computers

MDPI

Article

Meta-Heuristic Optimization Algorithm-Based Hierarchical Intrusion Detection System

Kamal A. ElDahshan, AbdAllah A. AlHabshy and Bashar I. Hameed *

Mathematics Department, Faculty of Science, Al-Azhar University, Cairo 11884, Egypt
* Correspondence: basharibh78@gmail.com

Abstract: Numerous network cyberattacks have been launched due to inherent weaknesses. Network intrusion detection is a crucial foundation of the cybersecurity field. Intrusion detection systems (IDSs) are a type of machine learning (ML) software proposed for making decisions without explicit programming and with little human intervention. Although ML-based IDS advancements have surpassed earlier methods, they still struggle to identify attack types with high detection rates (DR) and low false alarm rates (FAR). This paper proposes a meta-heuristic optimization algorithm-based hierarchical IDS to identify several types of attack and to secure the computing environment. The proposed approach comprises three stages: The first stage includes data preprocessing, feature selection, and the splitting of the dataset into multiple binary balanced datasets. In the second stage, two novel meta-heuristic optimization algorithms are introduced to optimize the hyperparameters of the extreme learning machine during the construction of multiple binary models to detect different attack types. These are combined in the last stage using an aggregated anomaly detection engine in a hierarchical structure on account of the model's accuracy. We propose a software machine learning IDS that enables multi-class classification. It achieved scores of 98.93, 99.63, 99.19, 99.78, and 0.01, with 0.51 for average accuracy, DR, and FAR in the UNSW-NB15 and CICIDS2017 datasets, respectively.

Keywords: intrusion detection system; machine learning; feature selection; grey wolf optimization; extreme learning machine; Archimedes optimization algorithm; honey badger algorithm

check for updates

Citation: ElDahshan, K.A.; AlHabshy, A.A.; Hameed, B.I. Meta-Heuristic Optimization Algorithm-Based Hierarchical Intrusion Detection System. *Computers* **2022**, *11*, 170. https://doi.org/10.3390/computers11120170

Academic Editor: Leandros Maglaras

Received: 3 October 2022
Accepted: 21 November 2022
Published: 28 November 2022

1. Introduction

Recent statistics from DataReportal state that the number of internet users worldwide has quadrupled over the past ten years, rising from 2.18 billion to 4.95 billion from 2012 to 2022 [1] as a result of the rapid evolution of online users, apps, services, and devices in various domains. IoT technology is a trending area in portable information, and it has great importance in regard to creation, transfer, storage, and deletion sessions. Thus, the confidentiality, integrity, and availability of information security must be considered in order to overcome system vulnerabilities and to prevent intruders from damaging the system by trying to steal, destroy, or alter the information therein [2,3]. The term "intrusion" is used in an information system to describe any activity that undermines a system's security policy. IDS is the process used to discover intrusions [4]. IDS helps to detect, determine, and identify unauthorized system activities [5]. As shown in Figure 1, IDS may be software, hardware, or a combination divided into four types. Host-based IDS checks attacks on its system, in contrast to network-based IDS, which checks the overall network activities. Signature-based IDS monitors any skewness from the normal behavior based on rules that require the continuous updating of the databases, while anomaly-based IDS identifies suspicious threats. Hence, detecting anomalies is a classification (ML) problem [6,7]. Using feature selection methods (wrapper, filter, and embedded) or learning-based methods to reduce misclassification and to minimize the data dimensionality of big data network transactions, choosing the best features is an area of high interest in the ML community [8,9].

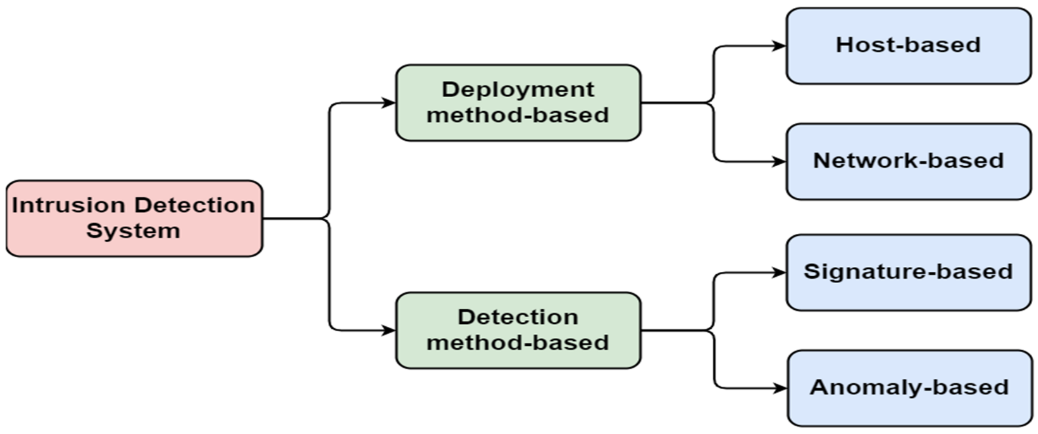


Figure 1. Taxonomy of common IDSs.

There are ML algorithms capable of discovering hidden relationships in data and making judgments without explicit guidance [10,11]. The majority of these techniques are based on a back-propagation learning algorithm. Hence, IDS is a real-time system that requires a high classifier performance speed during the system design planning stage. A rapid and reliable approach to real-time applications for the detection of attacks is necessary for IDS. As the learning algorithm and architecture of the extreme learning machine (ELM) are efficient and straightforward, ELM can detect attacks more rapidly and precisely than conventional identification approaches, such as support vector machines (SVM) [12,13].

Recently, the use of ELMs has increased in real-time classification applications, as they outperform their peers in categorization. ELM can improve computational efficiency by selecting the optimal features and identifying the number of neurons in the hidden layer. The most effective selection methods are meta-heuristic-based, including evolutionary, physics, and swarm algorithms. These methods can select the most ideal data features and hyperparameter values to construct an efficient ML-model-based IDS [12].

Thus, an ML-based detection strategy that works well in terms of accuracy and FAR is required to determine how many correct predictions were made and how effective the classifier was in reducing the number of wrongly classified cases. As such, this paper focuses on constructing a robust software anomaly-based IDS using the techniques discussed from a classification point of view. Additionally, it provides information about performance metrics such as the accuracy, DR, FAR, precision, and F1-score. This paper makes the following main contributions:

- Proposes a new IDS that uses ML rather than deep learning (DL) to create the classification system, which has resulted in more efficient and less complicated models;
- Utilizes ELM for the categorization of attacks—ELM has already been utilized for IDS, but not quite in the manner that is being proposed;
- Calculates the importance of features and chooses the best ones to detect attack types using efficient meta-heuristic optimization algorithms;
- Selects optimal values of ELM parameters using a novel hybrid meta-heuristic-based hyperparameter selection method;
- Utilizes an aggregation of ELM binary models, with one for each attack type, and collects these models via a hierarchical layer to derive an interpretable and highly accurate output.

The rest of the paper is organized as follows: Section 2 briefly discusses some research on ML-based IDSs. Section 3 describes the preliminary concepts of some standard methods for ML classification and meta-heuristic optimization. Section 4 and its subsections describe the detailed stages of the proposed system. The experimental analytic procedures, the UNSW-NB15 and CICIDS2017 dataset specifications, a demonstration of the performance metrics, and a discussion of the results are presented in Section 5. Section 6 concludes the paper.

2. Literature Review

In this section, the state of the art is presented as regards the use of ML anomaly-based IDSs for networks and the key theoretical approaches that use one or more of the important datasets that are available for training the ML models.

Some researchers have presented overviews of various approaches to network intrusion detection that focus on improving the overall performance. These include ELM, SVM, Random Forests (RF), Decision Trees (DT), Deep Neural Networks (DNN), Multilayer Perceptron (MLP), etc. Some researchers also focus on feature selection and hyperparameter optimization, representing two more common areas of study. This section concentrates on the state of the art of multi-class classification using the selected benchmark malware datasets.

Moualla et al. [12] suggested a network IDS based on multi-class ML that is dynamically scalable. Structured based on supervised ML, the approach comprises many steps. The dataset's unbalanced classes are addressed with the synthetic minority oversampling technique (SMOTE), which then extracts relevant features from the dataset for each class using the extra trees classifier. In the next step, the ELM model acts as a binary classifier for each type of attack (i.e., "One-Versus-All") for its identification. Finally, a fully connected layer is used to learn from all the possible permutations of the ELM classifier's outputs. A logistic regression layer is then used to make soft judgments across all classes.

Ren et al. [14] proposed an efficient IDS that combines data sampling and feature selection in a single hybrid data optimization process. To obtain the best training dataset, they utilized the RF classifier as the evaluation criterion, the Genetic Algorithm (GA) to improve the sampling ratio, and the Isolation Forest (iForest) to filter out irrelevant data. Once again, GA and RF were employed for feature selection to determine the best possible subset of features to use. Finally, the best training dataset was collected via data sampling, the best features were chosen through feature selection, and an RF-based IDS was developed.

Gu and Lu. [15] suggested a naive Bayes feature embedding system based on SVM for intrusion detection. In order to create new high-quality data, their system applies the naive Bayes feature transformation approach to the original features. Next, the system trains an SVM classifier on the changed data to create an intrusion detection model. The proposed detection system is effective and resilient in experiments across various intrusion detection datasets.

Faker and Dogdu. [16] integrated big data with deep learning techniques to boost IDS' effectiveness. A Deep Feedforward Neural Network (DNN) and two ensemble techniques—RF and Gradient Boosting Tree (GBT)—were used to classify the network traffic datasets. They used a homogeneity metric to select the datasets' most essential features. The ML models were evaluated using a five-fold cross-validation approach. The proposed method was implemented using Apache Spark, a distributed computing framework, the Keras deep learning library was used for the deep learning application, and the Apache Spark ML library was used for the ensemble techniques.

He et al. [17] proposed a system that extracts different levels' features from the network connection instead of a large feature vector for the sake of more efficient processing. The system employs a multimodal-sequential intrusion detection technique utilizing a hierarchical progressive network with Multimodal Deep Autoencoder (MDAE) and Long Short-Term Memory (LSTM) technology. The system can easily combine information on multiple levels' characteristics inside a network and automatically learn temporal information across nearby connections using a hierarchical progressive network.

Wang et al. [18] developed a model for IDS using an ameliorated Deep Belief Network (DBN). Conventional approaches, such as neural network training methods, begin training a model with pre-prepared parameters, including thresholds and randomly initialized weights, which can cause problems such as the biasing of the model toward suboptimal solutions and the extension of the training time. To improve the model, they employed a Kernel-based Extreme Learning Machine (KELM) escorted by supervised learning in place of the back-propagation (BP) algorithm in DBN. Considering the poor classification performance that might result from randomly initializing kernel parameters using KELM, an improved grey wolf optimizer was developed to tune these parameters.

Vinayakumar et al. [19] developed a versatile and efficient IDS using a DNN to identify and categorize unknown and unanticipated cyberattacks. The proposed model is based on a hybrid intrusion detection warning system within a highly scalable framework deployed on a commodity hardware server to assess the network and host-level activity for the purpose of processing and analyzing large amounts of data in near real time. In-depth comparisons with traditional ML classifiers on many benchmark IDS datasets led to the DNN model's selection. In addition, the suggested DNN model was used to collect real-time features from hosts and networks to spot intrusions and attacks.

Choobdar et al. [20] developed an SDN-based IDS that can be deployed as an application module in the controller. In the suggested system, there are three distinct stages. Initially, the features are pre-trained using sparsely stacked autoencoders (AE), and later, these are learned in an unsupervised way. The system is trained using the Softmax algorithm in the second stage, and in the third stage, the system's parameters are adjusted.

Lee et al. [21] proposed the Generative Adversarial Networks (GAN) model as a solution to the problem of data imbalance by employing an unsupervised DL technique to produce synthetic data that are highly comparable to real data. After fixing data imbalances with a GAN, the proposed model uses RF to improve the detection performance.

Lee and Park [22] developed a DL-based IDS for use in networks when the ratio of malicious to benign traffic is exceptionally high. The system's goal is to address issues of data asymmetry and achieve high-performance intrusion detection using unsupervised learning models (AE and GAN models) during DL. The AE-CGAN (autoencoder–conditional GAN) model was suggested to enhance the intrusion detection efficiency.

Researchers have explored and developed effective IDSs based on ML and DL. The authors' comprehensive evaluations of experiments with ML classifiers were used to analyze several benchmark malware datasets that are available to the public, such as UNSW-NB15 and CICIDS 2017, with three key challenges: (1) Some researchers assessed their systems by establishing a few performance metrics that do not allow for deep analysis, such as FAR, which is one primary types of IDS indicators. (2) They did not state the optimal values of the hyperparameter selection methods. (3) The performance is affected by significant differences in the feature numbers of each dataset. To deal with the challenges outlined above, this paper considers feature selection methods and identifies the best hyperparameter values of ELM when constructing multiple binary models to detect attack types separately. It also evaluates the proposed system using six metrics: accuracy, precision, DR, FAR, specificity, and the F1-score.

3. Preliminary Concepts

Software anomaly-based IDSs use different ML algorithms. Hence, enhancing the designed ML pipeline increases the efficiency of the proposed software anomaly-based IDS. The pipeline includes feature selection methods, the classifier, and hyperparameter optimization. The next subsections provide brief descriptions of the various issues.

3.1. Feature Selection Methods

Feature selection (FS) is approached via three methods: filter, embedded, and wrapper. The filter method identifies and extracts a feature subset via the given evaluation criteria, the wrapper method utilizes the learning algorithms, and the embedded (hybrid) approach combines both. The embedded method selects features based on their weights after training an ML model [23,24].

Table 1 summarizes the characteristics of the three FS methods. Each method is described, with examples of its benefits and drawbacks.

FS is the process of identifying the most important features and discarding or removing irrelevant, redundant, and noisy features to derive a better classification model that requires less learning time and has a lower data dimensionality [25]. In the context of a feature set size of $fsiz$, the FS methods search for an optimal feature subset among the competitive 2^{fsiz} possible subsets. Depending on the problem at hand, the definition of an optimal subset may change [26].

Table 1. The benefits and drawbacks of the filter, embedded, and wrapper methods.

Method	Benefits	Drawbacks	Examples
1. The filter method relies on the dataset's characteristics to select a subset of features without running any classifiers through statistical analysis [27].	• Independence of the classification algorithm. • Lower computational complexity. • Filters are better than embedding methods and wrappers in terms of the time consumed.	• No interaction with the classifier. • It is not apparent how one sets a threshold for selecting the subset features and how noise is filtered out. • The selected feature subset has a poorer classification accuracy than the embedded and wrapper methods.	• Correlation feature selection (CFS). • Information gain. • Consistency-based filter.
2. The embedded method incorporates the FS phase into the classifier training process and is usually tailored to the specific learning machine issue [23].	• Interacting with the classification algorithm. • Faster than the wrapper method. • More accurate than the filter method. • Less prone to overfitting.	• Classification algorithms are dependent on subset selection.	• Recursive Feature Elimination for SVM (RFE-SVM). • Feature Selection • Perceptron (FS-P).
3. The wrapper method is considered one of the most widely used and robust supervised learning algorithms, and classification is employed as a black box to test the efficiency of the best features [28].	• Interacting with the classification algorithm. • Classification algorithms are dependent on subset selection. • Has the best classification accuracy compared to embedded and filter methods.	• More prone to overfitting than embedded and wrapper methods. • Costly in terms of computation.	• Genetic Algorithm (GA). • Particle Swarm Optimization (PSO). • Grey wolf optimization (GWO).

Grey Wolf Optimizer (GWO)

Recently, a large number of meta-heuristic algorithms were created and used for a wide range of optimization issues. The GWO algorithm for mathematical modeling was created by Mirjalili et al. [29] and requires the adjustment of a few parameters, allowing for exploration and exploitation to be accomplished in a straightforward manner while offering favorable convergence [30].

The inspiration for the GWO is the predatory behaviors of grey wolves in the wild. Wolves live in groups with a rigid hierarchy and several participants (5 to 12). In the GWO, the population is separated into alpha (α)—group leader, beta (β)—can take over as leader of the group if the alpha wolf gets sick or dies, and then delta (δ) and omega (φ)—the rest of the group (see Figure 2). Predation involves a series of steps: pursuit, encirclement, harassment, and, finally, attack [31,32]. Each wolf represents a possible solution, while the prey itself is the best option. α, β, and δ are the best solutions. The remaining nominees are marked as φ. Based on the locations of α, β, and δ, the positions of the omegas are updated. A wolf (*wo*) uses Equations (1)–(4) to figure out how far away it is from the three best solutions. It then uses Equation (5) to change its position [32]:

$$\vec{A} = 2\,\vec{a}\cdot\vec{v_1} - \vec{a} \tag{1}$$

$$\vec{C} = 2\,\vec{v_2} \tag{2}$$

$$\vec{D}_\alpha = \left|\vec{C}\cdot\vec{X}_\alpha - \vec{X}_{wo}\right|,\ \vec{D}_\beta = \left|\vec{C}\cdot\vec{X}_\beta - \vec{X}_{wo}\right|,\ \vec{D}_\delta = \left|\vec{C}\cdot\vec{X}_\delta - \vec{X}_{wo}\right| \tag{3}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}\cdot\vec{D}_\alpha,\ \vec{X}_2 = \vec{X}_\beta - \vec{A}\cdot\vec{D}_\beta,\ \vec{X}_3 = \vec{X}_\delta - \vec{A}\cdot\vec{D}_\delta \tag{4}$$

$$\vec{X}_{wo}(nt) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{5}$$

where $\vec{X}_{wo}$, $\vec{X}_\alpha$, $\vec{X}_\beta$, and $\vec{X}_\delta$ are the position of the wolf vectors for wo, α, β, and δ, while nt is the next iteration of the wolf vector. $\vec{A}$ and $\vec{C}$ are the coefficient vectors. $\vec{D}_\alpha$, $\vec{D}_\beta$, and $\vec{D}_\delta$ represent distance vectors between α, β, δ, and wo. $\vec{v_1}$ and $\vec{v_2}$ are vectors derived at random from the range [0,1]. A variable with an arrow over it is a vector quantity. Thus, writing an arrow over a variable is the standard way of writing a vector. "| |" represents the absolute operator, which extracts the magnitude of an actual number independently of its sign, while "·" represents the multiplication operator. $\vec{a}$ is calculated using the equation below:

$$\vec{a} = \frac{2 - 2\cdot t}{t_{max}} \tag{6}$$

where t is the number of current iterations, and t_{max} is the maximum number of iterations.

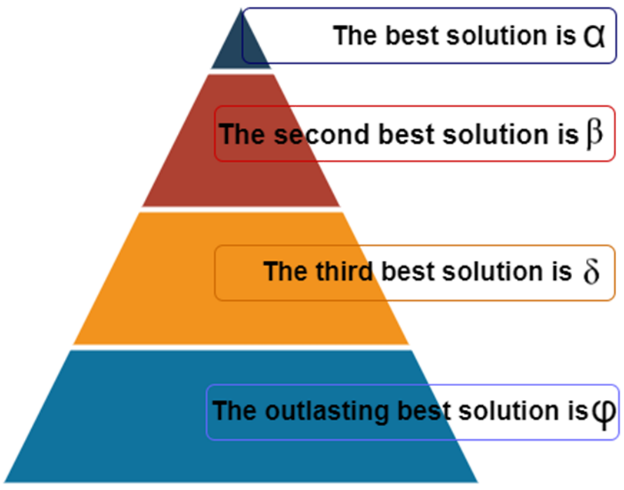

The best solution is α
The second best solution is β
The third best solution is δ
The outlasting best solution is φ

Figure 2. Wolf group composition scheme.

3.2. Hyperparameter Optimization

Hyperparameter optimization in ML involves selecting the appropriate hyperparameters for a learning algorithm. The values of a hyperparameter are utilized to tune the learning process. The hyperparameters' values must be tuned so that the model can optimally handle the ML task. Different algorithms and methods can be used to determine the best hyperparameter values, including meta-heuristic-based search algorithms [33]. Archimedes optimization and honey badger were recently introduced as heuristic-based search algorithms.

3.2.1. Archimedes Optimization Algorithm (AOA)

AOA is a population-based meta-heuristic physics algorithm inspired by Archimedes' rules of buoyancy in water. AOA is an algorithm that uses a population of objects as candidates to achieve a particular aim. AOA starts by determining how to fit the initial population and then begins to repeat the process with a set limit. The search for objects involves the generation of volumes, densities, and accelerations at random. By considering both exploitation and exploration, the AOA is a global optimization method that can be described through a mathematical model with the following steps [34]:

Step 1—Initialization. This involves the initialization of all the objects' locations, the volume (vol), density (den), and acceleration (acc) using the equations below:

$$o_l = lb_l + rand \times (ub_l - lb_l); l = 1, 2, \ldots, POP \tag{7}$$

$$\begin{aligned} den_l &= rand \\ vol_l &= rand \end{aligned} \tag{8}$$

$$acc_l = lb_l + rand \times (ub_l - lb_l) \tag{9}$$

where o_l represents the l^{th} object in a population of POP objects, while the minimum and maximum boundaries of the solution space are denoted by lb_l and ub_l, respectively. $rand$ creates random numbers from the range [0,1] in the G-dimensional vector. In this step, the best population is assessed, and the best-fitting object with the highest fitness value is chosen and assigned o_{best}, den_{best}, vol_{best}, and acc_{best}.

Step 2—Updates volumes and densities. The object's volumes and densities for the next iteration $t + 1$ are updated and can be formulated using the equations below:

$$den_l^{t+1} = den_l^t + rand \times (den_{best} - den_l^t) vol_l^{t+1} = vol_l^t + rand \times (vol_{best} - vol_l^t) \tag{10}$$

where $rand$ refers to a random numeric that is uniformly distributed between [0, 1]. den_{best} and vol_{best} refer to the volume and density of the best object identified up to this point.

Step 3—Transfer operator and density factor (*TF*). This step involves the transfer from exploration to exploitation. A collision between objects is commenced and, after a set time, the objects strive for a state of balance. The TF rises until it reaches a stable value and is defined using the equation below:

$$TF = exp\left(\frac{t - t_{max}}{t_{max}}\right) \tag{11}$$

where t is the number of current iterations, and t_{max} is the maximum number of iterations. TF progressively grows over time until it reaches 1. Another feature that helps AOA in global-to-local searches is the density decreasing factor ddf, which becomes smaller over time and is defined using the equation below:

$$ddf^{t+1} = exp\left(\frac{t_{max} - t}{t_{max}}\right) - \left(\frac{t}{t_{max}}\right) \tag{12}$$

Through iteration, the value of ddf^{t+1} decreases over time until it reaches a previously identified target region. AOA enables a good balance between exploration and exploitation if this variable is regulated well.

Step 4—Update the object's acceleration and normalization. The object's acceleration for the iteration $t + 1$ update process acc_l^{t+1} is split into three phases. These are called the "exploration phase," "Exploitation phase," and the "normalized acceleration phase". The following offers a more detailed explanation.

Step 4.1—Exploration phase (a collision between objects occurs). If $TF \le 0.5$, which indicates a collision between objects, we choose the random material (mr) and update the object's acceleration for iteration $t+1$ using the equation below:

$$acc_l^{t+1} = \frac{den_{mr} + vol_{mr} \times acc_{mr}}{den_l^{t+1} \times vol_l^{t+1}} \tag{13}$$

where $deni_l$, vol_l, and acc_l represent the density, volume, and acceleration of the object l, respectively. The density, volume, and acceleration of the random material are represented by, den_{mr}, vol_{mr}, and acc_{mr}, respectively. In particular, it is essential to note that $TF \le 0.5$ enables exploration in one-third of the iteration time. The exploration–exploitation efficiency is altered when a value other than 0.5 is used.

Step 4.2—Exploitation phase (no collision between objects occurs). In the absence of collisions (i.e., TF > 0.5), the following equation should be used to update the object's acceleration at the next iteration, $t+1$:

$$acc_l^{t+1} = \frac{den_{best} + vol_{best} \times acc_{best}}{den_l^{t+1} \times vol_l^{t+1}} \tag{14}$$

where acc_{best} represents the object's acceleration and is the best possible value.

Step 4.3—Normalize acceleration. Here, the acceleration is normalized to compute the alteration percentage using the equation below:

$$acc_{l-norm}^{t+1} = up \times \frac{acc_l^{t+1} - \min(acc)}{\max(acc) - \min(acc)} + lo \tag{15}$$

where the range of normalization is defined as up = 0.9 and lo = 0.1, respectively. acc_{l-norm}^{t+1} specifies the proportion of the step that each object alters. The acceleration values are high if the object l is far from the global optimum, indicating that it is in the exploration phase. Conversely, it will be in the exploitation phase, i.e., demonstrating the advancement of the search from its exploration phase to its exploitation phase. The AOA is successful in achieving stability with regard to exploitation and exploration.

Step 5—Update position. During the exploration phase (i.e., $TF \le 0.5$), the position must be updated, whereby the position of object l^{th} is modified for the next iteration $t+1$ using the equation below:

$$b_l^{t+1} = b_l^t + Con_1 \times rand \times acc_{l-norm}^{t+1} \times ddf \times \left(b_{rand} - b_l^t\right) \tag{16}$$

where b_l^{t+1} represents the l^{th} object's updated position in a population of POP objects. Con_1 is a constant that always equals 2. Contrarily, however, during the exploitation phase (i.e., TF > 0.5), the objects update their locations using the equation below:

$$b_l^{t+1} = b_{best}^t + F \times Con_2 \times rand \times acc_{l-norm}^{t+1} \times ddf \times \left(TR \times b_{best} - b_l^t\right) \tag{17}$$

where Con_2 is a constant equal to 6. TR rises as time goes on and is directly proportional to the transfer operator, which can be represented as $TR = Con_3 \times TF$. The range of TR is $[Con_3 \times 0.3,\ 1]$, and Con_3 is a constant equal to 2. We begin with small proportions, since this causes the step size of the random walk to be significant and the difference between the best and the present positions to be considerable. During the search, this proportion gradually increases to diminish the disparity between the best and the actual positions, achieving a balance between exploration and exploitation. F presents the direction of the object as a flag that is arranged according to the object positions, which are defined using the equation below:

$$F = \begin{cases} +1 \; if \; p \le 0.5 \\ -1 \; if \; p > 0.5 \end{cases} \tag{18}$$

where $p = 2 \times rand - Con_4$, and Con_4 is a constant equal to 0.5.

Step 6—Evaluation. Through iteration, AOA uses the objective function and identifies the best possible solution for each object as the evaluation while allocating the best values to each of the following variables: b_{best}, vol_{best}, den_{best}, and acc_{best}.

3.2.2. Honey Badger Algorithm (HBA)

The honey badger is a huge weasel-like animal with black and white fluffy hair that is found in the semi-deserts of Africa, Southwest Asia, and the Indian subcontinent. The HBA is a honey badger swarm-based algorithm modeled based on the clever hunting behavior of honey badgers, who either smell and dig or follow the honeyguide bird to find food. The HBA process may be broken down into two steps: the "digging phase" and the "honey phase", respectively. Thus, in the first phase, referred to as the digging mode, it utilizes its sense of smell to identify the prey's approximate position. Once it reaches the prey, it changes its position to find the best location for digging and capturing it. The honey badger uses the honeyguide bird as a direct guide to the beehive in a later phase, referred to as the honey mode. The following shows a mathematical explanation of these processes [35].

Step 1—Initialization phase. We set the starting population size (PS) and locations of the honey badgers using the equation below:

$$z_k = lb_k + r_1 \times (upb_k - lob_k);\ k = 1, 2, \ldots, PS \tag{19}$$

where z_k represents the k^{th} honey badger in a population of PS objects. In this context, upb_k and lob_k represent the lower and upper boundaries of the search space. r_1 is a random number in the range [0,1].

Step 2—Defining intensity (I). Calculating the intensity based on the prey's attention force and the badger's proximity to the prey is the purpose of this step, k^{th}. This equation states that if the prey's scent intensity (I_k) is high, it will move rapidly. If the scent intensity (I_k) is low, it will move slowly, based on the inverse square law, as stated by the following equation:

$$I_k = r_2 \times \frac{S}{4\pi d_k^2},\ S = (z_k + z_{k+1})^2,\ dis_k = z_{prey} - z_k \tag{20}$$

where S is the attention force, dis_k represents the distance that separates the prey from the k^{th} honey badger, and z_{prey} is the prey's position. r_2 is also a random value in the range of [0, 1].

Step 3—Update density factor (∂). This step controls the randomization of time-based changes to facilitate a smooth transition from exploration to exploitation. Here, we update the density factor (∂), which is reduced with each iteration, to render randomization less probable over time using the equation below:

$$\partial = cons \times exp\left(\frac{-t}{t_{max}}\right) \tag{21}$$

where t stands for the current iteration number, $cons$ is a constant number equal to 2, and t_{max} represents the maximum number of iterations.

Step 4—Escaping and averting local optimum. This step, along with the two steps that follow it, is used to break out of the optimal local positions. In this scenario, the HBA uses a flag called E that modifies the search direction to provide increased possibilities for agents/candidates to thoroughly scan the search space.

Step 5—Updating the positions of agents/candidates. The HBA position update process (z_{new}) is split into two phases, which were previously mentioned. These phases are called the "digging phase" and the "honey phase". The following offers a more detailed explanation.

5.1—Digging phase. Honey badgers utilize their keen sense of smell to locate their prey and then dig around it using their cardioid shape. The HBA update position procedure (z_{new}) is carried out throughout the digging phase via the equation below:

$$z_{new} = z_{prey} + E \times \gamma \times I \times z_{prey} + E \times r_3 \times \partial \times dis_k \times |\cos(2\pi r_4) \times [1 - \cos(2\pi r_5)]| \quad (22)$$

where z_{prey} represents the best estimate of the prey's location, I represents the intensity mentioned in step 2, and the update density factor is denoted by ∂, as mentioned in step 3. The honey badger's food-gathering capacity is given by the variable $\gamma \geq 1$ (which is set to 6 by default). dis_k is the distance from the honey badger to its prey. r_3, r_4, and r_5 denote three random numbers in the range of [0,1], respectively, "| |" represents the absolute operator. E is a flag used to modify the search direction, which is represented by the equation below:

$$E = \begin{cases} +1 \; if \; r_6 \leq 0.5 \\ -1 \; if \; r_6 > 0.5 \end{cases} \quad (23)$$

where r_6 is also a random value in the range $[0,1]$. The smell intensity I of z_{prey} is represented, and dis_k is the distance from the honey badger to its prey, and the search influence factor is ∂, which changes over time. Both are critical factors in the honey badger's digging phase. In addition, any disruption E may be caused by the prey during the digging activities, enabling the badger to choose an even better location for its approach.

5.2—Honey phase. Sometimes a honey badger follows a honeyguide bird to access a beehive. This can be represented via the following equation:

$$z_{new} = z_{prey} + E \times r_7 \times \partial \times dis_k \quad (24)$$

where z_{new} indicates the honey badger's new position, and z_{prey} indicates the prey's position. r_7 is also a random value in the range $[0,1]$. Equations (21) and (23) can be used to compute ∂ and E, respectively. In Equation (24), it can be observed that the honey badger searches near the prey's position z_{prey}, as it has already been identified, depending on the information obtained about the distance dis_k. The time-varying search behavior ∂ has an effect on the search at this point. An E disturbance may also be discovered by a honey badger.

3.3. Extreme Learning Machine (ELM) Classifier

The ELM was developed by Huang et al. [36]. It is a feedforward network with a single hidden layer and three components: input neurons, hidden neurons, and output neurons [37]. The ELM has recently caught the attention of an increasing number of researchers. It can overcome several challenges in ways that other techniques cannot due to its strong generalization capacity, low reliance on manual intervention, and the assurance of a specific level of learning accuracy. It saves a great deal of time and cost when compared to conventional neural networks [38,39]. Figure 3 depicts the model diagram for a single-hidden-layer feedforward network ELM model.

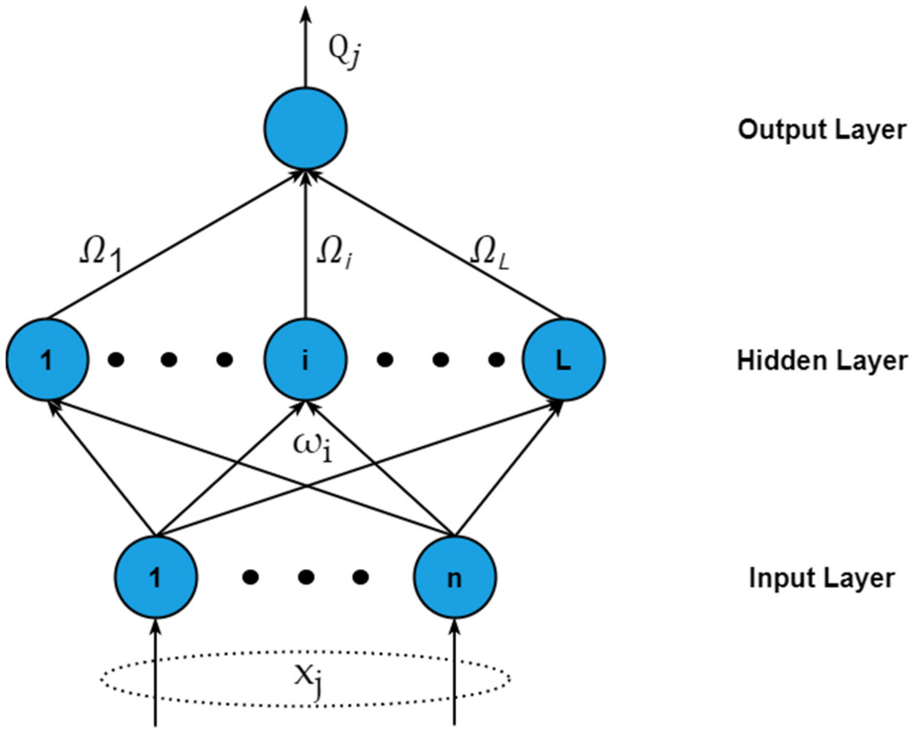


Figure 3. The architecture of the ELM, a single-hidden-layer feedforward network [39].

For any given classification problem, there are N unique samples (x_i, y_i), where $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in R^n$ is the sample input vector and $y_i = [y_{i1}, y_{i2}, \ldots, y_{im}]^T \in R^m$ is the output vector, while n is the number of features of the training sample, and m is the total number of training sample classes. L indicates the nodes used in the hidden layer of the ELM. The output of the ELM can be formulated using the following equations:

$$\sum_{i=1}^{L} \Omega_i g(\omega_i . x_j + u_i) = Q_j,\ j = 1, 2, \ldots, N \tag{25}$$

where $g(\bullet)$ is the activation function, and $\omega_i = [\omega_{i1}, \omega_{i2}, \ldots, \omega_{in}]^T$ indicates the input weight vector that connects the input layer nodes to the hidden layer nodes. $\Omega_i = [\Omega_{i1}, \Omega_{i2}, \ldots, \Omega_{in}]^T$ indicates the output weight vector that acts as the connecting link between the hidden layer nodes and output layer nodes. u_i indicates the offset value of the first node in the hidden layer. $\omega_i . x_j$ indicates the weight assigned to the inner product of the value and the training sample value. Q_j is the network model's actual output. The objective of the single-hidden-layer feedforward network is to minimize the output result's error value using the equation below:

$$\sum_{j=1}^{N} \| Q_j - y_j \| = 0 \tag{26}$$

where y_j is the predicted output, and $\| \ \|$ is a bounded operator between normal spaces. According to Equations (25) and (26), Ω_i, ω_i, and u_i exist to render the following equation true:

$$\sum_{i=1}^{L} \Omega_i g(\omega_i . x_j + u_i) = y_j,\ j = 1, 2, \ldots, N \tag{27}$$

Based on the matrix, the following simplification of Equation (27) can be made:

$$H\Omega = T \tag{28}$$

where H stands for the output value matrix of the hidden layer, Ω stands for the output weight matrix that extends from the hidden layer to the output layer, and T represents the predicted output matrix. In addition to this, $H,\ T$, and Ω are expressed via the following:

$$H = \begin{bmatrix} g(\omega_1.x_1 + u_1) & \cdots & g(\omega_L.x_1 + u_L) \\ \vdots & \ddots & \vdots \\ g(\omega_1.x_N + u_1) & \cdots & g(\omega_L.x_N + u_L) \end{bmatrix}_{N\times L} \tag{29}$$

$$\Omega = \begin{bmatrix} \Omega_1^T \\ \vdots \\ \Omega_L^T \end{bmatrix}_{L\times m} \tag{30}$$

$$T = \begin{bmatrix} y_1^T \\ \vdots \\ y_L^T \end{bmatrix}_{N\times m} \tag{31}$$

The equation $H\Omega = T$ cannot be demonstrated in the great majority of cases. Several criteria are determined to train a model, including Ω_i, ω_i, and u_i. The following equation demonstrates the importance of adjusting these criteria in order to achieve the minimum possible error:

$$\|H(\omega_i, u_1)\Omega_i - T\| = \min_{\omega_i,\Omega_i,u_i} \|H(\omega_i, u_i)\Omega_i - T\| \tag{32}$$

The conventional neural network algorithm would spend more time training the model due to its continuous optimization of the parameters during the iterative process of dealing with such issues. The input weights and the bias of the hidden layer are randomly initialized in the model training of the ELM. The output weight matrix is also provided. Since H is determined, the model becomes a linear system, $H\Omega = T$, which can be fixed with least-squares. Ω can be formulated using the equation:

$$\Omega = H^{+}T \tag{33}$$

In Equation (33), the symbol H^{+} represents the Moore–Penrose generalized inverse matrix of the weight matrix of H's output.

ELM is effective as a single-hidden-layer feedforward network with a high ability to generalize, as little manual intervention as possible, and a guarantee of the learning accuracy. Moreover, the algorithm's speed is significantly increased. It saves costs and a great deal of time. This occurs during training, as the algorithm does not require the repeated optimization of the input weights and hidden layers from the input layer to the hidden layer. There is a neuron offset, but the input weight and hidden layer offsets are obtained through random initialization. During training, we are only required to deduce how many neurons are in the hidden layer. We can directly obtain the model's output weight from this point and finish the training. The following Algorithm 1 is a summary of the learning process that is carried out by the ELM.

Algorithm 1: Standard ELM Procedure

```
Input: Activation function $g(\bullet)$
#Neurons of hidden layer $L$
N training samples $(x_i, y_i),\ x_i \in R^n,\ y_i \in R^m,\ i \in 1, 2, \ldots,\ N$.
Output: The output weight $\Omega$ from the hidden layer to the output layer.
    Begin
1.      Initialize randomly the input weights $\omega_i$ and the offset of the hidden layer $u_i$,
2.      Calculatethe output weight of the hidden layer $H$ and
3.      Calculate the output weight from hidden layer to the output layer $\Omega$.
    End
```

Our research focuses on constructing an ML-based network IDS. As IDS is a real-time application, ELM is strongly recommended, since it involves simple implementation, has an excellent generalizability, and requires less time spent on training, without iterative adjustment.

4. Proposed Methodology for the IDS Development

4.1. Proposed Development Pipeline

In binary classification, two classes correspond to the two decision elements. In multi-class approaches, there are many FN choice elements, with one for each of the FN classes. Learning a function that maps the set of input features to two decision elements is more straightforward than learning a function that maps the input features to FN decision elements. As a result, multi-class classification is more complicated than binary classification, especially when dealing with imbalanced big data.

After reviewing the previously identified challenges in the research, we determined that the ELM algorithm performs well in change classification, whether multi-class or binary. The learning process of the ELM technique is also affected by the selection of the optimum features. Additionally, the number of neurons in the hidden layer is a challenge in the hyperparameter tuning of the ELM algorithm.

The proposed multi-class classification issue was broken down into a collection of binary classifications to reduce the complexity burden on the classifiers in the aggregation stage. Figure 4 shows the major components of the proposed meta-heuristic optimization algorithms based on the hierarchical IDS. The figure is to scale, can be used to reliably detect the ever-changing data streams in future networks, and contains three principal stages. First, the proposed system starts with a set of essential phases that must be performed before the training process. More specifically, network traffic packets are the data stream captured and logged in the raw dataset before preprocessing, feature reduction, and subsampling to generate binary sub-datasets, guaranteeing the data quality before sending them to the IDS. Secondly, the training stage splits the binary sub-datasets into training and validation test sets, trains the ELM model using the training set, optimizes the hyperparameters using the validation set, and evaluates the binary ELM model using the test set. Finally, for the intrusion detection task, there is an aggregated anomaly detection engine that uses a hierarchical structure.

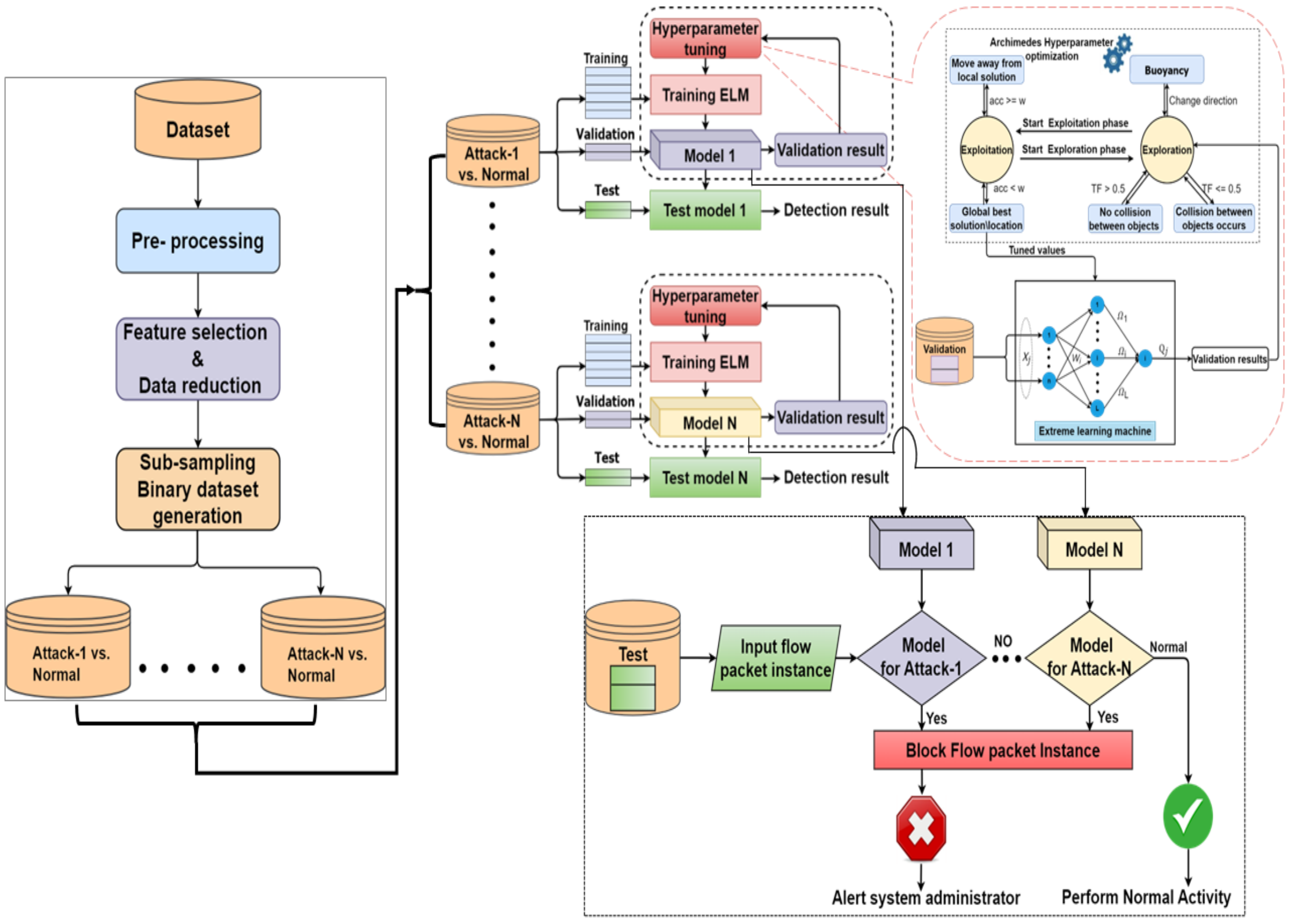


Figure 4. Context flowchart of the proposed system.

4.2. Essential Stage: Network Traffic and Data Preparation

The ML pipeline is commenced at this point, and the raw datasets obtained through a tcpdump or similar network packet collection technique are loaded and handled. There are differences in the dataset, especially when examining instances with a large amount of data and a high level of imbalance in the classes. Data preparation guarantees the model's quality in terms of its construction and prediction by imputing missing values, removing data redundancy, and excluding data outliers. To facilitate the ML modeling process, we proposed an FS method to discard irrelevant, redundant, or noisy features without affecting the data quality and performance of the classification model. Figure 5 represents different phases of the essential data stage for the proposed scheme.

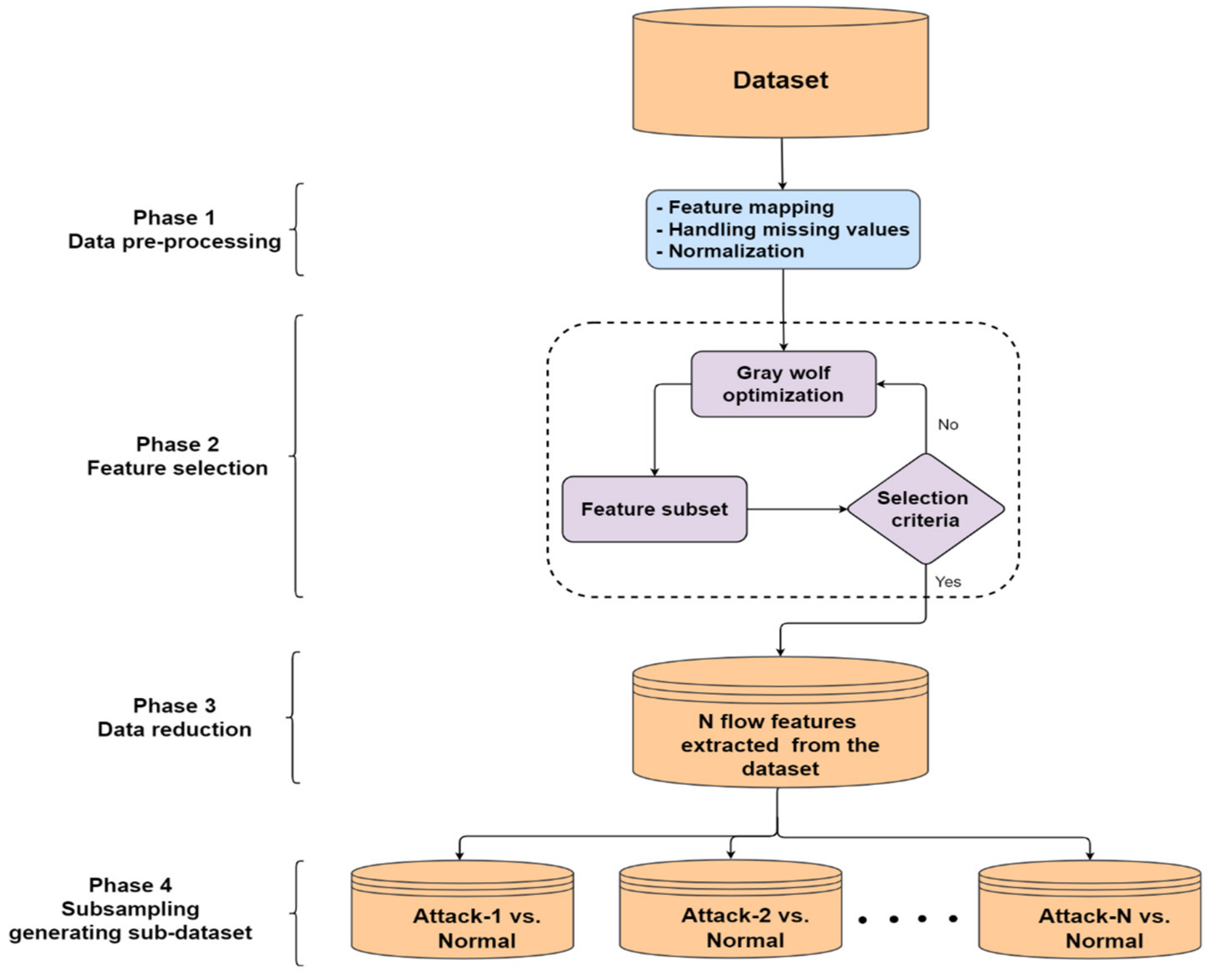


Figure 5. First stage of the essential data preparation procedures.

4.2.1. Data Preprocessing Phase

The proposed system accepts and loads the raw dataset and handles it through feature mapping, missing value imputation, and normalization to derive the final tuned dataset. When dealing with categorical features, a mapping strategy such as label-encoding (LE) is used to encode them numerically. The LE method assigns a numerical value to each classified feature. Since network systems occasionally lose packets while transferring or exchanging faulty data values, the missing values must be addressed as follows. The missing values of the numeric features are replaced with their mean values. Additionally, the missing values of the categorical features are replaced with unknown values. Feature transformation is supported by normalization. The "StandardScaler" normalization method eliminates any bias in the features and preserves their statistical characteristics as unaltered.

4.2.2. Feature Selection and Data Reduction Phase

The amount of digital data available around the world is continually increasing, because data are being collected for various reasons. As a result, ML algorithms can deal with the growth in both the volume and complexity of the data.

Binary GWO Feature Subset Selection (BGWO)

BGWO is a binary optimization selection method, wherein each feature in a dataset is represented as a binary cell in the solution vector. A value of 1 indicates that a feature is relevant and important, whereas a value of 0 indicates that a feature is not selected and is irrelevant to the objective function (see Figure 6).

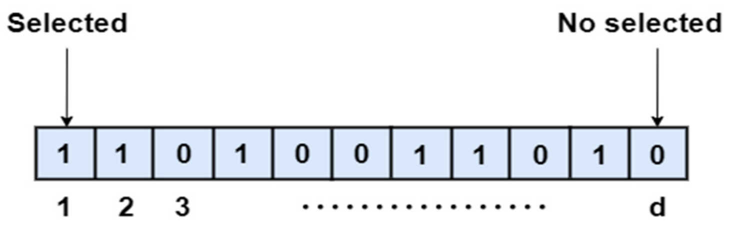


Figure 6. Binary representation of a solution.

All positions in the GWO may be found everywhere in a continuous space. This allows for the updating of the equations to be implemented simply. In BGWO, the search spaces are represented as a hypercube, i.e., the wolves move to be either closer to or further away from the hypercube by adjusting specific values (which must be binarized as 0 or 1). It is impossible to continue utilizing the same equations, as mentioned above, in GWO to keep it up to date [32]. BGWO follows the mathematical model of GWO to obtain the values of α, β, and δ and utilizes Equation (3) to acquire $\overrightarrow{D}_{\alpha}$, $\overrightarrow{D}_{\beta}$, and $\overrightarrow{D}_{\delta}$. Next, the sigmoid function (S_1) is applied to obtain s_1, s_2, and s_3, which can be formulated using the following equations:

$$s_1^d = 1/(1 + e^{-10(A^d.D_{\alpha}^d - 0.5)}) \tag{34}$$

$$s_2^d = 1/(1 + e^{-10(A^d.D_{\beta}^d - 0.5)}) \tag{35}$$

$$s_3^d = 1/(1 + e^{-10(A^d.D_{\delta}^d - 0.5)}) \tag{36}$$

where d refers to the wolf's d^{th} dimension.

The $bstep_1$, $bstep_2$, and $bstep_3$ values are calculated using Equations (37)–(39). Next, the result provides a binary value rather than a continuous one. Then, as shown in the Equations (34)–(36), we use the transfer function to perform the switch. The values of 0 and 1 are used for the comparison with random numbers:

$$bestp_1^d = \begin{cases} 1\ if\ (s_1^d \geq randn) \\ 0\ else \end{cases} \tag{37}$$

$$bestp_2^d = \begin{cases} 1\ if\ (s_2^d \geq randn) \\ 0\ else \end{cases} \tag{38}$$

$$bestp_3^d = \begin{cases} 1\ if\ (s_3^d \geq randn) \\ 0\ else \end{cases} \tag{39}$$

where $randn$ is a number chosen at random from the range [0,1]. $bstep_1$, $bstep_2$, and $bstep_3$ are the distances that wo moved relative to α, β, and δ (wo mentioned above in GWO). The following equations are used to determine X_1, X_2, and X_3:

$$X_1^d = \begin{cases} 1\ if\ (X_{\alpha}^d + bstep_1^d \geq 1) \\ 2\ else \end{cases} \tag{40}$$

$$X_2^d = \begin{cases} 1\ if\ (X_{\alpha}^d + bstep_2^d \geq 1) \\ 2\ else \end{cases} \tag{41}$$

$$X_3^d = \begin{cases} 1\ if\ (X_{\delta}^d + bstep_3^d \geq 1) \\ 2\ else \end{cases} \tag{42}$$

Finally, we employ straightforward stochastic crossover, illustrated by Equation (43), to update the location of X_{wo} in the subsequent iteration:

$$X_{sh}^{d}\,(nt) = \begin{cases} X_1^d \; if\; (rand < 1/3) \\ X_2^d \; if\; (1/3 \leq rand < 2/3) \\ X_3^d \; else \end{cases} \tag{43}$$

In turn, BGWO employs the FS method, which works by selecting the features of the purity classification through a summary formed by selecting the relevant features and ignoring the others for the sake of the maximization of the classification accuracy (see Figure 7).

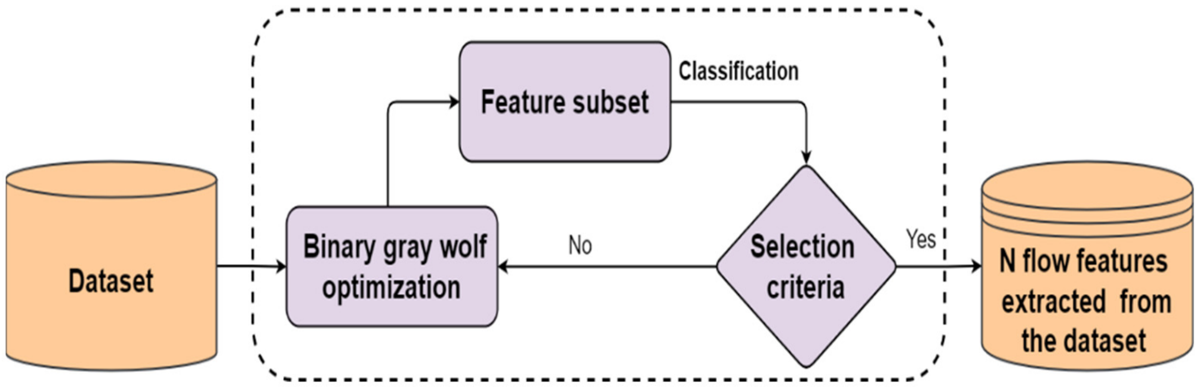

Dataset
Binary gray wolf optimization
Feature subset
Classification
Selection criteria
No
Yes
N flow features extracted from the dataset

Figure 7. BGWO FS method of purity classification.

4.2.3. Multiple-Attack-Based Dataset(s) Subsampling Phase

The reduced dataset with a subset of flow features already extracted using the BGWO FS method is split into multiple binary and balanced datasets. Each sub-dataset is derived based on a specific attack, i.e., a binary label. The derived dataset is subsampled randomly for use in the subsequent stage(s) and reduces the system complexity.

4.3. Classification Stage

We use a type of ML called the "supervised learning approach". Each instance in the dataset is classified into a label or category. Classification enables us to construct a model that assigns a subset of features to each category. The ELM classifier is introduced as one of the speediest learning algorithms. It is speedier than the multitude of back-propagation-based classifiers and meets the requirements related to the performance speed in many applications, especially those that run in real time.

ELM Hyperparameter Optimization

In the abovementioned learning algorithm, many parameters affect the classifier performance and change the evaluation outcomes, such as the accuracy, FAR, and others. These parameters include the number of hidden neurons, the activation function, the learning rate, and the number of iterations. The proposed system uses meta-heuristic physics and swarm algorithms to determine the best values for the hyperparameters so as to overcome such challenges. Figure 8 shows the major roles of the designed algorithm(s) in identifying the best values. The tuned dataset is first resampled randomly to generate a balanced binary subsample dataset per attack. Hence, the first stage outputs a set of the binary subsampled dataset(s) (i.e., one sub-dataset per attack), which is/are used to construct multiple binary ELM classifier(s) (i.e., software anomaly-based recognizers per attack). The binary ELM classifiers are utilized, and the hyperparameter is tuned using one of the designed physics and swarm-based search methods. Most ELMs significantly reduce the computational burdens and use one single-hidden-layer feedforward neural network, which involves using a “One-Versus-All” binary classifier that is applied to each class for the purpose of simple and fast processing. Hence, the most common parameter required for tuning the ELM classifier is the number of neurons of the single hidden layer.

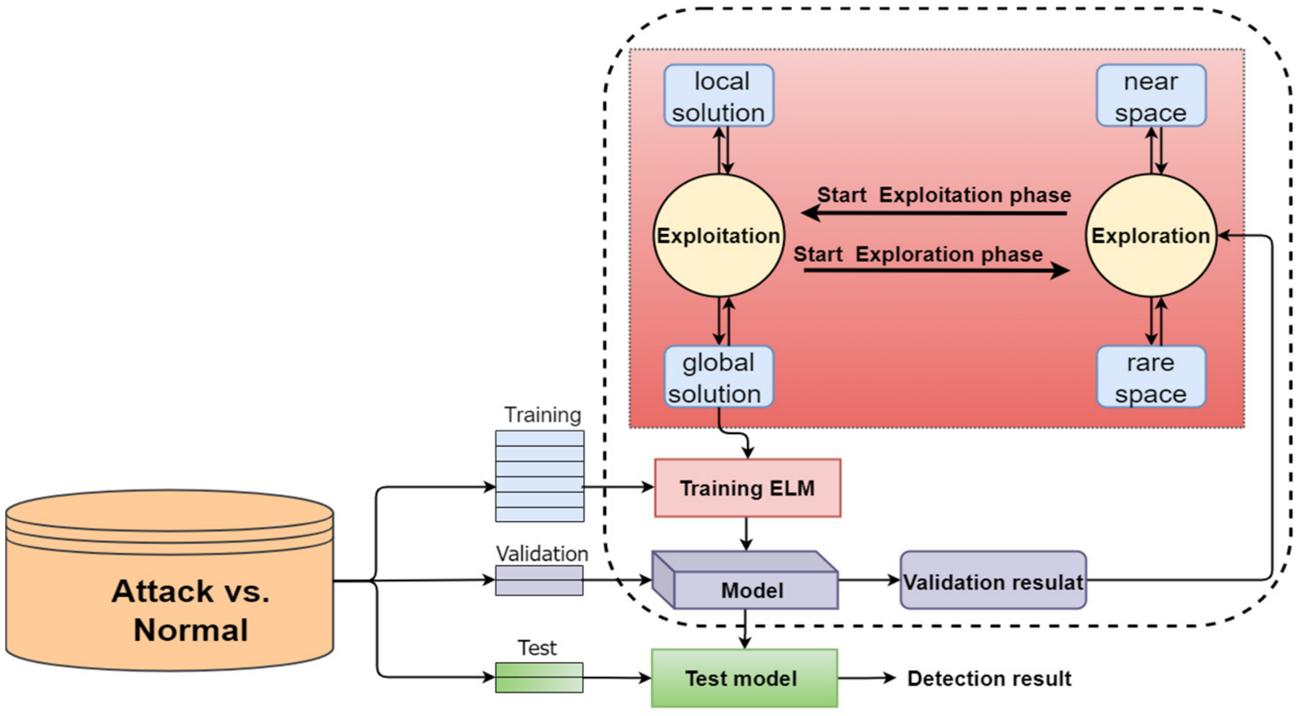


Figure 8. Hyperparameter optimization of a classification model.

Figure 9 represents an ELM tuning procedure using the physics meta-heuristic-based AOA. In AOA, we search for the best values of the most important field of the ELM to increase the accuracy when detecting and recognizing attacks. The proposed search occurs within the possible range of the number of hidden neurons and selects the best one per binary attack classifier. The ad hoc goal function is defined to construct a binary ELM classifier using the suggested value via AOA and to check the model’s performance in terms of its accuracy. Figure 10 represents the complete flowchart involved in the use of HBA to identify the optimal value of the number of hidden neurons, which is considered a hyperparameter and is used by the ELM classifier.

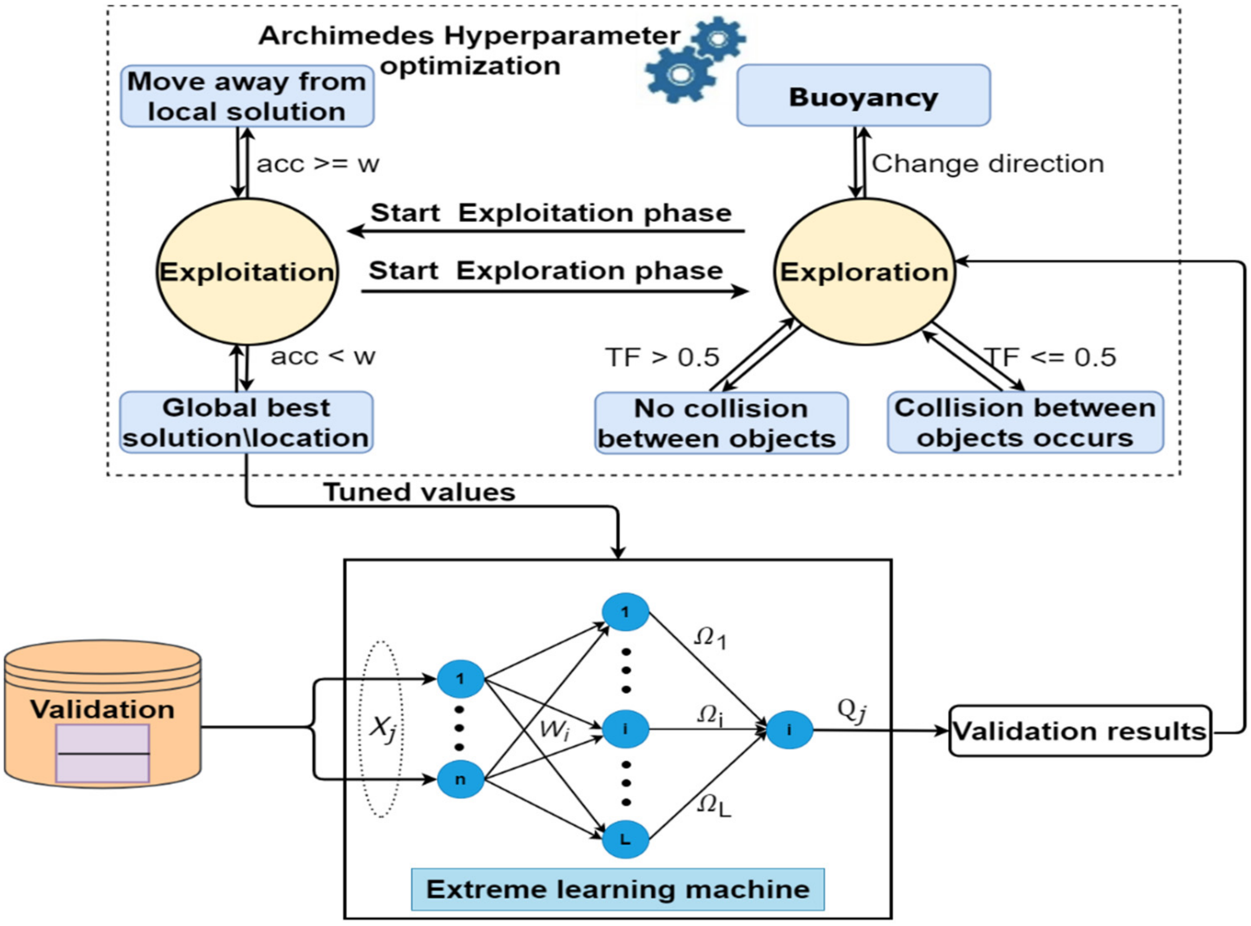
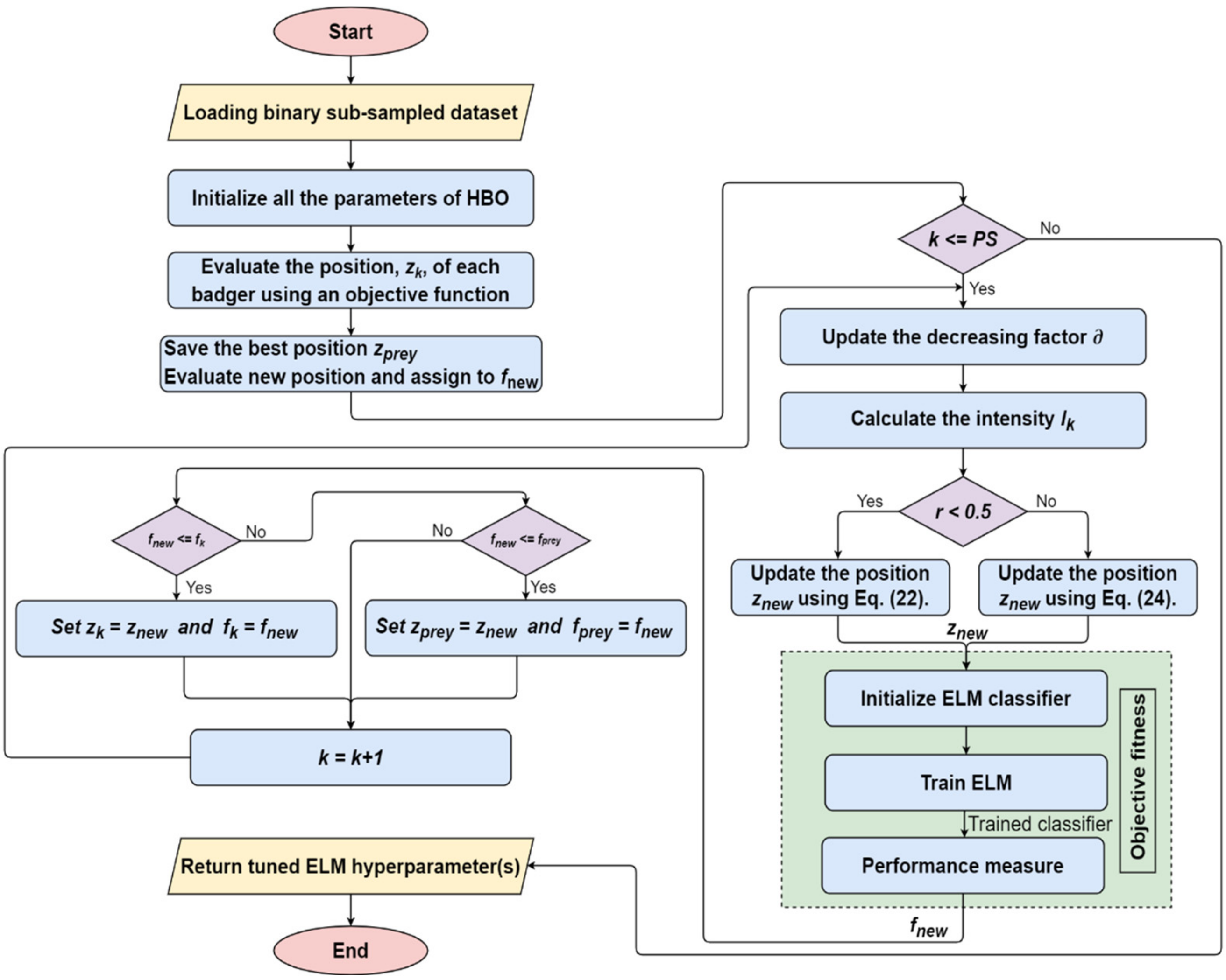


Figure 9. The designed AOA for tuning the ELM parameters.

4.4. Aggregated Hierarchical Classifiers Stage

The proposed multi-class classification issue was broken down into a collection of binary classifications to reduce the complexity burden on the classifiers in the aggregation.

The aggregation process is applied for the purpose of merging the multiple binary ELM classifiers so as to classify the inputs and categorize them into the multiple available groups. Figure 11 shows the flow steps of the aggregation of the ELM model, which is employed to detect anomalies and determine the attack type. The procedure of the binary models depends on the performance metric of the accuracy. The proposed aggregation scheme is dynamic and structured based on the number of attack types and their performance metrics. For the input flow packets, the aggregation model checks the anomaly skewness in a hierarchical order, starting with the most accurate binary ELM model and proceeding down to the least accurate binary ELM model. When the flow packet used passes all the check nodes, it is marked as normal activity; otherwise, the checking procedure alerts the system administrator to the instance, blocks it, and flags its type.



Start
Loading binary sub-sampled dataset
Initialize all the parameters of HBO
Evaluate the position, z_k, of each badger using an objective function
Save the best position z_{prey}
Evaluate new position and assign to f_{new}
$k <= PS$
No
Yes
Update the decreasing factor ∂
Calculate the intensity I_k
$r < 0.5$
Yes
No
Update the position z_{new} using Eq. (22).
Update the position z_{new} using Eq. (24).
z_{new}
Initialize ELM classifier
Train ELM
Trained classifier
Performance measure
Objective fitness
f_{new}
$f_{new} <= f_k$
$f_{new} <= f_{prey}$
Set $z_k = z_{new}$ and $f_k = f_{new}$
Set $z_{prey} = z_{new}$ and $f_{prey} = f_{new}$
$k = k+1$
Return tuned ELM hyperparameter(s)
End

Figure 10. Flowchart of the proposed HBA hyperparameter optimization algorithm.

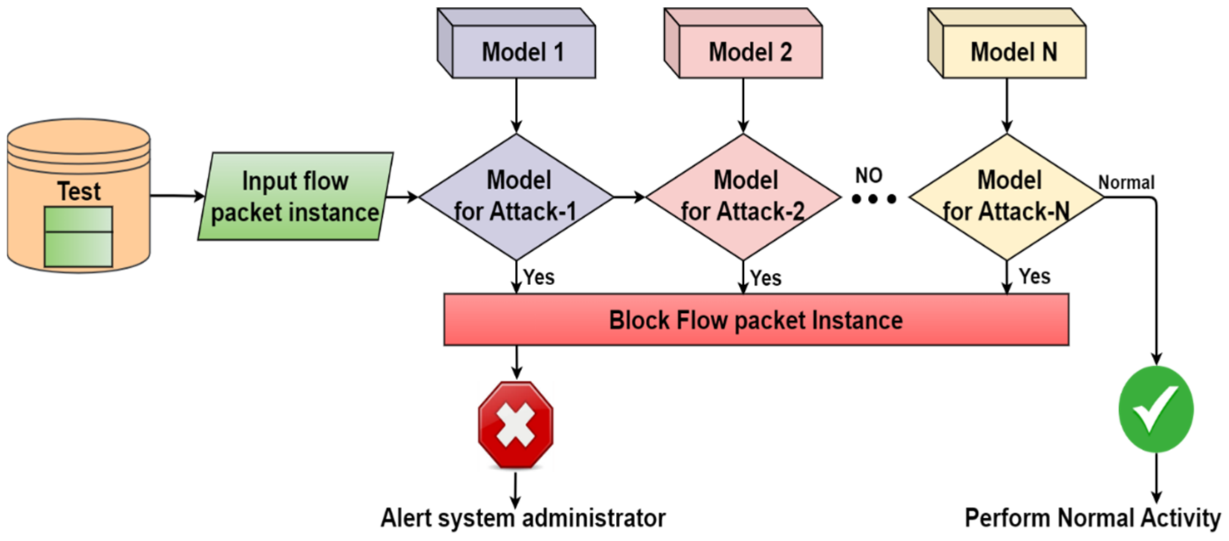


Figure 11. Aggregation of multiple binary ELM models.

5. Experimental Results and Discussion

5.1. Benchmark Datasets

Due to their numerous benefits over the other outdated standard datasets, such as KDD98, KDDCUP99, and NSLKDD, which are almost 20 years old and have proven insufficient for modern cybersecurity, lacking data on recent attack types and having inadequate normal instances, the UNSW-NB15 and CICIDS2017 benchmark datasets for IDS were used here.

5.1.1. UNSW-NB 15 Dataset

The UNSW-NB 15 dataset was created with the assistance of the IXIA Perfect Storm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) at the University of New South Wales. The researchers can use this dataset to better illustrate, validate, and test their proposal in a way that simulates real-world limitations and scenarios based on the ACCS declaration. The UNSW-NB 15 dataset is a significant standard and is the most realistic and challenging in the context of the intrusion detection problem. The dataset includes one hundred gigabytes' worth of raw network traffic. It covers recent modern, normal, and synthetic attack cases obtained from network traffic by configuring three virtual servers in its setup. The UNSWNB15 dataset has 2,540,044 records (instances) described by 49 attributes and organized into six distinct categories. These categories are as follows: flow features, fundamental features, content features, time features, extra produced features, and labeled features. The training dataset contains details on several forms of attack, which serve as labels. The distribution of each record included in the UNSW-NB15 dataset is displayed in Table 2. The records can be broken down into two primary categories: normal and attack. The data on the attacks are then organized into nine families according to the kinds of attacks carried out [40].

Table 2. Descriptions of different classes and the distribution of instances in the UNSW-NB15 dataset [40,41].

Class	No	Discerption
Normal	2,218,761	Benign transaction instances.
Generic	215,481	This is an attack style in which the attacker does not care how the underlying cryptographic primitives are implemented. Consider a cypher text protected by a V-bit key. In a brute-force assault of this type, the attacker would attempt every conceivable combination of these V bits, or 2^V in total.
Exploits	44,525	The attacker's planned series of operations aim to exploit an exploitable flaw in a target system or network. The attacker is aware of a security flaw in a given system or piece of software and uses this information to their advantage.
Fuzzers	24,246	An attacker takes action in searching for a security flaw in a system or network by flooding it with false data to bring it down.
Dos	16,353	A deliberate attempt to prevent legitimate users from accessing a server or network resource, typically by temporarily disrupting or stopping such services on a host connected to the internet.
Backdoors	2329	The process of secretly bypassing a system's security measures to gain unauthorized access to a system or its data and, potentially, to issue commands from outside the compromised system.

Table 2. *Cont.*

Class	No	Discerption
Reconnaissance	13,987	A group of attackers pretend to gather information about a computer system or network to access the security controls.
Analysis	2677	Utilized to penetrate online applications using various techniques, including port scanning, spam, and HTML file penetrations.
Shellcode	1511	The attacker creates malicious code and injects it into any program that launches a command shell, exploiting the software's flaws with minimal effort.
Worms	174	The attacker makes copies of their working files to spread to other computers. Most of the time, it spreads through a computer network by taking advantage of the weak security of the target computer.

5.1.2. CICIDS2017 Dataset

The Canadian Institute for Cybersecurity (CIC) released the CICIDS2017 dataset in late 2017 [42]. This dataset includes both normal instances and the latest instances of commonly used attacks. According to a study published by McAfee in 2016, the CICIDS2017 dataset includes the most widespread attacks. Data were collected and analyzed for five days using an open-source tool called CICFlowMeter [16]. The CICIDS2017 database simulates actual network activity. The attack profiles built by the developers insert various attack instances into fourteen families according to the kinds of attacks carried out other than normal network packets (instances). The CICIDS2017 dataset not only includes cutting-edge examples of cyberattacks carried out in networks, but it also satisfies all the other conditions typical of real-life cyberattacks [43]. Table 3 displays the distribution of the different traffic classifications in the dataset, where the number of instances is 2,830,743, and these are described by 78 features. The extracted features include the destination port, flow time, total forward packets, and total reverse packets.

Table 3. The description of different categories, classes, and instances' distribution in the CICIDS2017 dataset [18,42].

Category	Class	No	Description
Normal	Normal	2,359,087	Benign connection instances.
Dos	Dos Hulk	231,072	The goal of the attack is to render a computer or network resource temporarily inaccessible, overloading systems with unnecessary requests to block some or all valid requests from being completed. It is a common method used in this attack.
	DoS GoldenEye	10,293	
	DoS slowloris	5796	
	DoS Slowhttptest	5499	
DDos	DDos	41,835	The victim's bandwidth or resources are overloaded when many systems work together. These attacks often include many hacked computers (a botnet, for example) sending a deluge of traffic to the infiltrated server.
FTP-Patator	FTP-Patator	7938	Secure shell—representation of a brute force attack.
SSH-Patator	SSH-Patator	5897	File transfer protocol FTP-Patator of a brute force attack.

Table 3. *Cont.*

Category	Class	No	Description
Web	Web Attack—Brute Force	1507	Today, these new attacks are appearing daily because people and businesses are now taking security seriously. We use SQL Injection, in which an attacker constructs a string of SQL commands and uses them to coerce a database into returning the information, Cross-Site Scripting (XSS), which occurs when developers fail to properly test their code to determine the possibility of script injection, and Brute Force over HTTP, which uses a list of passwords to try to identify the administrator's password.
	Web Attack—XSS	652	
	Web Attack—Sql Injection	21	
PortScan	PortScan	158,930	Used to identify the access port on a network. An attacker can use this to learn about the listening habits of both the sender and the recipient.
Bot	Bot	1966	A collection of computers and other networked computers utilized by a botnet's creator to carry out their malicious plans. It gives an attacker access to the computer and its network and may be used to steal information or deliver spam.
Infiltration	Infiltration	36	Insider attacks utilize weak software such as Adobe Acrobat Reader. After successful exploitation, a backdoor is installed on the victim's machine and may perform IP sweeps, complete port scans, and Nmap service enumerations.
Heartbleed	Heartbleed	11	This a problem in the Open SSL cryptographic library, a widespread TLS implementation tool. It is abused by sending a faulty heartbeat request to a susceptible party (typically a server) to elicit a response.

5.2. Evaluation Results Using the UNSW-NB15 and CICIDS2017 Datasets

A binary meta-heuristic FS method is used to enhance the software-based anomaly IDS system to extract the most relevant of the available features. The FS method reduces the number of features by 65.31% and 51.28% for UNSW-NB15 and CICIDS2017, respectively, representing the most informative features. Table 4 lists the features used to classify the normal and attack data from the UNSW-NB15 and CICIDS2017 datasets. The configuration used for the BGWO is the FS method, which selects the features of the purity classification, including the value of each parameter (the number of iterations = 100, population size = 50). The objective function is the classification accuracy.

Table 4. Features selected from the datasets.

Dataset	Number of Available Features	Number of Selected Features	Index of Informative Features
UNSW-NB15	49	17	1,5,12,13,17,20,21,22,23,25, 28,32,33,38,43,44,46
CICIDS2017	78	38	3,5,7,12,13,14,17,18,19,21, 23,25,27,29,30,31,32,34,35,38, 39,41,46,49,51,52,53,56,57,58, 61,63,64,67,69,71,73,76

Due to the fact that the performance of the ELM classifier is affected by changed parameters, the hyperparameter optimization methods AOA and HBA were proposed to select the optimal value of the number of hidden neurons.

AOA and HBA are meta-heuristic-based optimization algorithms that identify the best hyperparameter values for ELM per attack. These algorithms' configurations include the values per parameter (the number of iterations = 100, population size = 75), and the fitness function is the accuracy metric. Table 5 represents the hyperparameter values suggested by AOA and HBA for the attack list of UNSW-NB15 and CICIDS2017 datasets.

Table 6 depicts the statistical analysis performance of the ELM classifier per attack using UNSW-NB15. The tuned ELM classifier using AOA achieves maximum accuracy and precision values of 99.62% and 100, respectively. It reaches values of 99.19%, 98.64%, and 98.91% for the average DR, specificity, and F1-score, respectively, for different attacks, and it achieves a superior FAR compared to the others for the attacks of worms. Additionally, it shows the tuned ELM using HBA, which exceeds DR by 100% for analysis attacks, with averages of 97.84%, 98.65%, 97.87%, and 98.63% in terms of the precision, accuracy, specificity, and F1-score, respectively. For FAR, the performance of the ELM tuned by HBA ranges from zero for a generic attack up to 0.04% for worms, which represent an average of 0.02% of all the attacks.

Table 5. The number of hidden neurons for the hyperparameter optimization of the ELM classifier, using the AOA and HBA algorithms for UNSW-NB15 and CICIDS2017 datasets.

Dataset	**Attack**	**Number of Hidden Neurons**	
		AOA	**HBA**
UNSW-NB15	Exploits	1479	**1293**
	Reconnaissance	883	**754**
	DoS	**716**	721
	Generic	**2495**	2817
	Shellcode	**198**	199
	Fuzzers	**904**	926
	Worms	47	**39**
	Backdoor	287	287
	Analysis	**311**	330
CICIDS2017	DDoS	**489**	671
	PortScan	**327**	354
	Bot	**37**	44
	Infiltration	**69**	70
	Web_Attack_Brute_Force	77	**68**
	Web_Attack_XSS	350	**281**
	Web_Attack_Sql_Injection	**900**	911
	FTP-Patator	289	**214**
	SSH-Patator	108	**93**
	DoS slowloris	83	**79**
	DoS Slowhttptest	**77**	83
	DoS Hulk	**3476**	3512
	DoS GoldenEye	**883**	384
	Heartbleed	**17**	23
	DDoS	**489**	671
	PortScan	**327**	354
	Bot	**37**	44
	Infiltration	**69**	70
	Web_Attack_Brute_Force	77	**68**

Table 6. Overall analysis of the tuned ELM classifier using the UNSW-NB15 reduced features.

Classifier	#Features	Tuning Algorithm	Statistic	Precision	DR	Accuracy	Specificity	FAR	F1-Score
ELM	17	AOA	Min	97.63%	95.92%	98.10%	97.57%	0%	97.92%
			Max	100%	99.83%	99.62%	100%	0.02%	99.62%
			Ave	98.65%	99.19%	98.93%	98.64%	0.01%	98.91%
		HBA	Min	96%	97.96%	97.14%	96.43%	0%	96.97%
			Max	99.54%	100%	99.58%	99.54%	0.04%	99.58%
			Ave	97.84%	99.44%	98.65%	97.87%	0.02%	98.63%

Table 7 illustrates the statistical analysis of the ELM classifiers' performance per attack using the CICIDC2017 dataset. The proposed tuned classifier achieves the best performance for Heartbleed in terms of the precision, DR, accuracy specificity, and F1-score, all of which exceed those of the others by 100%, and it achieves the minimum FAR of zero, regardless of whether AOA or HBA are used for the tuning. However, the tuned AOA classifier achieves better values in terms of the FAR than ELM using HBA.

Table 7. Overall analysis of the tuned ELM classifier using the CICIDS2017 reduced features.

Classifier	#Features	Tuning Algorithm	Statistic	Precision	DR	Accuracy	Specificity	FAR	F1-Score
ELM	38	AOA	Min	97.87%	99.34%	98.72%	97.96%	0%	98.66%
			Max	100%	100%	100%	100%	2.04%	100%
			Ave	99.48%	99.78%	99.63%	99.49%	0.51%	99.63%
		HBA	Min	77.78%	90.00%	84.62%	66.67%	0%	87.50%
			Max	100%	100%	100%	100%	33.33%	100%
			Ave	97.01%	99.04%	97.74%	95.54%	4.46%	98.02%

In term of the accuracy of each binary ELM classifier, the aggregation module creates a hierarchy classifier comprising multiple binary recognizers. Figure 12 represents the aggregated hierarchical multiple binary classifiers for the UNSW-NB15 dataset. Figure 12a demonstrates the reduced subset features and the use of AOA for the tuning of ELM binary classifiers, whereas Figure 12b illustrates the details of the use of HBA for the tuning of binary ELM classifiers. Figure 13 represents the aggregated hierarchical multiple binary classifiers for the CICIDS2017 dataset using the reduced features. Figure 13a illustrates the order of fourteen binary tuned classifiers using AOA based on the accuracy. Figure 13b demonstrates the tree compositions of the fourteen binary classifiers based on the accuracy using HBA for the hyperparameter optimization. A comprehensive analysis of the proposed system outcomes are discussed in terms of the performance metrics of the supervised learning, including the accuracy, DR, FAR, precision, and F1-score, versus the recent contributions and IDS systems, as illustrated in the next section.

5.3. Discussion

Here, we evaluate the use of the proposed BGWO FS and optimized ELM-based AOA and HBA as the primary detector for a hierarchical aggregation system. Our analysis confirms that the proposed system can achieve a remarkable accuracy, precision, DR, specificity, and F1-score, all of which are higher than those of the other state-of-the-art systems. The FAR of an IDS should be kept to a minimum, and the DR should be kept at a maximum. The proposed system performs exceptionally well, with a far lower FAR and higher DR than those in previous studies. This section compares the proposed approach with the most up-to-date research methods in terms of the accuracy, DR, FAR, precision, and F1-score so as to validate the effectiveness and reliability of the proposed system.

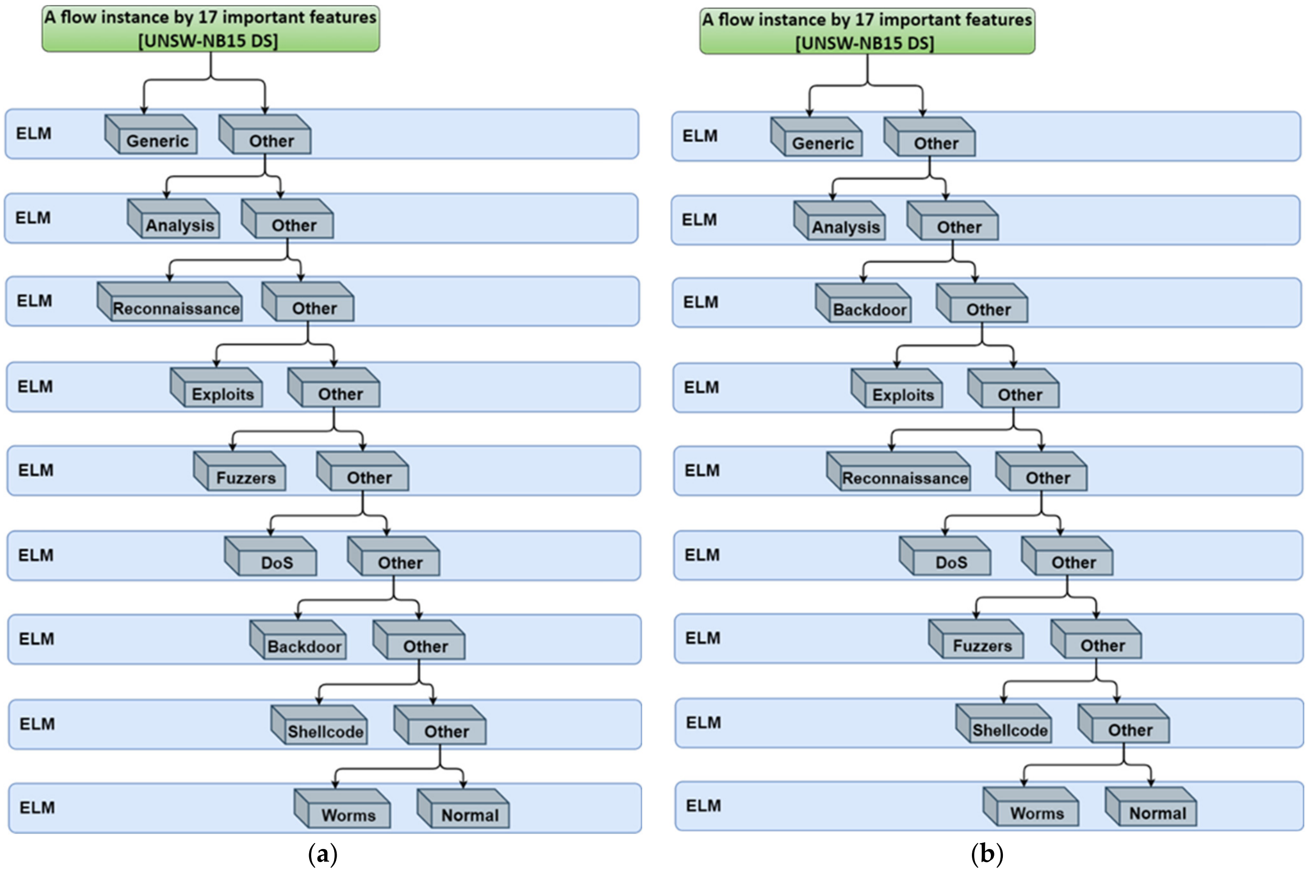


Figure 12. Aggregated hierarchical multiple binary classifiers for the UNSW-NB15 dataset are listed as: (**a**) description of the aggregated hierarchical multiple binary classifiers for AOA optimization; (**b**) description of the aggregated hierarchical multiple binary classifiers for HBA optimization.

Accuracy is a metric for determining which model best identifies the relationships and trends between variables in a dataset based on the input data. It calculates this by dividing the total number of correct classifications by the total number of classifications made and formulated using the following equation:

$$\text{Accuracy} = \frac{TP + TN}{FN + TP + FP + TN} \tag{44}$$

where the true positive (TP) is the number of positive instances (attack events) detected correctly. The true negative (TN) is the number of negative instances (normal events) detected correctly. The false positive (FP) refers to the number of non-positive events mistakenly identified as positive. The false negative (FN) refers to the number of positive events incorrectly identified as negative [44].

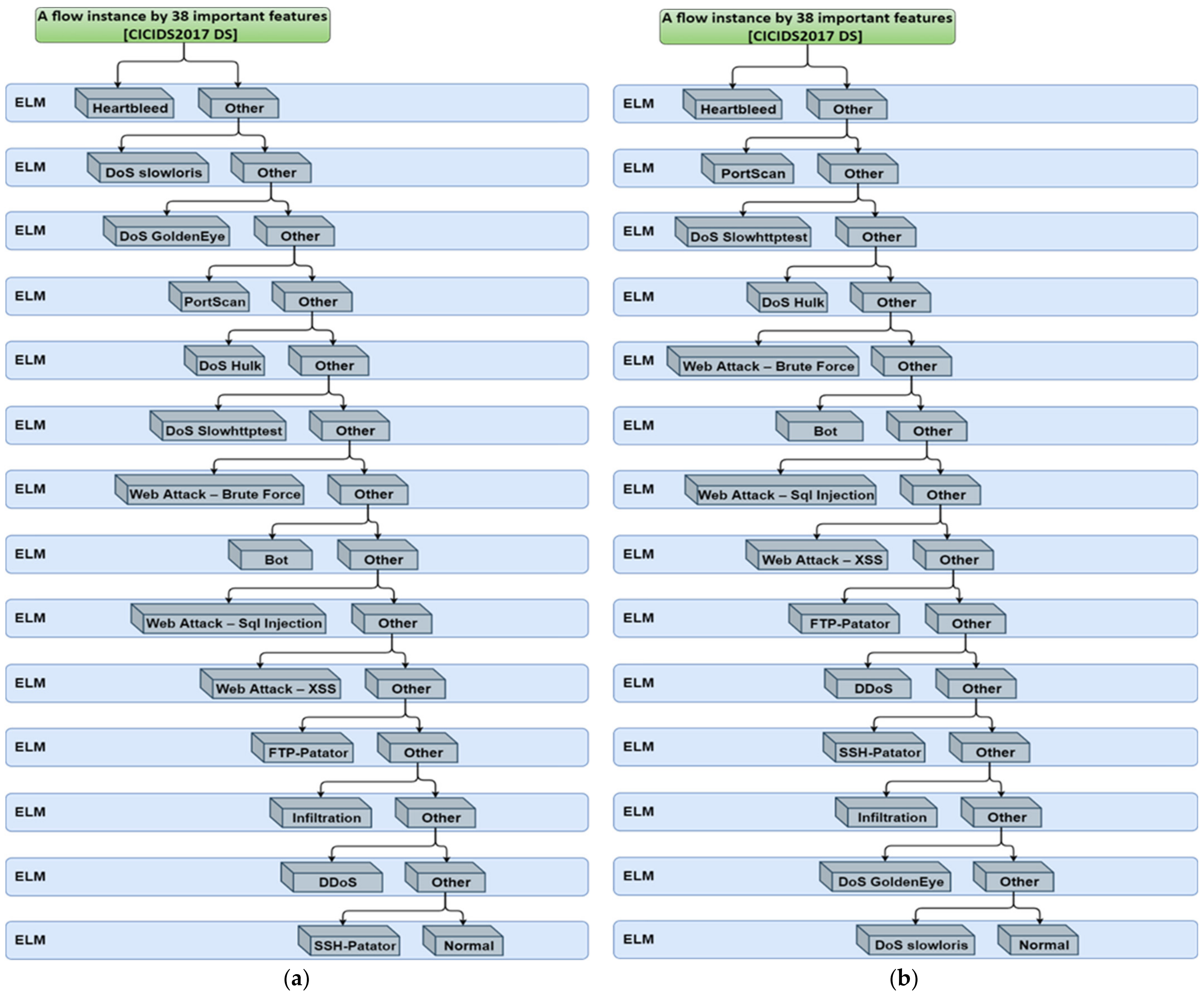


Figure 13. Aggregated hierarchical multiple binary classifiers for the CICIDS2017 dataset, listed as: (**a**) description of the aggregated hierarchical multiple binary classifiers for AOA optimization; (**b**) description of the aggregated hierarchical multiple binary classifiers for HBA optimization.

Table 8 illustrates the accuracy recognition rate in the context of UNSW-NB15, where the proposed system reached the most significant average accuracy rates of 98.93% when using the AOA–ELM, which is the top result compared to all the other systems, and 98.65% when using HBA-ELM, which is the second-best result compared to all the other systems. The sample systems reached an average accuracy of 98.79% $\pm$ 0.14, representing the general population's standard deviation of the accuracy rates.

Table 8. Accuracy comparison of the proposed system in relation to the other systems using the UNSW-N15 dataset.

Attack	The Proposed AOA-ELM	The Proposed HBA-ELM	Moualla et al. [12]	Ren et al. [14]	Sharma et al. [45]	Gu et al. [15]	Jiang et al. [46]	Rajagopal [47]	Manjunatha et al. [48]	Vinayakumar et al. [19]
Exploits	**99.03%**	**98.9%**	93.91%	92.6%	90.12%	84.2%	79.21%	76.22%	84.2%	89.9%
Reconnaissance	**99.03%**	98.73%	**98.74%**	98.8%	95.33%	95.7%	89.45%	20.77%	95.7%	92.7%
DoS	**98.97%**	98.71%	98.14%	93.1%	94.9%	94.9%	92.12%	83.8%	94.9%	**99.4%**
Generic	**99.62%**	99.58%	98.34%	**100%**	98.23%	91.5%	96.37%	11.51%	91.5%	78.3%
Shellcode	98.57%	97.91%	**99.92%**	99.2%	99.4%	**99.5%**	92.79%	18.4%	**99.5%**	99%
Fuzzers	**98.98%**	98.67%	98.92%	95.3%	91.47%	91.6%	93.43%	29.36%	91.6%	98.8%
Worms	98.1%	97.14%	97.28%	**100%**	99.92%	99.9%	65.31%	15%	99.9%	99.9%
Backdoor	98.71%	98.93%	99.06%	98%	**99.11%**	**99.2%**	83.53%	49%	**99.2%**	95.1%
Analysis	99.38%	99.25%	**99.44%**	98.2%	99.26%	99.1%	84.67%	58%	99.1%	**99.55%**
Average	**98.93%**	**98.65%**	98.19%	97.24%	96.42%	95.07%	86.32%	40.23%	94.51%	94.74%

Underlined and bold: the best result amongst the systems. Bold: the second-best result amongst the systems.

Table 9 shows the details of the accuracy recognition rate of each class in CICIDS2017 obtained by Choobdar et al. [20], Zhiqiang et al. [49], Vinayakumar et al. [19], and our system. From Table 9, the accuracy values in relation to all the attack types achieved by the proposed approach are higher than those of the other three approaches, except for the results of Zhiqiang et al. [49], who reached a 99.8% accuracy in DDoS attacks. The accuracies achieved range from 98.72% to 100% for SSH-Patator applied to DoS goldeneye, DoS slowloris, and Heartbleed, respectively. Moreover, the proposed system achieved an average accuracy of 99.63% versus the values of 96.07% derived by Choobdar et al. [20], 88.58% derived by Zhiqiang et al. [49], and 93.99% derived by Vinayakumar et al. [19]. The proposed system achieved percentage increases of approximately 3.56%, 11.05%, and 5.64% in terms of the average accuracy compared to the other systems.

Table 9. Accuracy comparison of the proposed system in relation to the other systems using the CICIDS2017 dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Choobdar et al. [20]	Zhiqiang et al. [49]	Vinayakumar et al. [19]
DoS Hulk	**99.92%**	**99.91%**	99.2%	97.48%	NA
PortScan	**99.97%**	**99.96%**	98.5%	99.72%	85.5%
DDoS	**98.89%**	98.72%	98.2%	**99.8%**	85.5%
DoS GoldenEye	**100%**	90.91%	95.2%	**95.84%**	NA
FTP-Patator	**99.45%**	**98.78%**	98.7%	98.71%	NA
SSH-Patator	**98.72%**	**98.47%**	94.8%	91.57%	95.8%
DoS slowloris	**100%**	84.62%	**98.4%**	97.62%	92.8%
DoS Slowhttptest	**99.94%**	**99.92%**	87.7%	85.52%	NA
Bot	**99.75%**	**99.69%**	98.2%	31%	95.9%
Web Attack—Brute Force	**99.83%**	**99.83%**	95.2%	NA	**98.8%**
Web Attack—XSS	**99.61%**	**99.48%**	95.3%	NA	98.8%
Infiltration	**99.14%**	98.38%	**98.9%**	NA	NA
Web Attack—Sql Injection	**99.68%**	**99.66%**	97%	NA	98.8%
Heartbleed	**100%**	**100%**	89.7%	NA	NA
Average	**99.63%**	**97.74%**	96.07%	88.58%	93.99%

Some researchers are more interested in DR (recall) than other metrics. The DR shows that the critical metric for IDS is the proportion of successfully classified attacks relative to the total attack instances, and it is formulated using the following equation:

$$\text{DR} = \frac{TP}{TP + FN} \tag{45}$$

When addressing these issues, we concentrate more on attack instances than usual, because wrongly classifying attacks from among attack instances causes more damage than cases when they are wrongly classified from among normal instances.

Table 10 demonstrates that the proposed system achieves a better average DR when using HBA–ELM and AOA–ELM, yielding values of 99.44% and 99.19%, respectively, compared to the other systems. Table 10 shows that the proposed system achieves the best DR for most attack type(s), and the system set out by Moualla et al. [12] achieves higher DR values of 99.86% and 99.91% for shellcode and worms attacks, respectively.

Table 10. DR comparison of the proposed system with other systems using the UNSW-N15 dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Moualla et al. [12]	Ren et al. [14]	Jagruthi et al. [50]	Rajagopal et al. [47]	Wang et al. [18]
Exploits	**99.62%**	**99.55%**	86.05%	66.3%	97%	85%	60.4%
Reconnaissance	99.73%	**99.76%**	93.16%	82%	72%	74.8%	66.7%
DoS	**99.65%**	**99.48%**	82.47%	46.1%	57%	25%	41.4%
Generic	**99.65%**	**99.63%**	97.05%	96.6%	96%	98.32%	99.8%
Shellcode	**99.56%**	98.90%	**99.86%**	78%	11%	58.22%	62%
Fuzzers	**99.83%**	**99.79%**	95.8%	38.1%	0.1%	60.97%	62.8%
Worms	**95.92%**	97.96%	**99.91%**	79.5%	1.6%	37.5%	50%
Backdoor	**99.14%**	**99.86%**	98.11%	40.3%	64%	10.79%	0%
Analysis	**99.63%**	**100%**	98.89%	6.1%	60%	11%	64.8%
Average	**99.19%**	**99.44%**	94.59%	59.22%	50.97%	51.29%	56.43%

Table 11 shows the detailed DR of each class obtained by the proposed system and the other systems. Table 11 shows that the proposed system achieves the best and second-best average DR values for each attack type, namely 99.78% and 99.04%, when using HBA–ELM and AOA–ELM.

Table 11. DR comparison of the proposed system in relation to other systems using the CICIDS2017 dataset.

Attack	The Proposed AOA-ELM	The Proposed HBA-ELM	Choobdar et al. [20]	Lee et al. [21]	Ho et al. [51]	Ferrag et al. [52]	Hosseini et al. [53]	Lee et al. [22]	Wang et al. [18]	Toupas et al. [43]
DoS Hulk	**99.93%**	**99.93%**	98.5%	99.34%	**99.96%**	96.78%	98.8%	99.73%	89.4%	99.25%
PortScan	**99.99%**	**99.99%**	97.1%	99.95%	**99.99%**	99.88%	99.79%	99.96%	92.1%	99.79%
DDoS	99.83%	99.49%	97.5%	**99.93%**	**99.94%**	99.87%	99.9%	99.92%	70.4%	99.9%
DoS Golden Eye	**100%**	90.00%	93%	99.42%	99.92%	67.57%	99.27%	**99.44%**	89.4%	99.27%
FTP-Patator	99.34%	99.34%	95.4%	**99.84%**	**99.73%**	99.63%	99.59%	**99.84%**	77.1%	99.59%
SSH-Patator	**99.46%**	**99.46%**	95.6%	**99.75%**	99.32%	99.9%	98.97%	99.75%	97.3%	98.97%
DoS slowloris	**100%**	**100%**	96%	99.48%	**99.65%**	97.75%	89.93%	99.31%	89.4%	89.93%
DoS Slowhttptest	**100%**	**100%**	88.1%	**99.05%**	99.63%	93.84%	86.87%	89.95%	89.4%	86.76%
Bot	**99.66%**	**99.66%**	**97.3%**	53.13%	66.37%	46.47%	95.12%	54.51%	87.4%	95.11%
Web Attack—Brute Force	**99.72%**	**99.72%**	87.6%	60%	**99.53%**	73.26%	98.31%	94.84%	94.5%	98.31%

Table 11. *Cont.*

Attack	The Proposed AOA-ELM	The Proposed HBA-ELM	Choobdar et al. [20]	Lee et al. [21]	Ho et al. [51]	Ferrag et al. [52]	Hosseini et al. [53]	Lee et al. [22]	Wang et al. [18]	Toupas et al. [43]
Web Attack—XSS	**99.52%**	**99.52%**	96.2%	60%	92.8%	30.62%	**98.31%**	94.84%	94.5%	**98.31%**
Infiltration	99.61%	**99.65%**	98.2%	60%	91.66%	**100%**	81.66%	66.67%	NA	81.66%
Web Attack—Sql Injection	**99.90%**	**99.87%**	95%	60%	80.95%	50%	98.31%	94.84%	94.5%	98.31%
Heartbleed	**100%**	**100%**	88.7%	**100%**	**100%**	**100%**	95%	**100%**	NA	**95%**
Average	**99.78%**	**99.04%**	94.59%	84.99%	94.96%	82.54%	95.70%	92.40%	88.26%	96.84%

A high FAR value significantly reduces the effectiveness of the IDS. Even if the value of FAR can be kept to a minimum, this does not indicate that the system is entirely safe and immune to assault. FAR is a reducing function representing the proportion of normal instances incorrectly classified as attacks and is formulated using the following equation:

$$\text{FAR} = \frac{FP}{FP + TN} \tag{46}$$

Traditional ML-based IDSs suffer from high FAR values, mainly due to the fact that dataset imbalance is not considered. The proposed system maintains the lowest FAR values, even though the sample size of the worm attacks in the UNSW-NB 15 dataset is very small, with only 174 instances, and infiltration, web attack—SQL Injection, and Heartbleed attacks in the CICIDS2017 only number 36, 21, and 11, respectively. This indicates that the proposed system can learn the features of the data more effectively and carry out accurate classification with a small amount of data. Tables 12 and 13 demonstrate that the proposed system achieves a lower FAR than the other systems.

Table 12. FAR comparison of the proposed system in relation to other systems using the UNSW-N15 dataset.

Attack	The Proposed AOA-ELM	The Proposed HBA-ELM	Moualla et al. [12]	Ren et al. [14]	Wang et al. [18]	Salman et al. [54]
Exploits	**0.02%**	**0.02%**	**0.09%**	0.34%	2.9%	1.40%
Reconnaissance	**0.02%**	**0.02%**	**0.04%**	0.18%	2.4%	4.90%
DoS	**0.02%**	**0.02%**	**0.09%**	0.54%	7.6%	4.20%
Generic	**0%**	**0.00%**	0.16%	**0.03%**	0.9%	0.39%
Shellcode	**0.02%**	0.03%	**0%**	0.22%	0.68%	11%
Fuzzers	**0.02%**	**0.02%**	**0.03%**	0.62%	4.7%	NA
Worms	**0%**	0.04%	**0.03%**	0.21%	0.08%	20%
Backdoor	**0.02%**	**0.02%**	**0.01%**	0.20%	1.2%	3.70%
Analysis	**0.01%**	0.02%	**0.01%**	0.39%	1.3%	7.83%
Average	**0.01%**	**0.02%**	0.05%	0.30%	2.42%	6.68%

Table 13. Comparison of FAR values of the proposed system and other systems using the CICIDS dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Wang et al. [18]
DoS Hulk	**0.09%**	**0.11%**	1.40%
PortScan	**0.04%**	**0.06%**	0.09%
DDoS	**2.04%**	**2.04%**	**0.80%**
DoS GoldenEye	**0%**	8.33%	**1.40%**
FTP-Patator	**0.44%**	1.78%	**0.32%**
SSH-Patator	**1.93%**	2.42%	**1.30%**
DoS slowloris	**0%**	33.33%	**1.40%**
DoS Slowhttptest	**0.13%**	**0.17%**	1.40%
Bot	**0.17%**	**0.28%**	0.32%
Web Attack—Brute Force	**0.06%**	**0.06%**	**0.34%**
Web Attack—XSS	**0.31%**	0.55%	**0.34%**
Web Attack—Sql Injection	**1.34%**	12.75%	**0.34%**
Infiltration	0.55%	0.55%	NA
Heartbleed	0%	0%	NA
Average	**0.51%**	4.46%	**0.79%**

Precision refers to the system's ability to determine how many positive classifications are correct. It is calculated by dividing the number of true positives by the number of events that were classified as positive and can be formulated using the following equation:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{47}$$

As is observable from Tables 14 and 15, the precision rate of the proposed system is considerably higher in relation to all the attack categories than that of the other systems, indicating that the proposed system's overall performance is good.

Table 14. Comparison of the precision of the proposed and other systems using the UNSW-N15 dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Moualla et al. [12]	Ren et al. [14]	Jagruthi et al. [50]	Rajagopal et al. [47]	Wang et al. [18]
Exploits	**98.44%**	98.25%	91%	75.9%	**100%**	63.41%	90.1%
Reconnaissance	**98.33%**	**97.7%**	93%	9%	73%	90.65%	68%
DoS	**98.29%**	97.93%	**100%**	35.1%	53%	41.6%	7.6%
Generic	99.58%	99.54%	**100%**	**99.8%**	91%	99.42%	97.7%
Shellcode	**97.63%**	96.98%	**100%**	35.2%	72%	68.65%	15.2%
Fuzzers	**98.13%**	97.58%	**98%**	94.2%	40%	64.42%	65.4%
Worms	**100%**	**96%**	95%	77.8%	33%	57.69%	3.2%
Backdoor	**98.29%**	98.03%	**100%**	15.1%	77%	70%	0%
Analysis	**99.14%**	98.55%	**100%**	4.6%	44%	67.44%	100%
Average	**98.65%**	**97.84%**	97.44%	58.50%	64.78%	69.25%	49.69%

Table 15. Comparison of the precision of the proposed and other systems using the CICIDS 2017 dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Choobdar et al. [20]	Lee et al. [21]	Toupas et al. [43]	Lee et al. [22]	Wang et al. [18]
DoS Hulk	**99.91%**	**99.88%**	98.6%	99.59%	99.77%	99.63%	92.9%
PortScan	**99.96%**	**99.94%**	98.5%	99.37%	97.94%	99.38%	82.9%
DDoS	97.99%	97.98%	97.1%	**99.9%**	99.82%	**99.99%**	80.9%
DoS GoldenEye	**100%**	90%	96.9%	**99.56%**	97.54%	99.42%	92.9%
FTP-Patator	99.56%	98.26%	93.2%	**100%**	98.68%	**99.97%**	72.4%
SSH-Patator	97.87%	97.35%	93.2%	**100%**	99.05%	**99.66%**	94.6%
DoS slowloris	**100%**	77.78%	96.2%	99.74%	92.91%	**99.61%**	92.9%
DoS Slowhttptest	**99.87%**	**99.83%**	87.3%	99.14%	93.23%	99%	92.9%
Bot	**99.83%**	**99.72%**	95.9%	86.31%	71.92%	83.69%	83.6%
Web Attack—Brute Force	**99.94%**	**99.94%**	96.40%	**99.41%**	95.59%	99.40%	92.30%
Web Attack—XSS	**99.70%**	**99.46%**	97.3%	99.41%	95.59%	99.40%	92.30%
Infiltration	**98.67%**	98.56%	98.20%	**100%**	79.54%	**100%**	NA
Web Attack—Sql Injection	**99.45%**	**99.45%**	96.5%	**99.41%**	95.59%	99.40%	92.30%
Heartbleed	**100%**	**100%**	**88.40%**	**100%**	**100%**	**100%**	NA
Average	**99.48%**	97.01%	95.26%	**98.70%**	94.08%	98.47%	96.47%

In regard to the F1-score, it is common for the recall and precision to be in a state of trade-off. The score is commonly used as a performance metric for evaluating the effectiveness of IDS, since it considers both the precision and recall. Mathematically, it is expressed using the following equation:

$$\text{F1} - \text{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{48}$$

Using the confusion matrix derived from the testing of the proposed system, Table 16 shows the comprehensive F1-scores for each attack category gained by the proposed and other systems using the UNSW-NB15 dataset, which confirms the superiority of the proposed system in relation to the exploits, reconnaissance, DoS, generic, fuzzers, and worm attacks compared to Moualla et al. [12], Ren et al. [14], Jiang [46], and Jagruthi et al. [50]. The system set out by Moualla et al. [12] achieved higher F1-scores in relation to shellcode and backdoor attacks. As a result, the proposed system has a higher F1-score average (of 98.91%) than the other systems.

Table 16. Comparison of F1-scores of the proposed and other systems using the UNSW-NB15 dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Moualla et al. [12]	Ren et al. [14]	Jiang [46]	Jagruthi et al. [50]
Exploits	**99.03%**	**98.9%**	88.45%	70.8%	67.89%	98%
Reconnaissance	**99.03%**	**98.72%**	93.11%	85.3%	62.54%	73%
DoS	**98.96%**	**98.7%**	90.39%	39.9%	29.55%	55%
Generic	**99.62%**	**99.58%**	98.41%	98.3%	**98.85%**	94%
Shellcode	**98.59%**	97.93%	**99.92%**	48.6%	30.95%	19%
Fuzzers	**98.97%**	**98.68%**	96.34%	54.2%	37.47%	11%
Worms	**97.92%**	96.97%	**97.34%**	78.7%	10.75%	22%
Backdoor	98.71%	**98.93%**	**99.05%**	21.9%	8.97%	70%
Analysis	**99.39%**	99.27%	**99.44%**	5.3%	9.69%	0.1%
Average	**98.91%**	**98.63%**	95.83%	55.89%	39.63%	49.12%

Table 17 shows the comprehensive F1-scores of each attack category gained by the proposed and other systems using the CICIDS2017 dataset. Here, the proposed system achieved higher F1-scores for DoS hulk, portscan, DoS goldeneye, DoS slowloris, DoS Slowhttptest, bot, web attack—brute force, web attack—XXS, infiltration, and web attack—SQL Injection attacks compared to Choobdar et al. [20], J. Lee et al. [21], Toupas et al. [43], and Lee et al. [22]. The system set out by Lee et al. [21] achieved higher F1-scores for DDoS, FTP-Patator, and SSH-Patator than the other systems. The proposed system achieved a higher average F1-score of 99.63% than the other systems, which is a high value.

Table 17. Comparison of the F1-scores of the proposed and other systems using the CICIDS 2017 dataset.

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Choobdar et al. [20]	Lee et al. [21]	Toupas et al. [43]	Lee et al. [22]
DoS Hulk	**99.92%**	**99.91%**	96.3%	99.47%	99.25%	99.68%
PortScan	**99.97%**	**99.96%**	97.6%	99.66%	98.82%	99.67%
DDoS	98.90%	98.73%	98%	**99.96%**	**99.86%**	**99.96%**
DoS GoldenEye	**100%**	90%	85.8%	**99.49%**	98.35%	99.43%
FTP-Patator	**99.45%**	98.80%	95.9%	**99.92%**	99.12%	**99.92%**
SSH-Patator	98.66%	98.40%	92.5%	**99.87%**	**99%**	**99.87%**
DoS slowloris	**100%**	87.50%	98.3%	**99.61%**	88.85%	99.46%
DoS Slowhttptest	**99.94%**	**99.92%**	88.6%	99.09%	87.64%	98.98%
Bot	**99.75%**	**99.69%**	97.3%	65.77%	79.72%	65.94%
Web Attack—Brute Force	**99.83%**	**99.83%**	94.8%	97.73%	96.91%	**97.07%**

Table 17. *Cont.*

Attack	The Proposed AOA–ELM	The Proposed HBA–ELM	Choobdar et al. [20]	Lee et al. [21]	Toupas et al. [43]	Lee et al. [22]
Web Attack—XSS	**99.61%**	**99.49%**	97%	97.73%	96.91%	97.07%
Infiltration	**99.14%**	**99.10%**	98.2%	75%	79.16%	80%
Web Attack—Sql Injection	**99.67%**	**99.66%**	95%	97.73%	96.91%	97.07%
Heartbleed	**100%**	**100%**	88.3%	**100%**	**96.66%**	**100%**
Average	**99.63%**	**98.02%**	94.54%	95.07%	94.08%	95.29%

6. Conclusions

This paper proposed a new optimized IDS based on multiple hierarchical ELM classifiers. The new proposed IDS utilizes two novel meta-heuristic optimization algorithms, AOA and HBA, to enhance the classification performance of ELM, in addition to using enhanced BGWO optimization, which selects the most important features. As a result, the proposed system introduces a multi-class classifier for detecting attack types. It implicitly trains multiple binary classifiers to detect the differences between normal and attack instances. Each ELM classifier is introduced to accurately detect a specific attack class defined by the "One-Versus-All" method. In the newly proposed optimized IDS, the multi-class classification issue is broken down into a collection of binary classifications, as using binary classifications entails lower a complexity than multi-class classification. Each classifier is tuned to quickly obtain the optimal neuron number of the hidden layer.

The newly proposed IDS incorporates a three-stage pipeline that can be summarized by algorithms such as BGWO feature selection, the ELM classifiers, AOA, and HBA for tuning the ELM hyperparameters, which are themselves selected to construct adaptable and fast binary classifiers. Two benchmark datasets, UNSW-NB15 and CICIDS2017, were utilized for the validation. The newly proposed IDS is adaptable to any new dataset. The accuracy, DR, FAR, precision, specificity, and the F1-score were the evaluation metrics used to show that the proposed system is better than the other systems in the same general category. All the systems were evaluated using the same benchmark datasets, thus ensuring that the comparisons are unbiased and fair. The experimental results reveal that the proposed system considerably increased the overall accuracy and F1-score, giving values up to 98.9% and 99.6%m compared to the other systems using the UNSWNB-15 and CICIDS2017 datasets, respectively. Furthermore, it maintains the highest precision and DR and the lowest FAR. These results are promising. Additionally, according to the research results, the proposed IDS is highly successful in detecting attack types in imbalanced datasets. In future work, IDS could be used to independently utilize feature selection for each attack type and create a new classification approach to identify attack types.

Author Contributions: Conceptualization, K.A.E., A.A.A. and B.I.H.; methodology, K.A.E., A.A.A. and B.I.H.; software, B.I.H.; validation, K.A.E. and A.A.A.; formal analysis, B.I.H.; investigation, K.A.E. and A.A.A.; resources, K.A.E., A.A.A. and B.I.H.; data curation, B.I.H.; writing—original draft preparation, B.I.H.; writing—review and editing, K.A.E. and A.A.A.; visualization, B.I.H.; project administration, K.A.E.; funding acquisition, K.A.E., A.A.A. and B.I.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: UNSW-NB15 dataset available at URL: https://research.unsw.edu.au/projects/unsw-nb15-dataset (accessed on 1 October 2022). And CICIDS2017 dataset available at URL: https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 1 October 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. DataReportal—Global Digital Insights. Digital 2022: Global Overview Report—DataReportal—Global Digital Insights. Available online: https://datareportal.com/reports/digital-2022-global-overview-report (accessed on 8 September 2022).
2. Mahdavisharif, M.; Jamali, S.; Fotohi, R. Big data-aware intrusion detection system in communication networks: A deep learning approach. *J. Grid Comput.* **2021**, *19*, 46. [CrossRef]
3. Qureshi, A.-U.-H.; Larijani, H.; Mtetwa, N.; Javed, A.; Ahmad, J. RNN-ABC: A new swarm optimization based technique for anomaly detection. *Computers* **2019**, *8*, 59. [CrossRef]
4. Thakkar, A.; Lohiya, R. A survey on intrusion detection system: Feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artif. Intell. Rev.* **2021**, *55*, 453–563. [CrossRef]
5. Hameed, B.; AlHabshy, A.A.; ElDahshan, K.A. Distributed Intrusion Detection Systems in Big Data: A Survey. *Al-Azhar Bull. Sci.* **2021**, *32*, 27–44. [CrossRef]
6. Azeez, N.A.; Ayemobola, T.J.; Misra, S.; Maskeliūnas, R.; Damaševičius, R. Network intrusion detection with a hashing based apriori algorithm using Hadoop MapReduce. *Computers* **2019**, *8*, 86. [CrossRef]
7. Milenkoski, A.; Vieira, M.; Kounev, S.; Avritzer, A.; Payne, B.D. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Comput. Surv.* **2015**, *48*, 1–41. [CrossRef]
8. Ahmad, Z.; Khan, A.S.; Shiang, C.W.; Abdullah, J.; Ahmad, F. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Trans. Emerg. Telecommun. Technol.* **2021**, *32*, e4150. [CrossRef]
9. Sarker, I.H.; Kayes, A.; Badsha, S.; Alqahtani, H.; Watters, P.; Ng, A. Cybersecurity data science: An overview from machine learning perspective. *J. Big Data* **2020**, *7*, 41. [CrossRef]
10. Abou-Kreisha, M.T.; Yaseen, H.K.; Fathy, K.A.; Ebeid, E.A.; ElDahshan, K.A. Multisource Smart Computer-Aided System for Mining COVID-19 Infection Data. *Healthcare* **2022**, *10*, 109. [CrossRef]
11. Elzeki, O.; Sarhan, S.; Elfattah, M.A.; Salem, H.; Shams, M.Y. Biomedical Healthcare System For Orthopedic Patients Based On Machine Learning. *J. Eng. Appl.* **2006**, *16*, 616–622.
12. Moualla, S.; Khorzom, K.; Jafar, A. Improving the performance of machine learning-based network intrusion detection systems on the UNSW-NB15 dataset. *Comput. Intell. Neurosci.* **2021**, *2021*, 1–13. [CrossRef] [PubMed]
13. Wong, P.K.; Yang, Z.; Vong, C.M.; Zhong, J. Real-time fault diagnosis for gas turbine generator systems using extreme learning machine. *Neurocomputing* **2014**, *128*, 249–257. [CrossRef]
14. Ren, J.; Guo, J.; Qian, W.; Yuan, H.; Hao, X.; Jingjing, H.J.S. Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms. *Secur. Commun. Netw.* **2019**, *2019*, 7130868. [CrossRef]
15. Gu, J.; Lu, S. An effective intrusion detection approach using SVM with naïve Bayes feature embedding. *Comput. Secur.* **2021**, *103*, 102158. [CrossRef]
16. Faker, O.; Dogdu, E. Intrusion detection using big data and deep learning techniques. In Proceedings of the 2019 ACM Southeast Conference, Kennesaw, GA, USA, 18–20 April 2019; Kennesaw State University: Kennesaw, GA, USA, 2019; pp. 86–93.
17. He, H.; Sun, X.; He, H.; Zhao, G.; He, L.; Ren, J. A novel multimodal-sequential approach based on multi-view features for network intrusion detection. *IEEE Access* **2019**, *7*, 183207–183221. [CrossRef]
18. Wang, Z.; Zeng, Y.; Liu, Y.; Li, D. Deep belief network integrating improved kernel-based extreme learning machine for network intrusion detection. *IEEE Access* **2021**, *9*, 16062–16091. [CrossRef]
19. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]
20. Choobdar, P.; Naderan, M.; Naderan, M. Detection and Multi-Class Classification of Intrusion in Software Defined Networks Using Stacked Auto-Encoders and CICIDS2017 Dataset. *Wirel. Pers. Commun.* **2022**, *123*, 437–471. [CrossRef]
21. Lee, J.; Park, K. GAN-based imbalanced data intrusion detection system. *Pers. Ubiquitous Comput.* **2021**, *25*, 121–128. [CrossRef]
22. Lee, J.; Park, K. AE-CGAN model based high performance network intrusion detection system. *Appl. Sci.* **2019**, *9*, 4221. [CrossRef]
23. Bolón-Canedo, V.; Sánchez-Maroño, N.; Alonso-Betanzos, A. Feature selection for high-dimensional data. *Prog. Artif. Intell.* **2016**, *5*, 65–75. [CrossRef]
24. Nadimi-Shahraki, M.H.; Banaie-Dezfouli, M.; Zamani, H.; Taghian, S.; Mirjalili, S. B-MFO: A binary moth-flame optimization for feature selection from medical datasets. *Computers* **2021**, *10*, 136. [CrossRef]
25. Xue, B.; Zhang, M.; Browne, W.N.; Yao, X. A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evol. Comput.* **2015**, *20*, 606–626. [CrossRef]

26. Abdel-Basset, M.; El-Shahat, D.; El-henawy, I.; de Albuquerque, V.H.C.; Mirjalili, S. A new fusion of grey wolf optimizer algorithm with a two-phase mutation for feature selection. *Expert Syst. Appl.* **2020**, *139*, 112824. [CrossRef]
27. Cui, X.; Li, Y.; Fan, J.; Wang, T.; Zheng, Y. A hybrid improved dragonfly algorithm for feature selection. *IEEE Access* **2020**, *8*, 155619–155629. [CrossRef]
28. El-Hasnony, I.M.; Barakat, S.I.; Elhoseny, M.; Mostafa, R.R. Improved feature selection model for big data analytics. *IEEE Access* **2020**, *8*, 66989–67004. [CrossRef]
29. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
30. Faris, H.; Aljarah, I.; Al-Betar, M.A.; Mirjalili, S. Grey wolf optimizer: A review of recent variants and applications. *Neural Comput. Appl.* **2018**, *30*, 413–435. [CrossRef]
31. Chantar, H.; Mafarja, M.; Alsawalqah, H.; Heidari, A.A.; Aljarah, I.; Faris, H. Feature selection using binary grey wolf optimizer with elite-based crossover for Arabic text classification. *Neural. Comput. Appl.* **2020**, *32*, 12201–12220. [CrossRef]
32. Hu, P.; Pan, J.-S.; Chu, S.-C. Improved binary grey wolf optimizer and its application for feature selection. *Knowl. Based Syst.* **2020**, *195*, 105746. [CrossRef]
33. Desuky, A.S.; Cifci, M.A.; Kausar, S.; Hussain, S.; El Bakrawy, L.M. Mud Ring Algorithm: A new meta-heuristic optimization algorithm for solving mathematical and engineering challenges. *IEEE Access* **2022**, *10*, 50448–50466. [CrossRef]
34. Hashim, F.A.F.; Hussain, K.; Houssein, E.H.; Mabrouk, M.S.; Al-Atabany, W. Archimedes optimization algorithm: A new metaheuristic algorithm for solving optimization problems. *Appl. Intell.* **2021**, *51*, 1531–1551. [CrossRef]
35. Hashim, F.A.; Houssein, E.H.; Hussain, K.; Mabrouk, M.S.; Al-Atabany, W. Honey Badger Algorithm: New metaheuristic algorithm for solving optimization problems. *Math. Comput. Simul.* **2022**, *192*, 84–110. [CrossRef]
36. Huang, G.-B.; Zhu, Q.-Y.; Siew, C.-K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
37. Huang, G.-B.; Wang, D.H.; Lan, Y. Extreme learning machines: A survey. *Int. J. Mach. Learn. Cybern.* **2011**, *2*, 107–122. [CrossRef]
38. Zhang, K.; Hu, Z.; Zhan, Y.; Wang, X.; Guo, K. A smart grid AMI intrusion detection strategy based on extreme learning machine. *Energies* **2020**, *13*, 4907. [CrossRef]
39. Ali, H.; Elzeki, O.M.; Elmougy, S. Smart Attacks Learning Machine Advisor System for Protecting Smart Cities from Smart Threats. *Appl. Sci.* **2022**, *12*, 6473. [CrossRef]
40. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, Australia, 10–12 November 2015; IEEE: Piscataway Township, NJ, USA, 2015; pp. 1–6.
41. Kumar, V.V.; Sinha, D.; Das, A.K.; Pandey, S.C.; Goswami, R.T. An integrated rule based intrusion detection system: Analysis on UNSW-NB15 data set and the real time online dataset. *Clust. Comput.* **2020**, *23*, 1397–1418. [CrossRef]
42. Sharafaldin, I.; Gharib, A.; Lashkari, A.H.; Ghorbani, A.A. Towards a reliable intrusion detection benchmark dataset. *Secur. Commun. Netw.* **2018**, *2018*, 177–200. [CrossRef]
43. Toupas, P.; Chamou, D.; Giannoutakis, K.M.; Drosou, A.; Tzovaras, D. An intrusion detection system for multi-class classification based on deep neural networks. In Proceedings of the 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), Boca Raton, FL, USA, 16–19 December 2019; IEEE: Piscataway Township, NJ, USA, 2019; pp. 1253–1258.
44. Salem, H.; Attiya, G.; El-Fishawy, N. Intelligent decision support system for breast cancer diagnosis by gene expression profiles. In Proceedings of the 2016 33rd National Radio Science Conference (NRSC), Aswan, Egypt, 22–25 February 2016; IEEE: Piscataway Township, NJ, USA, 2016; pp. 421–430.
45. Sharma, J.; Giri, C.; Granmo, O.-C.; Goodwin, M. Multi-layer intrusion detection system with ExtraTrees feature selection, extreme learning machine ensemble, and softmax aggregation. *EURASIP J. Inf. Secur.* **2019**, *2019*, 1–16. [CrossRef]
46. Jiang, K.; Wang, W.; Wang, A.; Wu, H. Network intrusion detection combined hybrid sampling with deep hierarchical network. *IEEE Access* **2020**, *8*, 32464–32476. [CrossRef]
47. Rajagopal, S.; Kundapur, P.P.; Hareesha, K.S. A stacking ensemble for network intrusion detection using heterogeneous datasets. *Secur. Commun. Netw.* **2020**, *2020*, 4586875. [CrossRef]
48. Manjunatha, B.; Gogoi, P.; Akkalappa, M. Data Mining based Framework for Effective Intrusion Detection using Hybrid Feature Selection Approach. *Int. J. Comput. Netw. Inform. Secur.* **2019**, *11*, 1–12. [CrossRef]
49. Zhiqiang, L.; Zhijun, L.; Ting, G.; Yucheng, S.; Ghulam, M.-U.-D. A three-layer architecture for intelligent intrusion detection using deep learning. In Proceedings of the Fifth International Congress on Information and Communication Technology, London, UK, 20–21 February 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 245–255.
50. Jagruthi, H.; Kavitha, C. A Novel Framework for NIDS Using Stacked Ensemble Learning. In *Soft Computing for Security Applications*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 115–127. [CrossRef]
51. Ho, S.; Al Jufout, S.; Dajani, K.; Mozumdar, M. A novel intrusion detection model for detecting known and innovative cyberattacks using convolutional neural network. *IEEE Open J. Comput. Soc.* **2021**, *2*, 14–25. [CrossRef]
52. Ferrag, M.A.; Maglaras, L.; Ahmim, A.; Derdour, M.; Janicke, H.J.F.i. Rdtids: Rules and decision tree-based intrusion detection system for internet-of-things networks. *Future Internet* **2020**, *12*, 44. [CrossRef]

53. Hosseini, S.; Seilani, H. Anomaly process detection using negative selection algorithm and classification techniques. *Evol. Syst.* **2021**, *12*, 769–778. [CrossRef]
54. Salman, T.; Bhamare, D.; Erbad, A.; Jain, R.; Samaka, M. Machine learning for anomaly detection and categorization in multi-cloud environments. In Proceedings of the 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, USA, 26–28 June 2017; IEEE: Piscataway Township, NJ, USA, 2017; pp. 97–103.