, vol. 8, pp. 130452–130475, 2020, doi: 10.1109/ACCESS.2020.3009533.

[7] H.-N. Dai, H. Wang, G. Xu, J. Wan, and M. Imran, "Big data analytics for manufacturing internet of things: opportunities, challenges and enabling technologies," *Enterprise Information Systems*, vol. 14, no. 9–10, pp. 1279–1303, Nov. 2020, doi: 10.1080/17517575.2019.1633689.

[8] W. A. H. M. Ghanem *et al.*, "Metaheuristic Based IDS Using Multi-objective Wrapper Feature Selection and Neural Network Classification," in *Advances in Cyber Security*, vol. 1347, M. Anbar, N. Abdullah, and S. Manickam, Eds., in Communications in Computer and Information Science, vol. 1347. , Singapore: Springer Singapore, 2021, pp. 384–401. doi: 10.1007/978-981-33-6835-4_26.

[9] M. Braik, A. Sheta, and H. Al-Hiary, "A novel meta-heuristic search algorithm for solving optimization problems: capuchin search algorithm," *Neural Comput & Applic*, vol. 33, no. 7, pp. 2515–2547, Apr. 2021, doi: 10.1007/s00521-020-05145-6.

[10] Y. Zhang and L. Xing, "A New Hybrid Improved Arithmetic Optimization Algorithm for Solving Global and Engineering Optimization Problems," *Mathematics*, vol. 12, no. 20, p. 3221, Oct. 2024, doi: 10.3390/math12203221.

[11] R. Ranjan and J. K. Chhabra, "A Modified Binary Arithmetic Optimization Algorithm for Feature Selection," *WSEAS TRANSACTIONS ON COMPUTER RESEARCH*, vol. 11, pp. 199–205, July 2023, doi: 10.37394/232018.2023.11.18.

[12] A. Zakeri and A. Hokmabadi, "Efficient feature selection method using real-valued grasshopper optimization algorithm," *Expert Systems with Applications*, vol. 119, pp. 61–72, Apr. 2019, doi: 10.1016/j.eswa.2018.10.021.

[13] F. Seghir, A. Drif, S. Selmani, and H. Cherifi, "Wrapper-Based Feature Selection for Medical Diagnosis: The BTLBO-KNN Algorithm," *IEEE Access*, vol. 11, pp. 61368–61389, 2023, doi: 10.1109/ACCESS.2023.3287484.

[14] S. Zhou, L. Xing, X. Zheng, N. Du, L. Wang, and Q. Zhang, "A Self-Adaptive Differential Evolution Algorithm for Scheduling a Single Batch-Processing Machine With Arbitrary Job Sizes and Release Times," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1430–1442, Mar. 2021, doi: 10.1109/TCYB.2019.2939219.

[15] J.-H. Yi *et al.*, "Behavior of crossover operators in NSGA-III for large-scale optimization problems," *Information Sciences*, vol. 509, pp. 470–487, Jan. 2020, doi: 10.1016/j.ins.2018.10.005.

[16] E. Pashaei and E. Pashaei, "Hybrid binary arithmetic optimization algorithm with simulated annealing for feature selection in high-dimensional biomedical data," *J Supercomput*, vol. 78, no. 13, pp. 15598–15637, Sept. 2022, doi: 10.1007/s11227-022-04507-2.

[17] M. Zivkovic, C. Stoean, A. Petrovic, N. Bacanin, I. Strumberger, and T. Zivkovic, "A Novel Method for COVID-19 Pandemic Information Fake News Detection Based on the Arithmetic Optimization Algorithm," in *2021 23rd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, Romania: IEEE, Dec. 2021, pp. 259–266. doi: 10.1109/SYNASC54541.2021.00051.

[18] Q. Al-Tashi, H. Md Rais, S. J. Abdulkadir, S. Mirjalili, and H. Alhussian, "A Review of Grey Wolf Optimizer-Based Feature Selection Methods for Classification," in *Evolutionary Machine Learning Techniques*, S. Mirjalili, H. Faris, and I. Aljarah, Eds., in Algorithms for Intelligent Systems. , Singapore: Springer Singapore, 2020, pp. 273–286. doi: 10.1007/978-981-32-9990-0_13.

[19] Q. Al-Tashi *et al.*, “Enhanced Multi-Objective Grey Wolf Optimizer with L関y Flight and Mutation Operators for Feature Selection,” *Computer Systems Science and Engineering*, vol. 47, no. 2, pp. 1937–1966, 2023, doi: 10.32604/csse.2023.039788.

[20] Q.-T. Yang, X.-X. Xu, Z.-H. Zhan, J. Zhong, S. Kwong, and J. Zhang, “Evolutionary Multitask Optimization for Multiform Feature Selection in Classification,” *IEEE Trans. Cybern.*, vol. 55, no. 4, pp. 1673–1686, Apr. 2025, doi: 10.1109/TCYB.2025.3535722.

[21] M. Eid. Marwa and R. M. Zaki, “Classification of Student Performance Based on Ensemble Optimized Using Dipper Throated Optimization,” *JAIM*, vol. 2, no. 1, pp. 36–45, 2022, doi: 10.54216/JAIM.020104.

[22] H. Liu, G. Hu, X. Wang, A. G. Hussien, and L. Zhang, “Enhanced Particle Swarm Optimization Algorithm Based on SVM Classifier for Feature Selection,” *CMES*, vol. 142, no. 3, pp. 2791–2839, 2025, doi: 10.32604/cmes.2025.058473.

[23] S. S. Vasan, S. Bhaskar, and N. N, “Enhanced Chaotic Dragonfly Optimization for Early Alzheimer’s Diagnosis: A Feature Selection Technique,” in *2025 3rd International Conference on Inventive Computing and Informatics (ICICI)*, Bangalore, India: IEEE, June 2025, pp. 1204–1209. doi: 10.1109/ICICI65870.2025.11069903.

[24] R. Wang, “Performance Improvement of Krill Foraging Optimization Algorithm Based on Shuffled Frog Leaping Algorithm and Meme Grouping,” *IJCAI*, vol. 48, no. 23, Dec. 2024, doi: 10.31449/inf.v48i23.6786.

[25] R. Alwajih *et al.*, “Hybrid binary whale with harris hawks for feature selection,” *Neural Comput & Applic*, vol. 34, no. 21, pp. 19377–19395, Nov. 2022, doi: 10.1007/s00521-022-07522-9.

[26] A. Vakhnin, I. Ryzhikov, H. Niska, and M. Kolehmainen, “A Novel Multi-Objective Hybrid Evolutionary-Based Approach for Tuning Machine Learning Models in Short-Term Power Consumption Forecasting,” *AI*, vol. 5, no. 4, pp. 2461–2496, Nov. 2024, doi: 10.3390/ai5040120.

[27] S. Mou, J. Gan, Y. Yang, Y. Lan, and C. Rao, “An enhanced bat algorithm based intelligent inspired architecture for resilient macroeconomic prediction,” *Sci Rep*, Nov. 2025, doi: 10.1038/s41598-025-28612-3.

[28] M. Premkumar *et al.*, “An enhanced Gradient-based Optimizer for parameter estimation of various solar photovoltaic models,” *Energy Reports*, vol. 8, pp. 15249–15285, Nov. 2022, doi: 10.1016/j.egyr.2022.11.092.

[29] B. R. Bharani *et al.*, “Grey wolf optimization and enhanced stochastic fractal search algorithm for exoplanet detection,” *Eur. Phys. J. Plus*, vol. 138, no. 5, p. 424, May 2023, doi: 10.1140/epjp/s13360-023-04024-y.

[30] R. M. Rizk-Allah and A. E. Hassanien, “A comprehensive survey on the sine–cosine optimization algorithm,” *Artif Intell Rev*, vol. 56, no. 6, pp. 4801–4858, June 2023, doi: 10.1007/s10462-022-10277-3.

[31] S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, “Enhanced multi-verse optimizer for task scheduling in cloud computing environments,” *Expert Systems with Applications*, vol. 168, p. 114230, Apr. 2021, doi: 10.1016/j.eswa.2020.114230.

[32] M. Ghasemi, S. K. Mohammadi, M. Zare, S. Mirjalili, M. Gil, and R. Hemmati, “A new firefly algorithm with improved global exploration and convergence with application to engineering optimization,” *Decision Analytics Journal*, vol. 5, p. 100125, Dec. 2022, doi: 10.1016/j.dajour.2022.100125.

[33] S. K. Sahoo, A. K. Saha, S. Nama, and M. Masdari, “An improved moth flame optimization algorithm based on modified dynamic opposite learning strategy,” *Artif Intell Rev*, vol. 56, no. 4, pp. 2811–2869, Apr. 2023, doi: 10.1007/s10462-022-10218-0.

[34] L. Abualigah, A. Diabat, S. Mirjalili, M. Abd Elaziz, and A. H. Gandomi, “The Arithmetic Optimization Algorithm,” *Computer Methods in Applied Mechanics and Engineering*, vol. 376, p. 113609, Apr. 2021, doi: 10.1016/j.cma.2020.113609.

[35] R. Alshorman, B. H. Abed-alguni, and Y. E. Alqudah, “AOAFS: A Malware Detection System Using an Improved Arithmetic Optimization Algorithm,” *Technologies*, vol. 13, no. 4, p. 145, Apr. 2025, doi: 10.3390/technologies13040145.

[36] M. Sarhan, S. Layeghy, and M. Portmann, “Towards a Standard Feature Set for Network Intrusion Detection System Datasets,” *Mobile Netw Appl*, vol. 27, no. 1, pp. 357–370, Feb. 2022, doi: 10.1007/s11036-021-01843-0.

[37] D. Pessach and E. Shmueli, “Algorithmic Fairness,” in *Machine Learning for Data Science Handbook*, L. Rokach, O. Maimon, and E. Shmueli, Eds., Cham: Springer International Publishing, 2023, pp. 867–886. doi: 10.1007/978-3-031-24628-9_37.

[38] S. García, A. Fernández, J. Luengo, and F. Herrera, “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining:

Experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, May 2010, doi: 10.1016/j.ins.2009.12.010.

[39] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nat Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020, doi: 10.1038/s41592-019-0686-2.

[40] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, Mar. 2011, doi: 10.1016/j.swevo.2011.02.002.

[41] B. Trawiński, M. Smętek, Z. Telec, and T. Lasota, "Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms," *International Journal of Applied Mathematics and Computer Science*, vol. 22, no. 4, pp. 867–881, Dec. 2012, doi: 10.2478/v10006-012-0064-z.

[42] J. Demsˇar and J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research 7 (2006) 1–30*, 2006.