



Article

A Meta-Classification Model for Optimized ZBot Malware Prediction Using Learning Algorithms

Shanmugam Jagan ¹, Ashish Ashish ², Miroslav Mahdal ^{3,*} , Kenneth Ruth Isabels ⁴, Jyoti Dhanke ⁵, Parita Jain ⁶ and Muniyandy Elangovan ⁷ 

¹ Department of CSE, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai 600062, India; drsjagan@veltech.edu.in

² Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram 522302, India; dr.aashishsinha@kluniversity.in

³ Department of Control Systems and Instrumentation, Faculty of Mechanical Engineering, VSB-Technical University of Ostrava, 17. Listopadu 2172/15, 70800 Ostrava, Czech Republic

⁴ Department of Mathematics, Saveetha Engineering College (Autonomous), Chennai 600062, India; ruthisabels@saveetha.ac.in

⁵ Department of Engineering Science (Mathematics), Bharati Vidyapeeth's College of Engineering, Pune 412115, India; jyoti.dhanke@bharativedyapeeth.edu

⁶ Department of CSE, KIET Group of Institutions, Ghaziabad 201206, India; parita.jain@kiet.edu

⁷ Department of R&D, Bond Marine Consultancy, London EC1V2NX, UK; muniyandy.e@gmail.com

* Correspondence: miroslav.mahdal@vsb.cz

Abstract: Botnets pose a real threat to cybersecurity by facilitating criminal activities like malware distribution, attacks involving distributed denial of service, fraud, click fraud, phishing, and theft identification. The methods currently used for botnet detection are only appropriate for specific botnet commands and control protocols; they do not endorse botnet identification in early phases. Security guards have used honeypots successfully in several computer security defence systems. Honeypots are frequently utilised in botnet defence because they can draw botnet compromises, reveal spies in botnet membership, and deter attacker behaviour. Attackers who build and maintain botnets must devise ways to avoid honeypot traps. Machine learning methods support identification and inhibit bot threats to address the problems associated with botnet attacks. To choose the best features to feed as input to the machine learning classifiers to estimate the performance of botnet detection, a Kernel-based Ensemble Meta Classifier (KEMC) Strategy is suggested in this work. And particle swarm optimization (PSO) and genetic algorithm (GA) intelligent optimization algorithms are used to establish the ideal order. The model covered in this paper is employed to forecast Internet cyber security circumstances. The Binary Cross-Entropy (loss), the GA-PSO optimizer, the Softsign activation functions and ensembles were used in the experiment to produce the best results. The model succeeded because Fileless malware, gathered from well-known datasets, achieved a total accuracy of 93.3% with a True Positive (TP) Range of 87.45% at zero False Positive (FP).

Keywords: honeypots; botnet; malware; soft sign; genetic algorithm; kernels and cyber threats

MSC: 68T01; 68T05; 68U01



Citation: Jagan, S.; Ashish, A.; Mahdal, M.; Isabels, K.R.; Dhanke, J.; Jain, P.; Elangovan, M. A Meta-Classification Model for Optimized ZBot Malware Prediction Using Learning Algorithms. *Mathematics* **2023**, *11*, 2840. <https://doi.org/10.3390/math11132840>

Academic Editor: Andrea Scozzari

Received: 1 June 2023

Revised: 21 June 2023

Accepted: 21 June 2023

Published: 24 June 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Cybercrime is a huge threat to information security [1] predicts that the annual expenses associated with cybercrime could reach USD 10.5 hundred billion by 2025, with a significant portion of this cost attributable to identifying malicious such as financial malware. Banking spyware has also been growing annually, and according to research [1], attacks involving financial services malware have expanded by 80% in 2021 by itself. One of these financial services variants, ZBot (hereafter referred to as God), Currently ranks among the most prevalent forms of banking malware [1]. However, since 2011, the source

code has been released to the public, enabling the generation of a wide variety of novel variants [2,3]. But since the ZBot password was leaked, numerous ZBot varieties have emerged, including Ramnit, ZbotPanda and Citadel.

1.1. Motivation and Incitement

The background of the work revolves around developing a meta-classification model for optimized ZBot malware prediction using learning algorithms. ZBot malware, also known as Zeus malware [4,5], is a notorious type of malware that primarily targets financial institutions and individuals for fraud and identity theft. Detecting and predicting the presence of ZBot malware is crucial for effective cybersecurity. The researchers aim to address the limitations of traditional ZBot malware detection methods by proposing a meta-classification model [6]. A meta-classification model combines the outputs of multiple base classifiers to improve the overall prediction accuracy. By utilizing various learning algorithms within the meta-classification framework, the researchers intend to enhance the accuracy and efficiency of ZBot malware prediction.

The motivation for this work stems from the increasing sophistication and evolving nature of ZBot malware, which makes it challenging for traditional detection methods to keep up with the rapidly changing threat landscape [7]. By employing machine learning algorithms and creating an optimized meta-classification model, the researchers aim to overcome these challenges and provide more effective ZBot malware prediction. The significance of this work lies in its potential to enhance the detection and prevention of ZBot malware, thereby mitigating the risks associated with financial fraud and identity theft [8,9]. A successful meta-classification model can improve the efficiency of security systems by accurately identifying ZBot malware instances, enabling prompt responses and proactive measures to safeguard individuals and organizations. Thereby, the background of the work highlights the need for an advanced and optimized approach to ZBot malware prediction. By leveraging learning algorithms within a meta-classification framework, the researchers aim to develop a robust and efficient model that can contribute to the field of cybersecurity and aid in the fight against ZBot malware.

1.2. Architecture of the ZBot Malware

Using control and command paths (C&C) for communication, ZBot can infect various systems. The first figure shows the various stages of the Command-and-control communication discussed by Preethi and Asokan [8]. This communication can occur using either a centrally controlled architecture or a mentoring architecture, with the latter being extra Strong and resilient [2]. This is because the ZBot bots won't be able to receive commands, update software, or download new configuration files from the Command-and-control server if that server goes down or isn't available [10]. Recently developed Zbots use a P2P C&C architecture. Because the file format does not refer to a fixed Command and control server, they are more resistant to takedown attempts. If the C&C server goes down, a mentorship (proxy bot) can be used to learn the necessary information and even be updated [10]. The stolen information is sent across the command-and-control network to the C&C server of the malware's designers, where it is decoded and saved in a database [11,12].

According to [2], ZBot behaves like a virus and spreads mostly through spam schemes, making it one of the most common ways malware is disseminated and infects Windows computers. Ref. [13] conducted extensive research on this topic and discovered that phishing is responsible for approximately 90% of data breaches. When the ZBot binary is executed on a Windows computer, it performs several tasks. Creating separate files named *local.ds* and *user.ds* is one option. *local.ds* contains the dynamic configuration file that can be downloaded from the (C&C) server, whereas *user.ds* stores stolen credentials and other facts that must be data sent to the C&C [14]. To manage network traffic, *svchost* is modified to include additional code [15]. Moreover, *Svchost* is responsible for software injected maliciously into many Windows processes, enabling ZBot to steal credentials and launch

financial attacks. These include algorithms like KNN, random forest, and decision trees for machine learning. Our hopes for this piece are that it will:

- Identifying a method will allow deep learning and machine learning algorithms to detect ZBot malware.
- Find the best-performing detecting machine learning algorithm.
- Examine the data to see if the features that lead to the highest detection accuracy can be applied to other data.
- Recognize the bare minimum of indicators that may have been used to spot ZBot.
- Determine if the functionalities yielding the greatest detection results are compatible with older and newer ZBot types.

Identify whether the features responsible for the best ZBot detection findings apply to additional ZBot versions.

2. Related Works

The previous research used a network perimeter scanning system [14,16] with three features and a correlations motor to spot malicious traffic flows linking an infected site to a hostile force [17]. The application was developed to monitor the ransomware communication pipeline [18] from initial infection to final recovery [19]. It correlates outbound and incoming information to detect malicious activity. It is developed on top of the open software SNORT 3.1.56.0, San Jose, CA, USA. Both employ two plugins, SLADE and SCADE, the latter of which analyses communication to identify potentially harmful traffic patterns. Some examples of such circulation patterns are:

- Internet Protocol address lookup hosts.
- Disruptions in the outbound connection.
- A pattern of communication that is evenly distributed, which is probable that the communication is malicious.

A programme called [18] uses two modules, the A and C-plane modules, along with the plane. A module to monitor network traffic flows to detect clusters of infected computers [20]. The A-plane is in charge of determining what these servers are doing, while the C-plane collects network activity to identify all interacting hosts. It is possible to identify patterns of communication between hosts using characteristics that were extracted from both modules; if these patterns of communication are false, it means that a certain set of hosts is communicating maliciously. The plane A component, which depends on SCADE, can analyze the communications to spot harmful communication patterns. By training and analyzing the ZBot malware with the CFS and C4.5 classification algorithms, CONIFA [21] utilises machine learning techniques to identify the traffic in malware communication. It created an expense-customized form of the Classifier classification algorithm that uses strict and forgiving classifiers and compares the results for the prediction to a traditional framework called machine learning that uses a prize-insensitive copy of C4.5 binary classifier to increase CONIFA's accuracy and forecasting results after analyzing the training dataset. Generally, the conventional framework had a high detection rate. However, the recall rate dropped significantly when applied to the test dataset (83% to 57%). The CONIFA findings showed an improvement in accuracy, with a 67% increase in recall.

To find malware communication traffic, the Detector called RCC [14] monitors the network activity leaving a host. The RCC does this by using a TP and an MLP classifier. The MLP classifier is accustomed to categorising botnets based on factors like flow number, session time, uniformity score, and the Kolmogorov-Smirnov Test. It is made up of three layers they are input, output, and hidden layer. MLFFN [14] is a programme that uses TCP features to recognize botnet communication traffic by retrieving them from TCP connections that truly came from a host machine. The MLFFN [22] comprises a four-neuron output vector and six-neuron input nodes. ZBot-1, ZBot-2, Spyeeye-1, and Spyeeye-2 data sets—previous iterations of the Odin malware—were used to test MLFFN.

3. Methodology for KEMC: ZBot Malware Prediction

The advantage of using this paradigm at this time is that it emphasizes the control and command channel, which is presumed as a bot in the early stages of communication—Figure 1. The first stage is preprocessing when duplication is reduced, and data normalization is carried out, representing the design of the suggested classification algorithm. The essential features are chosen during the feature selection process, and these attributes are used as input by ML techniques to gauge the model’s effectiveness. Then, for the initial step of C&C identification of botnets, the superior one is selected. The proposed botmaster’s connection to the internet and its route through the control and command servers, which communicated with the workstations directly, are shown in Figure 2.

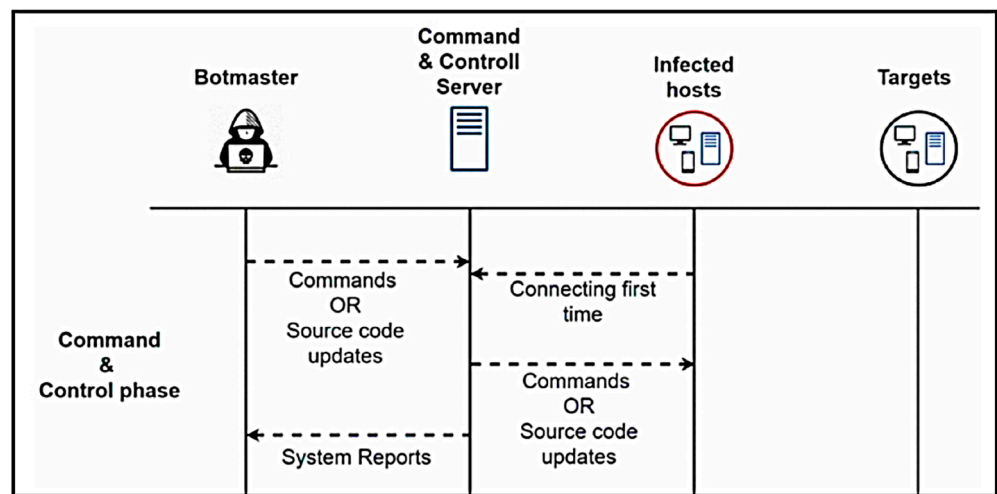


Figure 1. C&C Communication Stages.

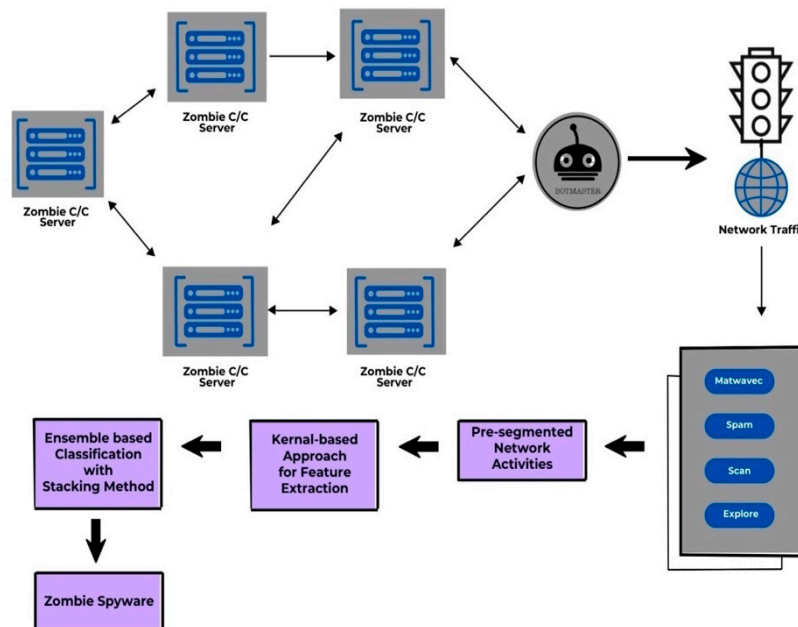


Figure 2. The architecture of the proposed botnet classification method.

3.1. Dataset Description

We use the publicly accessible Center for Cyber Clean collection, which includes the C09, C10, C08, and C13 datasets and contains packets with traffic of 6667 as port numbers for IRC & 80 for HTTP. A C&C server connection is necessary for the bot.

3.2. Kernel-Based Feature Selection

The main advantage of the method using kernel is that for a higher-dimensional decision boundary, we are optimising a primitive function that solely considers the dot product of the feature vectors updated. As a result, with the kernel function, we can easily swap out these dot product statements. Using this plan, the selection of features is optimized.

A point pool n is produced for the training phase D :

$$D = \left\{ (x_i, y_i)_{i=1}^n, x_i \in \mathbb{R}^d \text{ and } y_i \in \{+1, -1\} \right\} \tag{1}$$

Depending on the category to which x_i belongs, the value of y_i is either 1 or -1, a vector which is x_i with p dimensions. Any hyperplane with a set of points must have x to satisfy the conditions.

$$w \cdot x + b = 0 \tag{2}$$

where “ \cdot ” stands for the dot-product and “ w ” stands for the vector normally. The offset of the hyperplane along w is b/w in origin. In addition, resolving the following optimization issue involves evaluating the parameters (b, w) .

$$\begin{aligned} & \underset{w; b}{\operatorname{argmin}} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\ & \text{s.t. } \forall i, y_i \{w \cdot x_i + b\} \geq 1 - \xi_i; \xi_i \geq 0 \end{aligned} \tag{3}$$

The measure of regularization C is used to balance the trade-off between the error of the training sample and the margin. To achieve a good generalization performance, the value of C should be greater than zero, with slack variables ξ_i that take on positive values. Essentially, C serves as a regularization measure in this context.

$$\max \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{j=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j k(x_i | x_j) \tag{4}$$

$$t \sum_{i=1}^n y_i \alpha_i = 0 \tag{5}$$

$$0 \leq \alpha_i \leq C, i, j = 1, 2, \dots, n$$

This aids in the development of various kernel techniques and the analysis of their convergence characteristics. Moreover, the kernel function is represented by $k(x_i, x_j) = (x_i)^T \Gamma(x_j)$, and Γ stands for the Lagrange coefficients. The optimal parameters, α^* , w^* , and b^* must meet the Karush-Kuhn Tucker (KKT) criteria.

$$\alpha_i^* \left[y_i \left(\sum_{j=1}^n \alpha_j^* y_j k(x_i, x_j) + b^* \right) - 1 + \xi_i \right] = 0, i, j = 1, 2, \dots, n \tag{6}$$

A tiny subset of j is typically non-zero when categorizing. According to Equation (9), the nonzero j are known as support vectors since they produce the ideal separation hyperplane:

$$w^{*T} \varphi(x) + b^* = \sum_{j=1}^n \alpha_j^* y_j k(x_i, x_j) + b^* = 0 \tag{7}$$

The many kernels optimize kernel weights during SVM training. Equation (9) is translated by employing multiple kernels, and a double formed for multiple kernels are generated as shown in Equations (10)–(13):

$$\max \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{j=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \sum_{i=1}^2 \beta_r k_l(x_i, x_j) \tag{8}$$

$$\text{s.t. } \sum_{j=1}^n y_j \alpha_j = 0, 0 \leq \alpha_j \leq C, j = 1, 2, \dots, n \tag{9}$$

$$\beta_l \geq 0, \sum_{i=1}^m \beta_l = 1 \tag{10}$$

The variables of the above equations are

x_i, x_j : Input feature vectors or data points in the dataset.

α_i^* : Lagrange coefficients or dual variables associated with each data point. These coefficients determine the importance of each data point in defining the decision boundary.

y_i : The corresponding class labels or target values for each data point.

w^* : The weight vector that defines the optimal hyperplane for classification.

b^* : The bias term or threshold for the optimal hyperplane. The problem is broken down into two steps: determining the best weights, I am using the sparse approach and lowering the soft marginal error function, I using linear programming.

3.3. Ensemble Categorization Using Stacking Technique

To address a problem using computer intelligence, ensemble learning involves the fusion of numerous models and deliberate generation, such as classifications or experts [23]. Ensemble learning is often utilized to improve performances (classification, function approximation, prediction etc.)

As seen in Equation (14), ensemble F typically includes a collection of forecasters of a function f_i' .

$$F = \{f_i', i = 1, \dots, k\} \tag{11}$$

A set of examples $\{x_1, f(x_1)\}, \dots, \{x_n, f(x_n)\}$ which are contained in the data. The goal is to use an induction method or learner to stimulate a real unknown function f' (16), as shown in the Equation (15): $f: X \rightarrow R^1$, where, $f'(X) = f(x) \in X$

A method that predicts or estimates future outcomes is referred to as a forecast or predictor, denoted by f' as illustrated in Equation (16). This regression aims to minimize the Squared of the Mean Error, a type of lost method that measures the squared average difference between the actual values and the predicted.

$$MSE = \frac{1}{n} \sum_i^n (f'(x_i) - f(x_i))^2 \tag{12}$$

$$MSE(f) = \text{bias}(f)^2 + \text{var}(f) \tag{13}$$

Consideration of M1, M2, and M3 is the only model where x_{train} and y_{train} are the training datasets with dependent and independent variables, respectively. The testing dataset can be considered as x_{test} and y_{test} . Train M1, M2, and M3 are not dependent on the same dataset results in M1. M2. Fit x_{train}, y_{train} -, Fit x_{train}, y_{train} -, and M3. Fit x_{train}, y_{train} , which produces corresponding predicted outputs y_{m3}, y_{m2} , and y_{m1} . The final predictions are the majority of the vote (F_p) among these outputs. The stacking method is used to produce second-level data, which is similar to K-fold cross-validation. This approach was provided out of sampled predictions and captured the unique strengths of each model. Linear stacking uses a linear combination with values a_i, a_m , as shown in Equation (18), to predictions of the stack are $f = f_1, f_2, f_3, \dots, f_m$. Cross-validation and stacked regression are excellent combinations.

$$F(\text{stack})(x) = \sum_{i=1}^m a_i f_i(x) \tag{14}$$

The input vector used to train the meta-learner is represented by “an.” The trainee model can learn how to produce data to make final predictions at the next level. Stacking provides multiple models, allowing for the selection of different sub-learners to learn the best prediction combination. This technique is sometimes referred to as “blending” because the model of meta blends these predictions from the subsystems, as illustrated in Figure 3. Algorithm for Ensemble Classifiers with Stacking (PROPOSED).

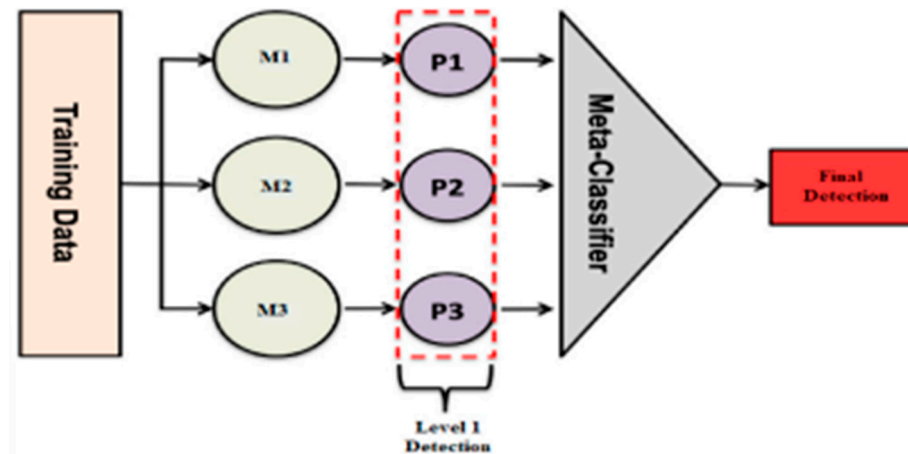


Figure 3. Ensemble classification process.

3.4. Genetic Algorithm

In the context of the paper, the Genetic Algorithm (GA) is utilized as an intelligent optimization algorithm to determine the optimal feature selection for feeding into the machine learning classifiers [24,25]. Optimization algorithms find wide applications in various fields [26,27]. Optimization algorithms have been used in applications ranging from structural [28] to blockchain [29] to predictive modelling [30].

The objective is to identify and inhibit bot threats and improve the performance of botnet detection. The GA is a metaheuristic algorithm inspired by natural selection and evolution [31]. It involves using a population of candidate solutions (in this case, features) that undergo selection, crossover, and mutation operations to evolve towards an optimal solution. The GA explores the solution space by iteratively evaluating and evolving candidate solutions based on their fitness or performance. In the suggested methodology, the GA is employed with the Particle Swarm Optimization (PSO) algorithm to establish the ideal order of features to use in the Kernel-based Ensemble Meta Classifier (KEMC) Strategy. The GA helps search for the most informative and relevant features by iteratively evaluating different combinations of features and selecting the ones that contribute the most to the performance of botnet detection.

By leveraging the GA’s ability to explore and exploit the search space efficiently, the model aims to identify the optimal subset of features that can effectively discriminate between botnet and legitimate network traffic. This feature selection process helps improve the accuracy and effectiveness of the machine learning classifiers used for botnet detection.

Hence, the GA plays a crucial role in the suggested methodology by intelligently optimizing the feature selection process, enabling the identification and inhibition of bot threats through improved botnet detection. The genetic algorithm draws inspiration from Darwin’s theory of natural evolution. It was initially proposed by Professor Holland in 1975 and is based on natural selection principles such as crossover, selection, and mutation. The algorithm associates each genetic chromosome with a potential solution to the problem and evolves the best model over multiple generations of reproduction. Biological evolutionary algorithms include evolutionary strategies, genetic algorithms and rules derived from other AI research projects. The first generation is created using the genetic algorithm’s code, and the process of individual evolution depends on the individual’s adaptation to the

environment, involving three fundamental processes: selection, crossing, and mutation. These processes aim for the achievement of the survival of the fittest individuals.

Selection: The term “selection” refers to the process of selection of the best individuals from a group while excluding the inferior ones. This process is based on evaluating their fitness. Individuals with a higher quality level are considered fitter and, therefore, more likely to be selected. They are also more likely to have offspring in the next generation. The individuals who are considered to be of high quality are selected to move to the next level based on their fitness values. Therefore, the stronger an individual’s ability to reproduce is, the higher the number of offspring that will carry its genes.

Crossing: The next step in the genetic algorithm process is crossing, where two high-quality individuals are randomly chosen to produce offspring at a specific point. This crossing process is essential to increase the searchability of the algorithm. Additionally, a mutation occurs when a few genes are altered randomly to improve the population in diversity. The offspring inherit superior traits from their parents, making them theoretically better suited to the environment. The algorithm eliminates individuals with a low fitness value after each iteration and keeps those with a high fitness value. The offspring continue to cycle this way until the global unique solution for the fitness method is discovered. Genetic algorithms are widely used in various fields, such as structural health monitoring, materialist identification, and data mining, and their potential applications will continue to expand as the technology develops.

3.5. PSO with GA

Swarm intelligence optimization has become a popular computer technique for solving complex distributed problems except for global models and centralized control. It involves using the swarm intelligence traits, such as collaboration, distribution, speed, and resilience. Particle swarm optimization (PSO) is an evolutionary algorithm that was first proposed in 1995 and simulated the social behaviour of a biological population by distributing individual information across groups to direct the population towards the desired direction. Unlike genetic algorithms, PSO treats each member of the population as a particle that moves across the search space at a fixed speed, modifying it dynamically based on its own flight experience and that of its friends. PSO has been effectively used in various domains, including production scheduling, multi-objective optimization, system decision-making, fuzzy control, and system identification [32]. PSO uses a positive feedback loop for collective optimization, gradually moving towards a better area based on each particle’s fit with its surroundings before finding the ideal solution to the problem. Particle position and velocity are properties that particles modify according to a formula using individual extremum $pbest_i$ and group extremum $gbest_i$ to change location and speed.

$$V_i = v_i + c_1 * rand() * (pbest_i - x_i) + c_2 * rand() * (gbest_i - x_i) \quad (15)$$

$$X_i = x_i + v_i \quad (16)$$

This equation involves particles indexed from $i = 1$ to n , where n represents the total number of particles. The particle’s velocity is represented by v_i , while $rand()$ signifies a random number between zero and one. The particle’s current position is represented by x_i , and c_1 and c_2 represent the learning factor. In Equation (15), V_i represents the updated velocity of particle i , v_i is the current velocity of particle i , c_1 and c_2 are acceleration coefficients, $rand()$ generates a random number between 0 and 1, $pbest_i$ is the personal best position of particle i , and $gbest_i$ is the global best position among all particles.

Equation (16) represents the updated position of particle i , where X_i is the updated position, x_i is the current position, and v_i is the updated velocity calculated in the previous equation.

4. Results and Discussion

All the experimentations are carried out in Python 3.11.2 on a Windows system with an Intel (R) Core (TM) i7-3770 CPU 3.40 GHz and 16 GB of RAM.

4.1. Performance Evaluation

The experimental results were evaluated using several criteria: precision, sensitivity, specificity, and F-measure. A comparison was made between the proposed (Ensemble Classifier Algorithm Stacking Process) and two state-of-the-art techniques, namely Support Vector Machine (SVM), Extreme Learning Machine (ELM) and Convolutional Neural Network (CNN). Table 1 presents the statistically significant characteristics identified during a network of two users having a connection. The false non-negative rate, accuracy (Acc), true positive rate (TPR)& f-measure were the performance metrics that were considered for the supervision applications. Botnets like Neris, Rbot, Vitut, Menti, and Sogou were considered for this purpose. The definitions for these requirements were established as follows:

TP indicates an accurate botnet attack prediction.

TN indicates the standard data’s right prediction.

FP specifies the wrong categorization of a botnet assault.

FN specifies the inaccurate classification of normal data.

Table 1. Description of the feature set.

Attributes	Data Types	No. of Attributes	Desc
No. of Packet	Flt	6	No. of pkt transmitted
Bytes	Flt	8	Entire bytes received
Packet size sMean	Str	5	Stream pkt send from src to target
Packet size dMax	Flt	7	Max pkt send from src to target
Average CONV	Int	6	Periods of CONV
No. of DASQry	Flt	4	DNS query
DNS Ratio	Str	5	DNS pkt
Rep.TCP count	Int	5	TCP comm. Between 2 network

Accuracy (Acc): This is the ratio of all datasets correctly identified for bugging and debugging to all bugging and debugging reports.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{17}$$

Sensitivity (Sc), which is determined according to Equation (18), is the proportion of negative cases that are wrongly categorized.

$$Sc = \frac{FP}{FP + TN} \tag{18}$$

The ratio of non-negative examples that are correctly identified is known as specificity (Sp), and it is calculated using Equation (19).

$$Sp = \frac{TP}{TP + FN} \tag{19}$$

The F – measure in Equation (20) is created by merging precision and recall into a single metric.

$$F - \text{measure} = \frac{(1 + \beta)TP}{\beta(TP(TP + FN) + TP(TP + FP))} \tag{20}$$

The accuracy comparison of the existing CNN, ELM, SVM, and suggested PROPOSED algorithms is shown in Table 2. The comparison of accuracy between the proposed method and the current ELM, CNN, and SVM methods is presented in Figure 4. The number of

datasets analyzed is displayed on the X-axis, while the corresponding accuracy percentages are shown on Y-axis (Figure 5). The proposed method achieved an accuracy score of 94.08%, which is 3.68% better than the current ELM, CNN, and SVM methods that achieved 91.6% -> ELM, 92.56% -> CNN, and 92.56% -> SVM, respectively, as indicated by in Table 1. Table 3 compares the sensitivity achieved by the proposed approach with existing methods such as CNN, ELM, and SVM.

Table 2. Evaluation of accuracy.

No. of Packets	ELMs	CNNs	SVMs	PROPOSEDs
50	88.3	90.3	90.4	92.3
100	90.6	90.6	91.7	94.6
150	93.6	93.6	94.6	95.6
200	93.9	94.9	96.2	95.3
250	95	95.6	97	97.3

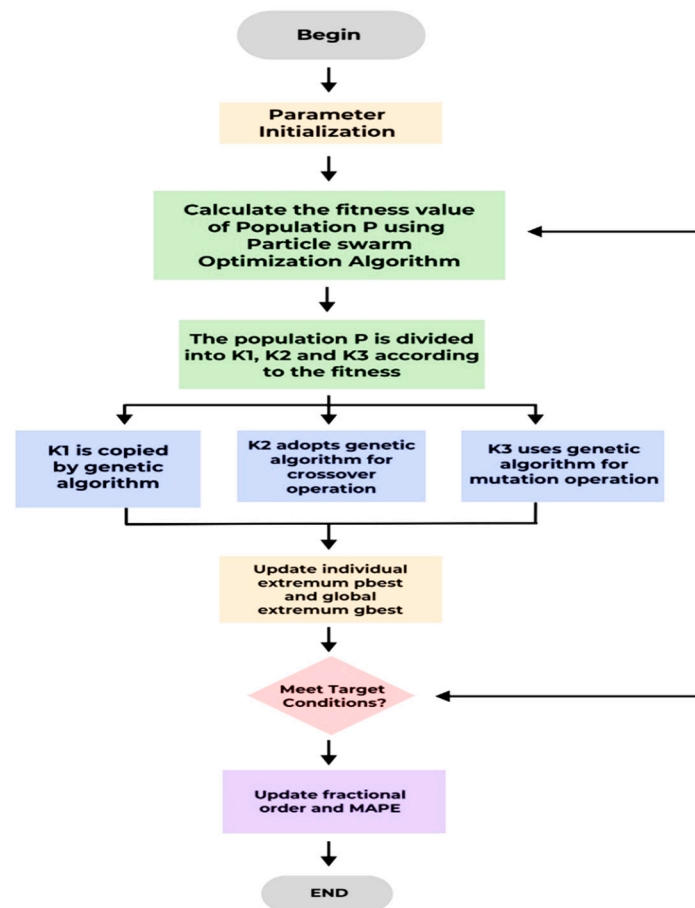


Figure 4. Structure of Genetic Techniques.

Table 3. Evaluation of sensitivity.

No. of Packets	ELMs	CNNs	SVMs	PROPOSEDs
50	79.6	79.9	81.2	83.4
100	81.6	82.3	84.5	85.6
150	84.7	84.9	86.5	87.8
200	85.8	86.4	87.8	89.6
250	90.3	89.8	91.2	91.6

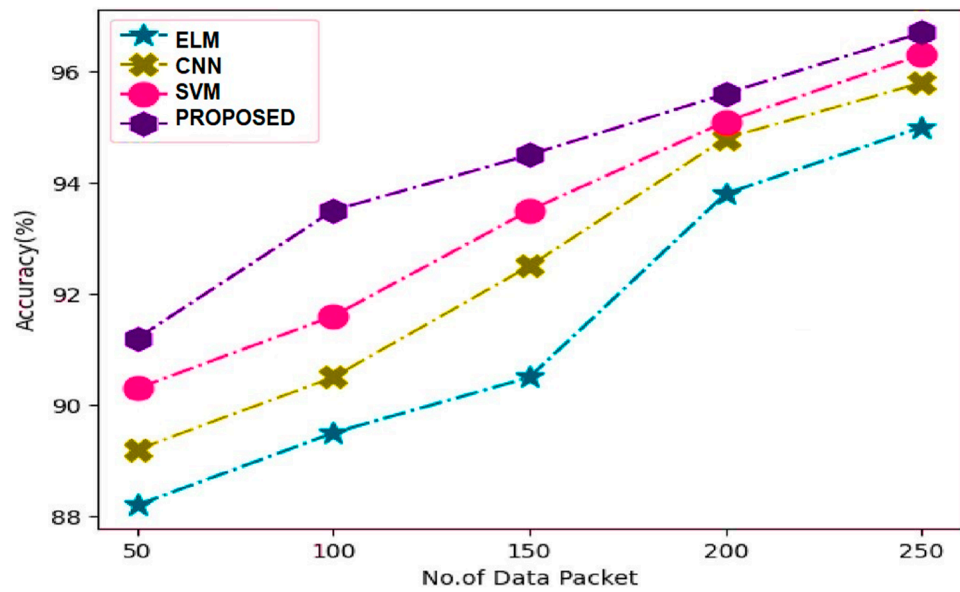


Figure 5. Comparison of accuracy.

Moreover, Figure 6 visually represents the sensitivity comparison between the PROPOSED approach and the CNN, ELM, and SVM methods. The number of datasets analysed (X-axis) versus % of sensitivity values computed (Y-axis). The PROPOSED approach scored a sensitivity of 86.5%, outperforming ELM by 3.44%, CNN by 3.3%, and SVM by 1.44%. In contrast, Table 4 shows the specificity comparison between PROPOSED and existing algorithms, where CNN, ELM, and SVM techniques achieved 83.14%, 83.8%, and 85.14%, respectively.

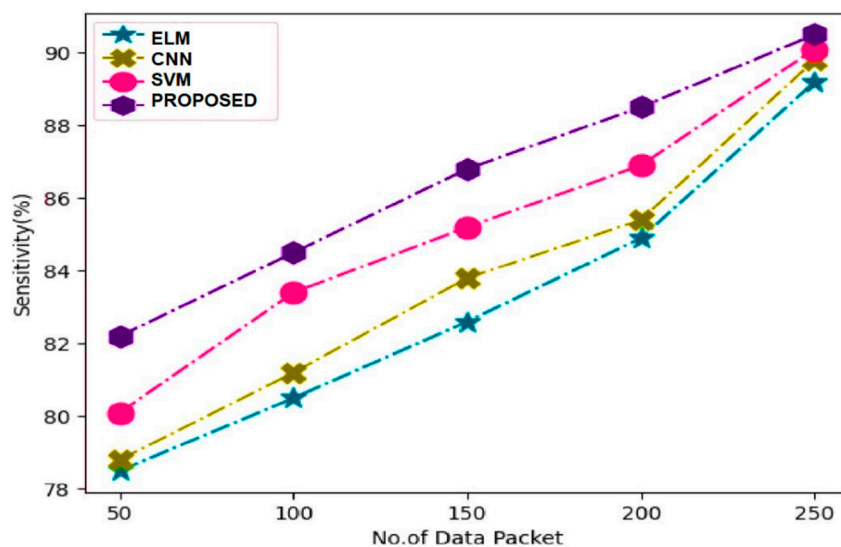


Figure 6. Comparison of sensitivity.

Table 4. Evaluation of specificity.

No. of Packets	ELMs	CNNs	SVMs	PROPOSEDs
50	80.3	82.2	83.4	82.4
100	83.6	84.5	85.6	87.5
150	85.7	85.6	87.8	86.3
200	86.8	87.8	88.6	86.5
250	89.9	88.9	80.2	89.4

Existing CNN, ELM, and SVM approaches are compared to the newly proposed method using the F-based metric, and the results are shown in Table 5. To the contrary, Figure 7 displays the percentage values of the F-measure on the Y-axis, and analytical datasets utilised on the X-axis. The PROPOSED method achieved the highest F-measure of 86.6%, outperforming ELM by 3.18%, CNN by 3.02%, and SVM by 1.86%. However, the existing CNN, ELM, and SVM methods scored 83.42%, 83.58%, and 84.74%, respectively.

Table 5. Evaluation of F-measure.

No. of Packets	ELMs	CNNs	SVMs	PROPOSEDs
50	68.2	69.6	69.9	70.5
100	72.6	73.6	74.7	76.8
150	76.8	77.5	78.6	78.5
200	80.3	81.6	82.6	83.7
250	83.5	85.9	86.2	86

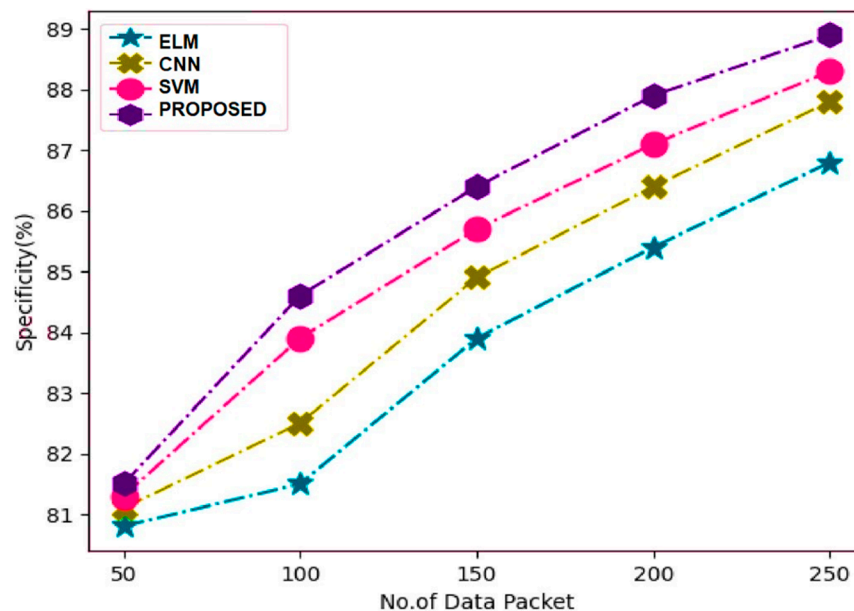


Figure 7. Comparison of specificity.

Figure 8 compares the F-based measure percentage values obtained by the newly proposed method and the existing CNN, ELM, and SVM methods, where the X-axis shows the number of datasets analyzed, and the Y-axis shows the percentages generated by the F-test. The PROPOSED method outperforms ELM, CNN, and SVM, achieving an F-measure percentage of 78.24%, 3% higher than ELM, 2.4% better than CNN, and 1.02% better than the SVM. In contrast, the existing CNN, ELM, and SVM methods attain F-measure percentages of 75.34%, 76.54%, and 77.32%, respectively. Additionally, Table 6 compares various parameters between the proposed method and the existing ELM, CNN, and SVM algorithms. The use of mathematical techniques in optimization technology is a practical application that is widely used across various fields of engineering to handle diverse challenges. Developing intelligent optimization techniques suitable for real-world engineering problems has long been an essential area of research due to the complexity and nonlinearity of these problems. Swarm intelligence is a kind of heuristic searching algorithm derived from genetic studies or observing the social behaviour of small creatures such as ants and bees. In swarm intelligence algorithms, individuals are represented by particles that move and interact with each other according to a set of rules. This approach is particularly useful in solving nonconvex, nonlinear, or non-differentiable optimization problems where the organizational structure of the method is not critical.

Swarm intelligence algorithms, which are commonly used, encompass particle swarm optimization and genetic algorithms are two examples of such methods.

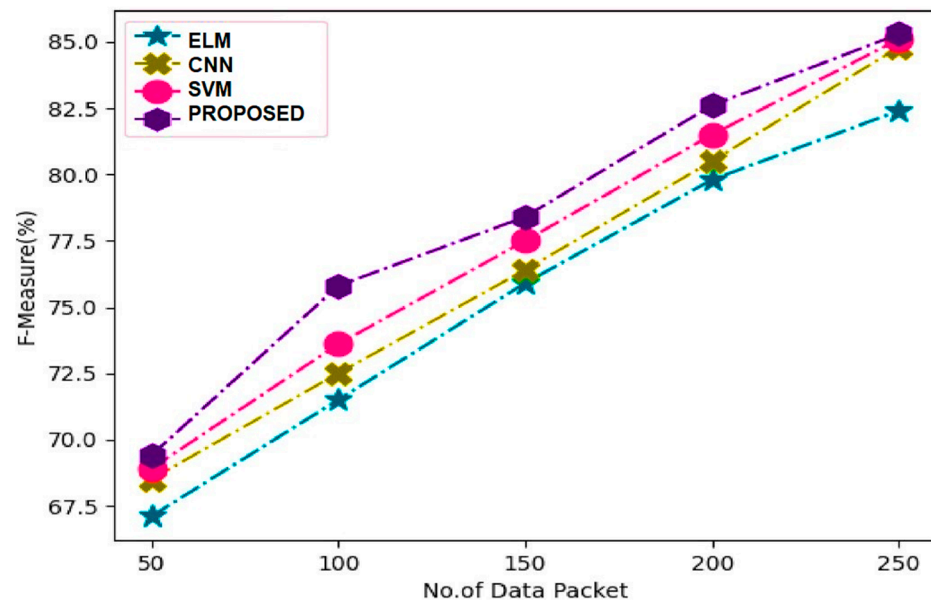


Figure 8. Comparison of F-measure.

Table 6. Simulation factors of configuration thresholds.

Name of Factor	Ranges
No. of layer	1–6
No. of Hidden layer	1–4
Learning Rate	10^{-1} – 10^{-5}
Drop Out	0.2–0.5
Optimizers	SGD, Adam, Adamax, Nadam, Ptrl, RMSprop
Function of activation	Sigmoid, softsign, selu
Function of loss	MSE

The process of adjusting hyperparameters in PROPOSED is challenging but significant. It involves selecting the appropriate activation and optimization functions and determining the optimal model structure. In addition, diagnostic procedures such as overfitting-underfitting must be enabled, and early halting functionality and batch-epoch definition should be established during the learning process. Underfitting occurs when NNs have not been trained for a sufficient amount of time, or the training set is not significant enough to establish the proper relationship between input and output variables, while overfitting happens when the model is too tightly matched to a small number of data points, rendering it only applicable to the original dataset. To prevent overfitting, the dropout regularization method is used to miss some neurons’ connections when training the model randomly. This study utilised the Optuna open-source platform to optimize PROPOSED performance and produce more accurate predictions for Android malware. Various PROPOSED model structures were designed by Optuna, and their hyperparameters were adjusted to assess their prediction performance. The PROPOSED was trained on the dataset for 2000 epochs with internal parameters updated after every 25 records, and Optuna was set to perform PROPOSED model optimization and evaluation 50 times. Each of the impacts of the hyperparameter on the PROPOSED system’s objective value (prediction accuracy) is depicted in Figure 5.

It is observed that the Dropout factor at the input nodes has the most significant impact (44%) on the model’s best value for the best prediction accuracy. The learning rate (lr) parameter has a 21% effect on the model’s best prediction. The number of units located

at the input nodes and the total number of layers contribute 20% and 15%, respectively, to the model’s best value. Figure 9 also provides a useful plot demonstrating the relationship between the different hyperparameter values and the best forecast result. This chart clarifies how the value of each hyperparameter is related to the PROPOSED model’s optimal value.

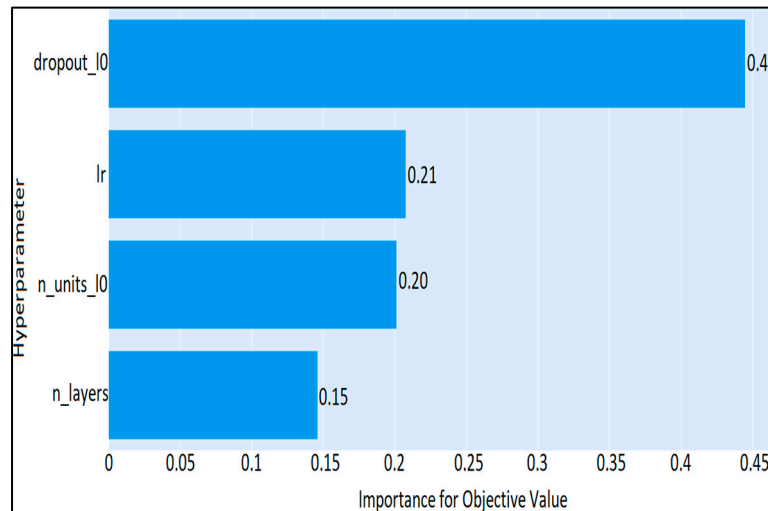


Figure 9. Relevance of hyperparameters to the suggested model.

Figure 10 displays the empirical distributions of the PROPOSED model. According to the confirmation performance analysis, the PROPOSED model outperformed the shallow are classified assessed in Section 4.2 in terms of performance when it came to predicting Android malware.

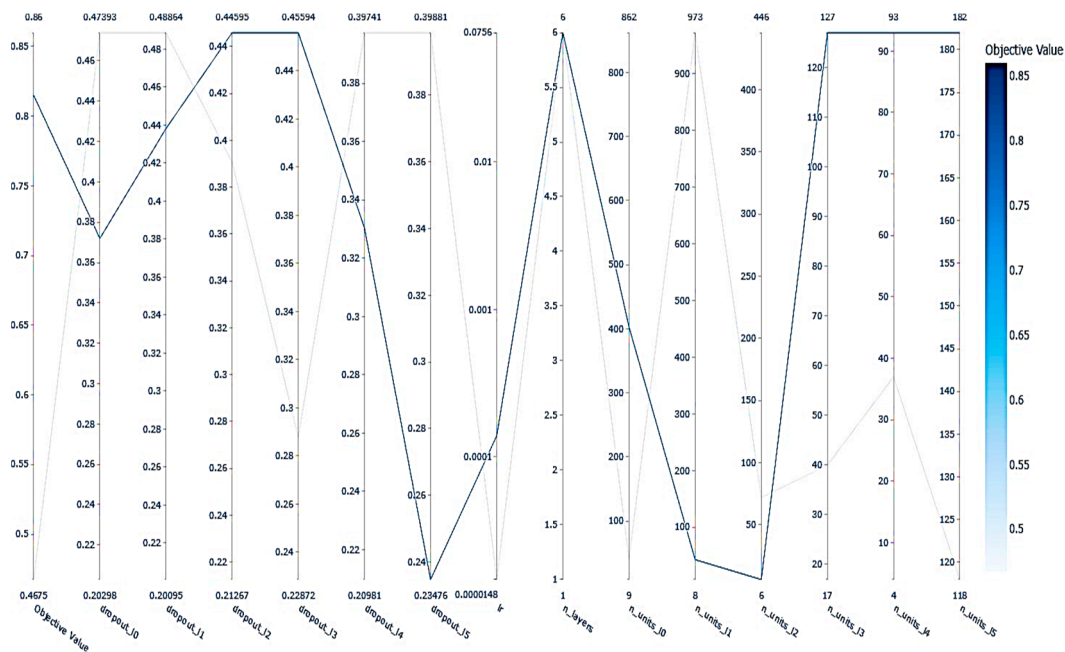


Figure 10. The suggested model’s hyperparameters are coordinated in a parallel scheme.

Figure 11 presents the remaining performance measures for the PROPOSED system. Figure 12a shows precision curves for learning and forecasting phases over the various epochs, whereas Figure 12b shows the cross-entropy (loss). Last but not just, Figure 12c provides the model’s confusion matrix. According to the evaluation results described above, the PROPOSED system with the best prediction results has a four-layer architecture, including two hidden layers. Figure 13 depicts the model’s abstract architectural perspective, and

Table 7 summarises each layer’s parameters. The Adamax optimizer, Binary Cross-Entropy (loss), and Softsign activation functions can all be used with the optimized model. Figure 14 displays the classifier’s history of optimization over the epochs. In particular, the model’s optimized prediction accuracy increased by 2% from the XGboost one to 86%.

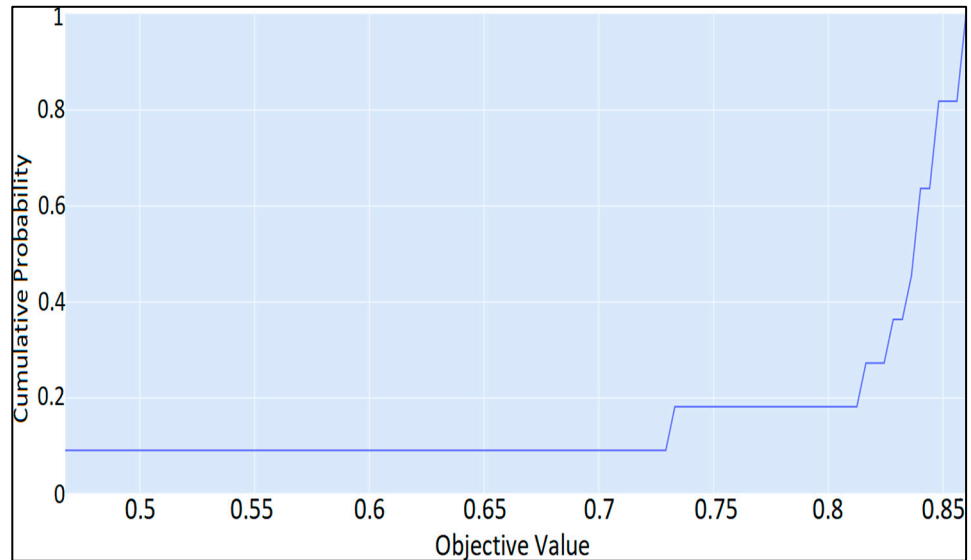


Figure 11. Probability distribution of the PROPOSED system.

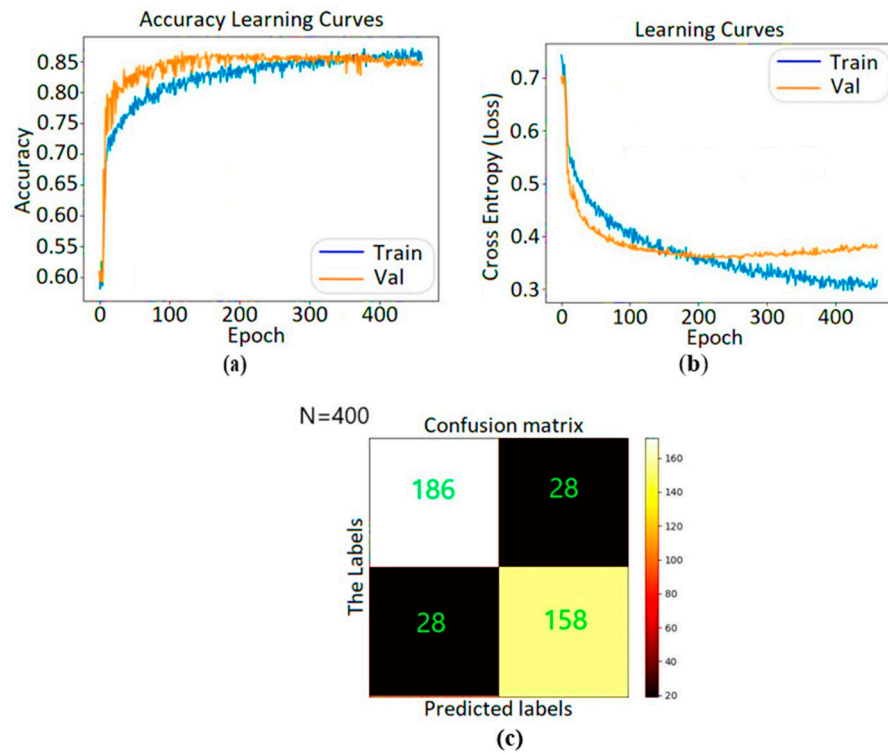


Figure 12. Analysis of the Proposed System’s Curve Performances (a) Accuracy (b) Cross Entropy (c) Confusion Matrix.

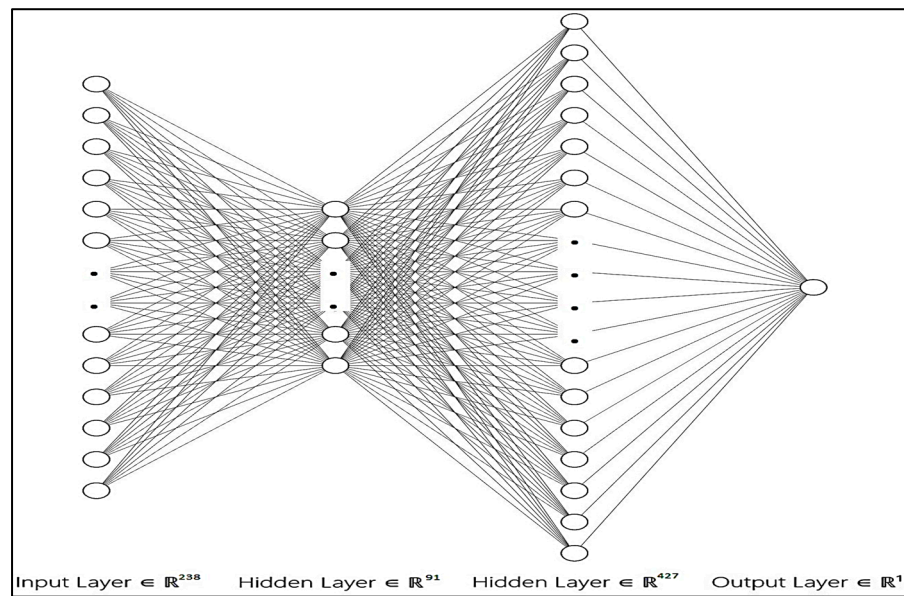


Figure 13. The proposed design has four levels: input (238) and hidden (427) and intermediate (91), and output (1).

Table 7. Total parameters of the PROPOSED.

Type of Layers	Shape of the Output Layer	Factors
Dense	(239, none)	238,953
1st dense	(92, none)	21,750
2nd dense	(428, none)	40,285
3rd dense	(1, none)	429

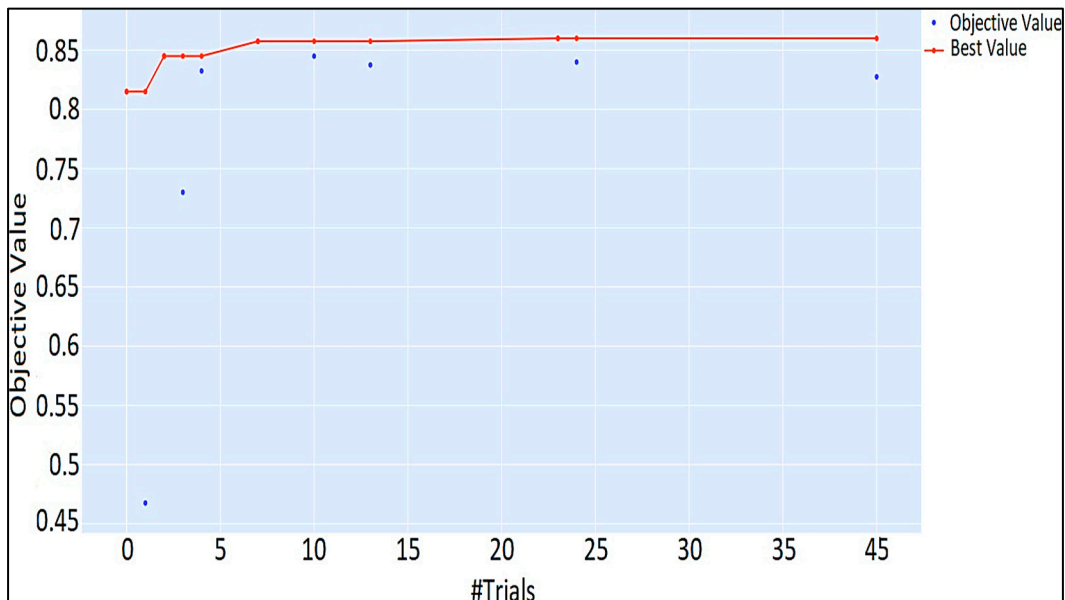


Figure 14. Evidence for the suggested model’s prior optimizations.

The proposed method consistently demonstrates superior performance, particularly in terms of accuracy, compared to ELMs, CNNs, and SVMs across various numbers of packets. With accuracy values ranging from 92.3% to 97.3%, the proposed method outperforms ELMs by 2 to 4 percentage points, CNNs by 0.4 to 4 percentage points, and SVMs by 0.3 to

2.9 percentage points. These results highlight the effectiveness of the proposed method in achieving higher accuracy rates, making it a promising approach for packet analysis tasks.

For 250 packets:

- Proposed accuracy: 97.3%
- ELMs accuracy: 95%
- CNNs accuracy: 95.6%
- SVMs accuracy: 97%

The proposed method exhibits superior performance compared to ELMs by 2.3 percentage points, CNNs by 1.7 percentage points, and SVMs by 0.3 percentage points in terms of accuracy.

4.2. Examination of SHAP- Shapley Additive Explanations Characteristics

Analyzing important features is crucial in machine learning (ML) models. This involves thoroughly evaluating and interpreting the model to determine the impact of input features on the system’s prediction. To gain a better understanding of the PROPOSED system and aid in this process, we utilized the SHAP unified framework [33]. Figures 15 and 16 illustrate the impact and contributions of the most significant input features to the system’s classification output. The factors with the most influence on the outcome are listed in descending order in Figure 15, with each input feature’s high and low values represented in red and blue text, respectively—for example, *com.Google.android.c1dm.intent.RECEIVE* appears to have the most significant influence on the result, where low values increase the probability of an application being malware, and high values decrease that likelihood. On the other hand, Figure 16 evaluates the average magnitude of influence on the system’s output for each input parameter, with only the features contributing to the system’s prediction displayed. The Intent. RECEIVE feature significantly affects the system’s output, with a SHAP value of more than 0.08 in both figures. Low values have a non-negative impact, making the system most accurate at around 0.1 SHAP value. However, maximum levels of the same features have a negative effect. As a result, projections of around -0.3 SHAP were off—this class. The DEFAULT feature affects the model’s output by 0.06%, where low amounts have a negative effect and high levels have an opposite effect at about 0.1. It is worth noting that only seven of the twenty features are Android permissions, with the rest being intents.

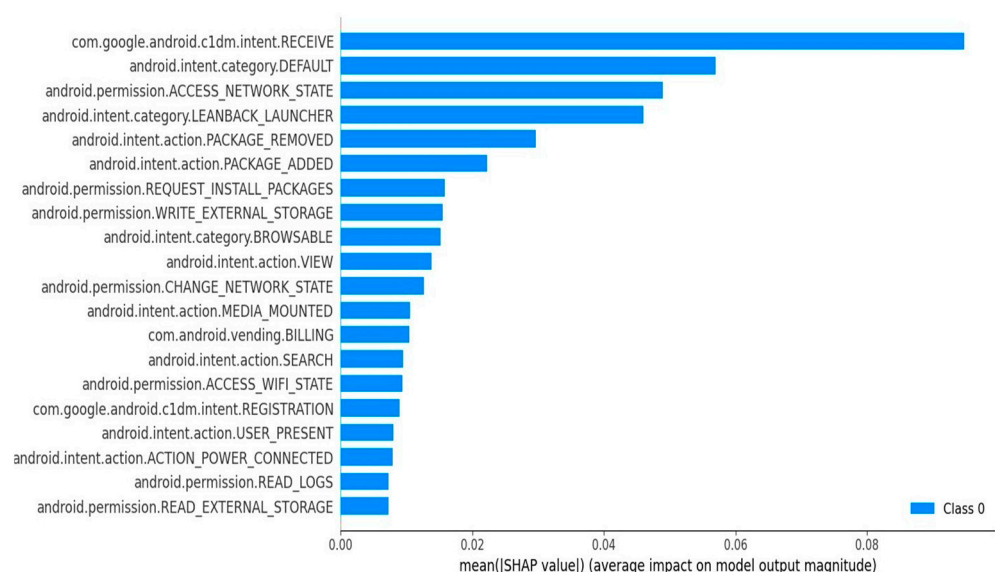


Figure 15. Features of the importance of the PROPOSED system.

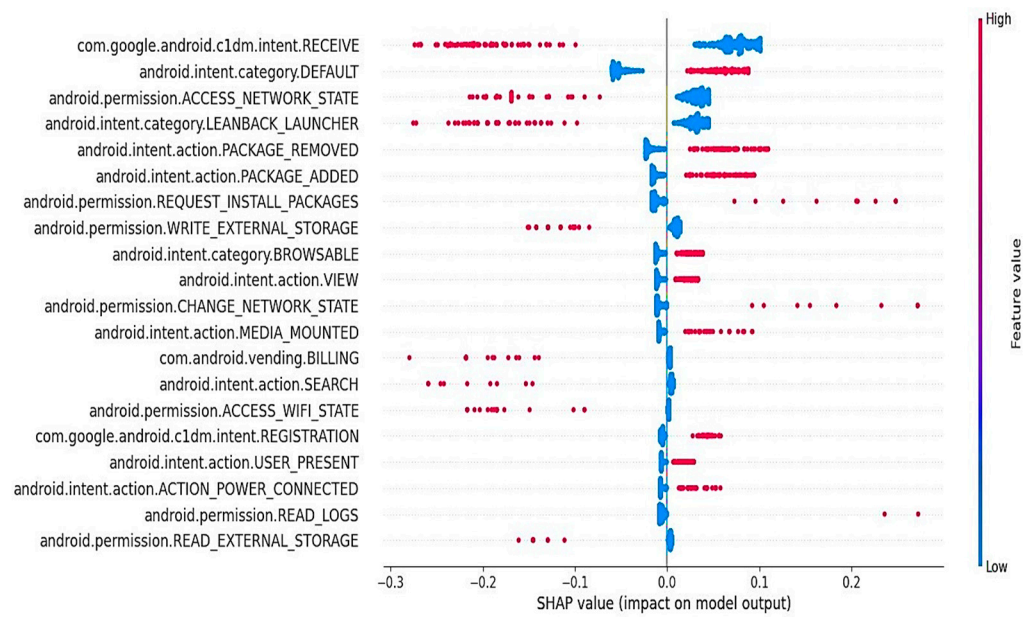


Figure 16. SHAP values for the features’ input in the PROPOSED system.

4.3. Discussion

The experimental results of the proposed Ensemble Classifier Algorithm Stacking Process were evaluated using various criteria such as precision, sensitivity, specificity, and F-measure. A comparison was made between the proposed method and three state-of-the-art techniques: Support Vector Machine (SVM), Extreme Learning Machine (ELM), and Convolutional Neural Network (CNN). The evaluation focused on the performance metrics relevant to botnet attack prediction, including false non-negative rate, accuracy (Acc), true positive rate (TPR), and F-measure. The accuracy comparison of the four algorithms is presented in Table 2. The proposed method achieved an accuracy score of 94.08%, outperforming the existing ELM, CNN, and SVM methods that achieved 91.6%, 92.56%, and 92.56%, respectively. This indicates the effectiveness of the proposed method in accurately identifying and categorizing botnet attacks.

Furthermore, the sensitivity evaluation in Table 3 demonstrates that the proposed approach achieved a sensitivity of 86.5%, surpassing the ELM by 3.44%, CNN by 3.3%, and SVM by 1.44%. The comparison in Table 4 shows that the proposed method achieved a specificity of 82.4%, while CNN, ELM, and SVM achieved 83.14%, 83.8%, and 85.14%, respectively. To assess the overall performance, the F-measure was used as a combined metric of precision and recall. The proposed method achieved the highest F-measure of 86.6%, surpassing ELM by 3.18%, CNN by 3.02%, and SVM by 1.86%. The existing CNN, ELM, and SVM methods achieved F-measure percentages of 75.34%, 76.54%, and 77.32%, respectively. These results highlight the superiority of the proposed method in accurately predicting and categorizing botnet attacks. The optimization of hyperparameters in the proposed method was crucial for achieving the best prediction accuracy. Factors such as the number of layers, hidden layers, learning rate, dropout, optimizers, activation, and loss functions were considered. The Optuna open-source platform was utilized to optimize the performance of the proposed method. Through optimization, the proposed model architecture with four layers, including two hidden layers, achieved the best prediction results. The Adamax optimizer, Binary Cross-Entropy (loss), and Softsign activation function were identified as the optimal choices for the model.

Furthermore, the importance of input features was analyzed using the SHAP (Shapley Additive Explanations) framework. The analysis revealed the significant impact of certain features on the system’s classification output—for example, the *com.Google.android.c1dm.intent.RECEIVE* feature had the most influence, where low values increased the probability of an application being classified as malware. The In-

tent.RECEIVE and class.DEFAULT features also exhibited notable influences on the system's output. Finally, the experimental results and evaluations demonstrate the effectiveness of the proposed Ensemble Classifier Algorithm Stacking Process in accurately predicting and categorizing botnet attacks. The proposed method outperformed the existing state-of-the-art accuracy, sensitivity, specificity, and F-measure techniques. The optimization of hyperparameters and the analysis of input feature importance further enhanced the performance and interpretability of the proposed method. These findings highlight the potential of the proposed method for effective botnet detection and classification in real-world applications. Future research can focus on expanding the evaluation to larger datasets and exploring the generalizability of the proposed method to other types of cybersecurity threats.

5. Conclusions

The effectiveness of the machine learning architecture is probed by changing the number of hidden layers and the number of neurons in the meta-ensembles. We built a model using machine learning to identify malware for the fileless. We could select the top-performing model and tweak its hyperparameters to improve its accuracy by training it on a fileless malware and benign samples dataset and then evaluating its performance on a testing dataset. Using publicly available datasets, we assess the efficacy of botnet detection. Less emphasis is placed on testing and evaluating network flow performance and determining whether a file has been subjected to a botnet attack. This study proposes combining the GA and PSO to the fractional order finding techniques for grayscale representations, which can then be used to develop an intelligent optimization-based fractional grey model. After putting this model through its paces, we found that it reliably forecasted future cyber security trends. The model needs more work before it can be used in other contexts. Methods for increasing the reliability of predictions could be investigated in subsequent studies. The primary focus of this publication was on short-term forecasting; hence, our future work will integrate algorithms based on deep learning that can be incorporated into this model to make predictions over the intermediate and long term.

Author Contributions: S.J.—conceptualization, methodology, formal analysis, investigation, writing—original draft, A.A.—conceptualization, methodology, formal analysis, investigation, writing—original draft, M.M.—conceptualization, methodology, resources, writing—review and editing, K.R.I.—conceptualization, formal analysis, investigation, writing—original draft, J.D.—methodology, validation, writing—original draft, P.J.—methodology, validation, writing—review and editing, M.E.—methodology, validation, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the project SP2023/074 Application of Machine and Process Control Advanced Methods supported by the Ministry of Education, Youth and Sports, Czech Republic.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available through email to the corresponding author upon request.

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

References

1. Etaher, N.; Weir, G.R.; Alazab, M. From Zeus to Zitmo: Trends in Banking Malware. In Proceedings of the 2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, 20–22 August 2015.
2. Kazi, M.A.; Woodhead, S.; Gan, D. Comparing the performance of supervised machine learning algorithms when used with a manual feature selection process to detect Zeus malware. *Int. J. Grid Util. Comput.* **2022**, *13*, 495. [[CrossRef](#)]
3. Sarojini, S.; Asha, S. Botnet detection on the analysis of Zeus panda financial botnet. *Int. J. Eng. Adv. Technol.* **2019**, *8*, 1972–1976. [[CrossRef](#)]
4. Aboaoja, F.A.; Zainal, A.; Ghaleb, F.A.; Al-Rimy, B.A.S.; Eisa, T.A.E.; Elnour, A.A.H. Malware Detection Issues, Challenges, and Future Directions: A Survey. *Appl. Sci.* **2022**, *12*, 8482. [[CrossRef](#)]

5. Owen, H.; Zarrin, J.; Pour, S.M. A survey on botnets, issues, threats, methods, detection and prevention. *J. Cybersecur. Priv.* **2022**, *2*, 74–88. [[CrossRef](#)]
6. Bukvić, L.; Škrinjar, J.P.; Fratrović, T.; Abramović, B. Price Prediction and Classification of Used-Vehicles Using Supervised Machine Learning. *Sustainability* **2022**, *14*, 17034. [[CrossRef](#)]
7. Preethi, P.; Asokan, R. Modelling LSUTE: PKE Schemes for Safeguarding Electronic Healthcare Records over Cloud Communication Environment. *Wirel. Pers. Commun.* **2019**, *117*, 2695–2711. [[CrossRef](#)]
8. Preethi, P.; Asokan, R. A High Secure Medical Image Storing and Sharing in Cloud Environment Using Hex Code Cryptography Method—Secure Genius. *J. Med. Imaging Health Inform.* **2019**, *9*, 1337–1345. [[CrossRef](#)]
9. Wu, Z.; Cao, J.; Wang, Y.; Wang, Y.; Zhang, L.; Wu, J. hPSD: A Hybrid PU-Learning-Based Spammer Detection Model for Product Reviews. *IEEE Trans. Cybern.* **2020**, *50*, 1595–1606. [[CrossRef](#)]
10. Riccardi, M.; Di Pietro, R.; Palanques, M.; Vila, J.A. Titans’ revenge: Detecting Zeus via its own flaws. *Comput. Netw.* **2013**, *57*, 422–435. [[CrossRef](#)]
11. Andriess, D.; Rossow, C.; Stone-Gross, B.; Plohmann, D.; Bos, H. Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus. In Proceedings of the 2013 8th International Conference on Malicious and Unwanted Software: “The Americas” (MALWARE), Fajardo, PR, USA, 22–24 October 2013.
12. Li, B.; Zhou, X.; Ning, Z.; Guan, X.; Yiu, K.-F.C. Dynamic event-triggered security control for networked control systems with cyber-attacks: A model predictive control approach. *Inf. Sci.* **2022**, *612*, 384–398. [[CrossRef](#)]
13. Quadir, A.; Jaiswal, D.; Daftari, J.; Haneef, S.; Iwendi, C.; Jain, S.K. Efficient Dynamic Phishing Safeguard System Using Neural Boost Phishing Protection. *Electronics* **2022**, *11*, 3133. [[CrossRef](#)]
14. Soniya, B.; Wilscy, M. Detection of randomized bot command and control traffic on an end-point host. *Alex. Eng. J.* **2016**, *55*, 2771–2781. [[CrossRef](#)]
15. Cheng, B.; Zhu, D.; Zhao, S.; Chen, J. Situation-Aware IoT Service Coordination Using the Event-Driven SOA Paradigm. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 349–361. [[CrossRef](#)]
16. Jiang, H.; Wang, M.; Zhao, P.; Xiao, Z.; Dustdar, S. A Utility-Aware General Framework with Quantifiable Privacy Preservation for Destination Prediction in LBSs. *IEEE/ACM Trans. Netw.* **2021**, *29*, 2228–2241. [[CrossRef](#)]
17. Yao, Y.; Zhao, J.; Li, Z.; Cheng, X.; Wu, L. Jamming and Eavesdropping Defense Scheme Based on Deep Reinforcement Learning in Autonomous Vehicle Networks. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 1211–1224. [[CrossRef](#)]
18. Thorat, S.A.; Khandelwal, A.K.; Bruhadeshwar, B.; Kishore, K. Payload Content based Network Anomaly Detection. In Proceedings of the 2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), Ostrava, Czech Republic, 4–6 August 2008.
19. Chen, P.; Liu, H.; Xin, R.; Carval, T.; Zhao, J.; Xia, Y.; Zhao, Z. Effectively Detecting Operational Anomalies in Large-Scale IoT Data Infrastructures by Using a GAN-Based Predictive Model. *Comput. J.* **2022**, *65*, 2909–2925. [[CrossRef](#)]
20. Guan, Z.; Jing, J.; Deng, X.; Xu, M.; Jiang, L.; Zhang, Z.; Li, Y. DeepMIH: Deep Invertible Network for Multiple Image Hiding. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *45*, 372–390. [[CrossRef](#)]
21. Azab, A.; Alazab, M.; Aiash, M. Machine Learning Based Botnet Identification Traffic. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016. [[CrossRef](#)]
22. Venkatesh, K.; Nadarajan, R.A. HTTP Botnet Detection Using Adaptive Learning Rate Multilayer Feed-Forward Neural Network. In *Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 38–48. [[CrossRef](#)]
23. Liu, X.; Shi, T.; Zhou, G.; Liu, M.; Yin, Z.; Yin, L.; Zheng, W. Emotion classification for short texts: An improved multi-label method. *Humanit. Soc. Sci. Commun.* **2023**, *10*, 306. [[CrossRef](#)]
24. Ganesh, N.; Shankar, R.; Čep, R.; Chakraborty, S.; Kalita, K. Efficient Feature Selection Using Weighted Superposition Attraction Optimization Algorithm. *Appl. Sci.* **2023**, *13*, 3223. [[CrossRef](#)]
25. Rajendran, S.; Ganesh, Čep, R.; Narayanan; Pal, S.; Kalita, K. A conceptual comparison of six nature-inspired metaheuristic algorithms in process optimization. *Processes* **2022**, *10*, 197. [[CrossRef](#)]
26. Eslami, M.; Neshat, M.; Khalid, S.A. A Novel Hybrid Sine Cosine Algorithm and Pattern Search for Optimal Coordination of Power System Damping Controllers. *Sustainability* **2022**, *14*, 541. [[CrossRef](#)]
27. Khajehzadeh, M.; Taha, M.R.; Eslami, M. Efficient gravitational search algorithm for optimum design of retaining walls. *Struct. Eng. Mech.* **2013**, *45*, 111–127. [[CrossRef](#)]
28. Kalita, K.; Dey, P.; Haldar, S.; Gao, X.-Z. Optimizing frequencies of skew composite laminates with metaheuristic algorithms. *Eng. Comput.* **2020**, *36*, 741–761. [[CrossRef](#)]
29. Cao, B.; Wang, X.; Zhang, W.; Song, H.; Lv, Z. A Many-Objective Optimization Model of Industrial Internet of Things Based on Private Blockchain. *IEEE Netw.* **2020**, *34*, 78–83. [[CrossRef](#)]
30. Li, B.; Tan, Y.; Wu, A.-G.; Duan, G.-R. A Distributionally Robust Optimization Based Method for Stochastic Model Predictive Control. *IEEE Trans. Autom. Control* **2022**, *67*, 5762–5776. [[CrossRef](#)]
31. Kalita, K.; Dey, P.; Haldar, S. Robust genetically optimized skew laminates. *Proc. Inst. Mech. Eng. Part C* **2019**, *233*, 146–159. [[CrossRef](#)]

32. Shankar, R.; Ganesh, N.; Čep, R.; Narayanan, R.C.; Pal, S.; Kalita, K. Hybridized Particle Swarm—Gravitational Search Algorithm for Process Optimization. *Processes* **2022**, *10*, 616. [[CrossRef](#)]
33. Gebreyesus, Y.; Dalton, D.; Nixon, S.; De Chiara, D.; Chinnici, M. Machine Learning for Data Center Optimizations: Feature Selection Using Shapley Additive exPlanation (SHAP). *Futur. Internet* **2023**, *15*, 88. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.