Received 18 June 2023, accepted 7 July 2023, date of publication 11 July 2023, date of current version 20 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3294263

RESEARCH ARTICLE

Automated Android Malware Detection Using Optimal Ensemble Learning Approach for Cybersecurity

HAYAM ALAMRO[1], WAFA MTOUAA[2], SUMAYH ALJAMEEL[3], AHMED S. SALAMA[4], MANAR AHMED HAMZA[5], AND ALADDIN YAHYA OTHMAN[5]

[1]Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh 11671, Saudi Arabia
[2]Department of Mathematics, Faculty of Sciences and Arts, King Khalid University, Muhayil 61421, Saudi Arabia
[3]Saudi Aramco Cybersecurity Chair, Department of Computer Science, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia
[4]Department of Electrical Engineering, Faculty of Engineering and Technology, Future University in Egypt, New Cairo 11845, Egypt
[5]Department of Computer and Self Development, Preparatory Year Deanship, Prince Sattam bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia

Corresponding author: Manar Ahmed Hamza (ma.hamza@psau.edu.sa)

The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through large group Research Project under grant number (RGP2/56/44). Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R361), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. We Would like to thank SAUDI ARAMCO Cybersecurity Chair for funding this project. This study is partially funded by the Future University in Egypt (FUE). This study is supported via funding from Prince Sattam bin Abdulaziz University project number (PSAU/2023/R/1444).

ABSTRACT Current technological advancement in computer systems has transformed the lives of humans from real to virtual environments. Malware is unnecessary software that is often utilized to launch cyber-attacks. Malware variants are still evolving by using advanced packing and obfuscation methods. These approaches make malware classification and detection more challenging. New techniques that are different from conventional systems should be utilized for effectively combating new malware variants. Machine learning (ML) methods are ineffective in identifying all complex and new malware variants. The deep learning (DL) method can be a promising solution to detect all malware variants. This paper presents an Automated Android Malware Detection using Optimal Ensemble Learning Approach for Cybersecurity (AAMD-OELAC) technique. The major aim of the AAMD-OELAC technique lies in the automated classification and identification of Android malware. To achieve this, the AAMD-OELAC technique performs data preprocessing at the preliminary stage. For the Android malware detection process, the AAMD-OELAC technique follows an ensemble learning process using three ML models, namely Least Square Support Vector Machine (LS-SVM), kernel extreme learning machine (KELM), and Regularized random vector functional link neural network (RRVFLN). Finally, the hunter-prey optimization (HPO) approach is exploited for the optimal parameter tuning of the three DL models, and it helps accomplish improved malware detection results. To denote the supremacy of the AAMD-OELAC method, a comprehensive experimental analysis is conducted. The simulation results portrayed the supremacy of the AAMD-OELAC technique over other existing approaches.

INDEX TERMS Cybersecurity, malware detection, ensemble learning, hunter prey optimization, machine learning.

I. INTRODUCTION

Cybersecurity is becoming a main area of immediate concern to network engineers and computer scientists, so satisfying solutions to several problems are in order [1]. Consequently, the fast technological developments and their inherent integrations in every aspect of lifestyles, various malware apps, and targets become well-identified and studied [2]. Android malware is the malware variety that gained significant interest in the web world. One common operating

The associate editor coordinating the review of this manuscript and approving it for publication was Binit Lukose.

system is Android, which dominates the operating system market [3].

Malware invasive methods emerge for avoiding identification, as few malware applications have more than 50 parameters that make detection a difficult one [4]. Hence, it is essential to devise techniques that deal with the continuous growth of Android malware to find it, deactivate or remove it efficiently. All these difficulties engage scholars in the area and urge them to continue more research to find malware and manage it properly [5]. Thus, researchers have developed three mechanisms to find Android malware such as dynamic, static, and hybrid analysis methods. Static analysis extracts the features that assist in identifying harmful performance for apps without a demanding actual application deployment [6]. But this kind of analysis suffered from code obfuscation methods which assist help malware author to avoid static methods. Dynamic analysis can be used for determining the malware of apps in their runtime [7]. Commonly, the static analysis feature offers the capability of locating the malware element using source code, while the dynamic analysis feature offers the capability of finding the location of malware in a runtime environment. Android developers and users can be exposed to unnecessary risks and dangers with malware [8]. This study covers malware detection methods. The detection of malware using the ML model includes Android Application Packages (APKs) for deriving an appropriate set of features. Deep learning (DL) and machine learning (ML) approaches can be used for recognizing malicious APKs [9]. Like malware detection, vulnerability detection in software code has two stages: training ML on derived attributes to find vulnerable code segments and feature generation utilizing code analysis [10].

This paper presents an Automated Android Malware Detection using Optimal Ensemble Learning Approach for Cybersecurity (AAMD-OELAC) technique. The AAMD-OELAC technique performs data preprocessing at the preliminary stage. For the Android malware detection process, the AAMD-OELAC technique follows an ensemble learning process using three ML models, namely Least Square Support Vector Machine (LS-SVM), kernel extreme learning machine (KELM), and Regularized random vector functional link neural network (RRVFLN). Finally, the hunter-prey optimization (HPO) algorithm is exploited for the optimal parameter tuning of the three DL models, and it helps accomplish improved malware detection results. To indicate the supremacy of the AAMD-OELAC approach, a comprehensive experimental analysis is carried out. In short, the key contributions are listed as follows.

- An intelligent AAMD-OELAC technique comprising data preprocessing, ensemble learning, and HPO-based hyperparameter tuning is presented for Android malware detection. To the best of our knowledge, the AAMD-OELAC technique never existed in the literature.
- Perform ensemble learning-based classification process comprising LS-SVM, KELM, and RRVFLN models for Android malware detection.
- The combination of the HPO algorithm and ensemble learning process improves the detection accuracy of Android malware. By utilizing multiple classifiers and optimization strategies, the model can effectively identify malicious patterns and behaviours in Android applications.

II. RELATED WORKS

Shaukat et al. [11] devise a new DL-related method for detecting malware. It delivered superior outcomes to classical methods by merging dynamic and static analysis benefits. Firstly, it visualizes a portable executable (PE) file as coloured images. Secondly, it extracted deep features from colour images utilizing fine-tuned DL method. Thirdly, it finds malware related to the deep features of SVM. Geremias et al. [12] presented a method using image-based DL called novel multi-view Android malware identification, applied threefold. Firstly, as per the many feature sets in multi-view settings, apps were assessed, thereby raising the data presented for the classification. Secondly, the derived feature set is transformed into image formats while preserving the essential elements of data distribution, keeping the data for the classifier task. Thirdly, built images are collectively depicted in one shot, all in a predefined image channel, allowing the implementation of DL structure.

Kim et al. [13] modelled a malware detection system called MAPAS that attains higher precision and adaptable use of computational resources. MAPAS examined the performances of malicious apps based on API call graphs of them through CNN. However, the presented MAPAS technique does not utilize a classifier method produced by CNN, it uses CNN to find typical attributes of the API call graph of malware. Fallah and Bidgoly [14] developed a technique related to LSTM for detecting malware-having the capability of differentiating benign and malware samples and identifying and detecting unseen and new types of malware. In this study, the author has executed many studies to show the abilities of the presented technique, including new malware family detection, malware identification, malware family identification, as well as assessing the minimal time needed to find malware.

Sihag et al. [15] introduced DL-based Android malware identification with the use of DYnamic features (De-LADY), a resilient obfuscation method. It has used behavioural features from dynamic analysis of an application performed in the emulated setting. Wang et al. [16] present a hybrid method related to DAE and CNN. Firstly, to enhance the precision of malware detection, the author reconstructed the high-dimensional feature of apps and used many CNN to find Android malware. Secondly, to diminish the training period, the author used DAE as a pre-training approach for CNN. With the consolidation, DAE and CNN method (DAE-CNN) can study flexible patterns quickly. Yadav et al. [17] presented a performance comparison of 26 existing pretrained CNN methods in Android malware detection. Depending on the outcomes, to find Android malware, an EfficientNet-B4 CNN-based approach was devised with the use of an image-based malware representation of the Android DEX file. From the malware images, EfficientNet-B4 extracted relevant attributes. Masum and Shahriar [18] devised a DL structure named Droid-NNet, for classifying malware. But this technique Droid-NNet, is a deep learner that surpasses existing cutting-edge ML approaches.

Idrees et al. [19] examine PIndroid – a new Permission and Intents-based structure to detect Android malware applications. As we know, PIndroid is the primary solution, which utilizes a group of permissions and purposes supplemented with Ensemble approaches for correct malware detection. In [20], the authors establish that once the concept drift was discussed, permissions create long-lasting and effectual malware detection methods. Taha and Barukab [21] introduce a mechanism for Android malware classification utilizing optimizer ensemble learning depending on GA. The GA was utilized for optimizing the parameter settings from the RF technique for obtaining the maximum Android malware classifier accuracy. Sabanci et al. [22] intended to categorize pepper seeds belonging to distinct cultivars with CNN techniques. Two methods are presented for classification. Initially, the CNN approaches (ResNet50 and ResNet18) are trained for pepper seeds. Secondary, diverse in the first, the features of pre-training CNN approaches are fused, and feature selection has been executed to the fused features. In [23], the authors examine recent algorithms utilized for Android Malware Detection. As a result, an outline of the Android system exposed the underlying processes and the problems facing its security structure.

III. THE PROPOSED MODEL

In this study, we have focused on designing the AAMD-OELAC technique for accurate and automated Android malware detection. The intention of the AAMD-OELAC method focused on the automatic recognition and classification of Android malware. To achieve this, the AAMD-OELAC technique encompasses data preprocessing, ensemble classification, and HPO-based parameter tuning. Fig. 1 exhibits the workflow of the AAMD-OELAC system.

A. DATA PREPROCESSING

At the initial stage, data preprocessing is performed to improve the quality of the actual data. As regards classification, it is vital for choosing features to signify the class the new record would concern [24]. Based on this, the permission and API calls can be eradicated in all Android applications, and both are features from the database. Androguard refers to a complete package tool planned for interacting with Android files and limited only to Python platforms. It could be utilized as a tool to reverse engineer single Android applications.

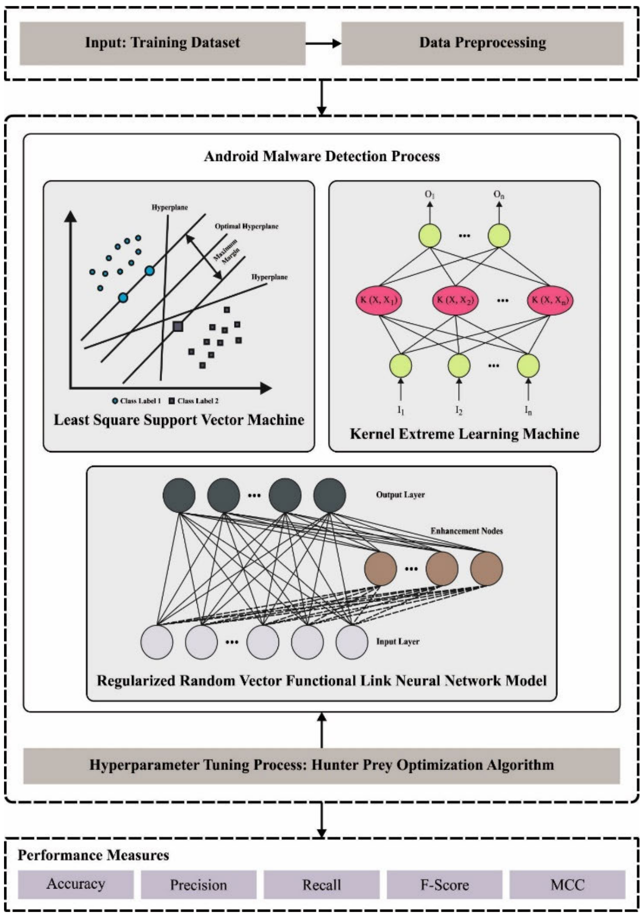

FIGURE 1. Workflow of AAMD-OELAC system.

The Androguard tool can be utilized for analyzing APK files by individually removing the DEX file permissions to all APK files. Hence, it is generated a data frame comprising applications (rows) and features (columns), whereas all the columns signify certain permission or API-call with binary values, but rows denote either malware or benign APK files.

B. ENSEMBLE LEARNING-BASED MALWARE CLASSIFICATION

For Android malware detection, the AAMD-OELAC technique follows an ensemble learning process using three ML models, namely LS-SVM, KELM, and RRVFLN.

1) LS-SVM MODEL

The LS-SVM is a revised edition of the SVM method. Also, it simplifies the computational procedure and minimizes computation costs [25]. Different from the SVM model, the LS-SVM model uses a sequence of linear equations for training:

$$Z = \alpha^T \varphi(x) + b + e_t \tag{1}$$

In Eq. (1): Z denotes the dependent parameter, x is the input, α indicates the weight coefficient, b shows the bias value, and $\varphi(x)$ indicates a non-linear mapping function.

Based on the optimization problem, the function estimation is defined as follows.

$$Minimize: \frac{1}{2}\alpha^T\alpha + \frac{1}{2}\gamma\sum_{t=1}^{M} e_t^2 \quad (2)$$

$$subjected\ (to)\ (f\ (x)) = \alpha^T\varphi\ (x) + b + e_t \quad (3)$$

where e denotes the error variable, $(f(x))$ shows the value of the dependent parameter and, γ shows the regulative constant, M symbolizes the number of datasets, T is a transpose. The Lagrange Multiplier removes the weight coefficient and the error variable. Afterwards resolving Eq. (3), we attain the subsequent matrix:

$$\begin{bmatrix} 0_{1\times n} & 1_{1\times n} \\ 0_{n\times 1} & P+\frac{1}{\gamma}\end{bmatrix}\begin{pmatrix} b \\ \beta\end{pmatrix} = \begin{pmatrix} 0 \\ z\end{pmatrix} \quad (4)$$

In Eq. (4), P indicates the kernel function, z represents the dependent variable, *and* β denotes the Lagrange multiplier:

$$f\ (x) = \sum_{t=1}^{M}\beta_t K\ (x, x_t) + b \quad (5)$$

2) KELM MODEL

ELM refers to a Feed Forward Neural Network (FWNN) that includes a single hidden layer. Compared to NN, in ELM, some variables are needed to train the model [26]. The weights and biases are not modified, whereas the hidden layer is required to be attuned. Based on these ELM characteristics, it has a faster convergence rate and better learning.

Assume training feature $\xi^*\in FV(V)$ and $\xi^* \in \{(f_i, y_j), i, j = 1, 2, 3, \ldots N$ where $f_i\in p_{i1}, f_2 l, \ldots f_N]$ characterize input feature vector and $y_j\in [y_{j1}, y_{j2}, \ldots, y_{jN}]$ signify the corresponding labels:

$$\Phi(f) = h(f)b = h\,(f)\,H^T\left(\frac{II}{C} + HH^T\right)^{-1}\hat{O}, \quad (6)$$

where H denotes an output matrix, $\hat{O}$ shows targeted output, $\frac{II}{c}$ denotes the kernel parameter, II represents the identity matrix, and C indicates the penalty parameter. Next, the kernel function of ELM is determined below:

$$K\tilde{ELM} = HH^T, \quad (7)$$

$$K\tilde{ELM} = h\,(f_i)\,h\,(f_j) = K\,(f_i, f_j)\,, \quad (8)$$

$$g\,(f) = \begin{bmatrix} K\,(f, f_i) \\ \cdot \\ \cdot \\ K\,(f, f_N)\end{bmatrix}\left(\left(\frac{II}{C} + K\tilde{ELM}\right)^{-1}\hat{O}\right), \quad (9)$$

Now $g(f)$ denotes the model function of ELM, and $\mathbb{K}(f, f_i)$ represents the kernel function of KELM. The kernel function can be determined by $K(f, f_i) = f\cdot f_i + \tilde{b}$. At last, the target labels $Y\in y$ and the error between the output $\hat{O}$ are evaluated as follows.

$$\sum_{j=1}^{N}\|(\hat{O} - y_j)\| = 0. \quad (10)$$

Now, the testing video is passed to the presented method, and in the output, labelled outcomes are produced (carrying a bag, normal walking, and wearing a coat). We obtain this image in the testing stage, where all the labels are allotted per the model training.

3) RRVFLN MODEL

As a kind of single-hidden layer neural network, RRVFLN integrates hidden and input layers to impact the output layer [27]. For a data sample with N input, the $i-th$ input sample is $x_i = [x_{i1}, x_{i2}, \ldots\ldots, x_{in}]^T\in R^n$, and the $i-th$ output sample is $y_i = [y_{i1}, y_{i2}, \ldots\ldots, y_{im}]^T\in R^m$. The RVFL network, the hidden layer node is L, is formulated as follows:

$$\sum_{j=1}^{L}\beta_j g\left(w_j x_i + b_j\right) + \sum_{j=L+1}^{L+d}\beta_j x_{ij} = 0_i, i = 1, 2, \ldots\ldots, N \quad (11)$$

where $w_j = [w_{j1}, w_{j2}, \ldots\ldots, w_{jn}]$ indicates the input weight, $g(x)$ denotes the activation function, b_j signifies the hidden layer enhancement node's bias, and $\beta_j = [\beta_{j1}, \beta_{j2}, \ldots\ldots, \beta_{jm}]^T$ denotes the outcome weight. w_j and b_j are commonly defined stochastically:

$$H\beta = O \quad (12)$$

where H signifies the hidden layer enhancement node's output, β characterizes the output weight, and O is the output result that can be extended below:

$$H = \begin{bmatrix} g(w_1x_1 + b_1) & \ldots & g(w_Lx_1 + b_L) & x_{11} & \ldots & x_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ g(w_1x_N + b_1) & \ldots & g(w_Lx_N + b_L) & x_{N1} & \ldots & x_{Nn}\end{bmatrix} \quad (13)$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \\ \beta_{L+1}^T \\ \vdots \\ \beta_{L+n}^T\end{bmatrix} \quad (14)$$

$$O = \begin{bmatrix} o_1 \\ \vdots \\ o_N\end{bmatrix} \quad (15)$$

In contrast with the traditional RRVFLN network, the standardized RRVFLN can effectively prevent overfitting problems and improve the network's normalisation capacity. Mostly, finds the output weights and the least training error:

$$L_{RVFL} = \frac{1}{2}CE_2^2 + \frac{1}{2}\beta_2^2, \quad (16)$$

In Eq. (16), the output error $E = O - Y$, C shows the regularization factor that can be utilized to trade off the effects between model complexity and training error. To ascertain the minimal value of L_{RVFL}, the closed-form solution of β is attained by fitting the grad of RRVFLN concerning β to 0:

$$\beta = \begin{cases} \left(H^T H + \frac{I}{C}\right)^{-1} H^T T & if\ N > L+n \\ H^T \left(\frac{I}{C} + H^T H\right)^{-1} T & if\ N < L+n' \end{cases} \tag{17}$$

In Eq. (17), I denote an identity matrix of dimension $L+n$.

C. PARAMETER TUNING USING HPO ALGORITHM

In this work, the HPO approach is exploited for the parameter tuning of the ML models. The HPO method is a recent optimization approach to resolving optimization problems [28]. This method is stimulated by the hunting strategy of a few carnivores, namely leopards, lions, and tigers, along with prey, including antelopes and deer. In the presented technique, the hunter generally gives importance to the prey distant from their group. The hunter modifies his location toward the prey distant from their group whereas the prey modifies their location towards the group. The position of the search agent can be regarded as a safer location using the optimal fitness function value. The hunter-prey optimizer steps are given below:

Like other optimization algorithms, in every iteration, the location of every prey and hunter is updated based on the rules, and the newest location of every member is *re*-evaluated through the main function. The location of all the members in the initial group can be generated randomly in the search space as follows:

$$x_j = rand\ (1,d) \times (ub - lb) + lb, \tag{18}$$

In Eq. (18), d indicates the number of parameters of the problem. x_i denotes the position of hunter, lb, and ub denotes the minimum and the maximum value for the problem variable

Next, the search module of the hunters usually includes two steps: exploitation and exploration. In exploration, the hunters tend to search the region of potential prey. Exploitation is employed afterwards a promising region is found, that is hunter must minimize the random behaviour to find prey around a potential region:

$$x\ (t+1) = x\ (t) + \frac{[(2CZP_{pos} - x(t)) + (2(1-C)Z\mu - x(t)]}{2}, \tag{19}$$

In Eq. (19), $x(t)$ and $x(t+1)$ denote the existing and the next location of the hunter, P_{pos} shows the location of prey, μ represents the mean of each position, C denotes the balance variable between exploitation and exploration, its value declining from 1 to 0.02, and Z denotes the adaptive parameter:

$$C = 1 - it\left(\frac{1-0.98}{\mathrm{Max}It}\right), \tag{20}$$

$$Z = \vec{R}_1 \otimes idx + \vec{R}_2 \otimes (\sim idx)\,, \tag{21}$$

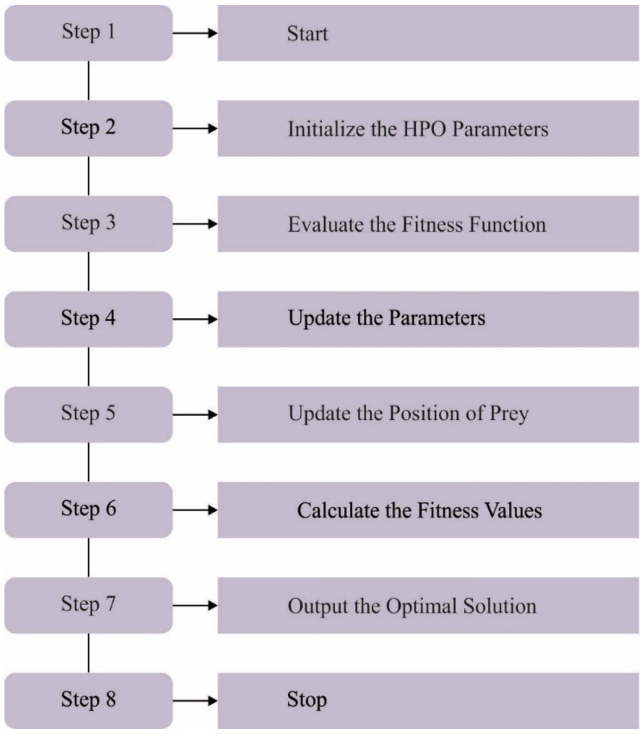
Step 1
Start
Step 2
Initialize the HPO Parameters
Step 3
Evaluate the Fitness Function
Step 4
Update the Parameters
Step 5
Update the Position of Prey
Step 6
Calculate the Fitness Values
Step 7
Output the Optimal Solution
Step 8
Stop

FIGURE 2. Flowchart of HPO algorithm.

where it and $MaxIt$ denote the exiting and maximum amount of iterations. $\vec{R}_1$ and $\vec{R}_2$ show random vectors within [0, 1], and idx indicates the vectors index number that fulfils the condition $\vec{R}_1 < C$. Next, based on the hunting scenario, once the hunter has taken the prey, it dies, and then, the hunter moves toward the position of the dead prey. The iteration continues with the location of prey being the newest hunter location. Furthermore, assuming that the safer position was the optimal global position, as it provides the prey with greater opportunities for survival, the hunter might select another prey, as follows:

$$x\ (t+1) = T_{pos} + CZ\cos\ (2\pi R_3) \times \left(T_{pos} - x\ (t)\right), \tag{22}$$

In Eq. (22), $x\ (t)$ and $x(t+1)$ shows the existing and next location of the hunter, T_{pos} denotes the global optimum location that includes the better fitness from the initial iteration to the existing iteration, and R_3 represent the random integer. Fig. 2 illustrates the flowchart of the HPO technique.

Lastly, it is significant how the hunter and prey are selected as follows:

$$x\ (t+1) = \begin{cases} x(t) + 0.5[(2CZP_{pos} - x(t)) + (2\,(1-C)\,Z\mu - x\ (t)] \\ \qquad R_4 < \beta \\ T_{pos} + CZ\cos\ (2\pi R_3) \times \left(T_{pos} - x\ (t)\right) \\ \qquad R_4 \geq \beta \end{cases} \tag{23}$$

In Eq. (23), R_4 shows the random integer within [0, 1], and β denotes the adjusting parameter.

TABLE 1. Details of database.

Class	No. of Samples
Benign	5000
Malware	2500
Total Samples	7500

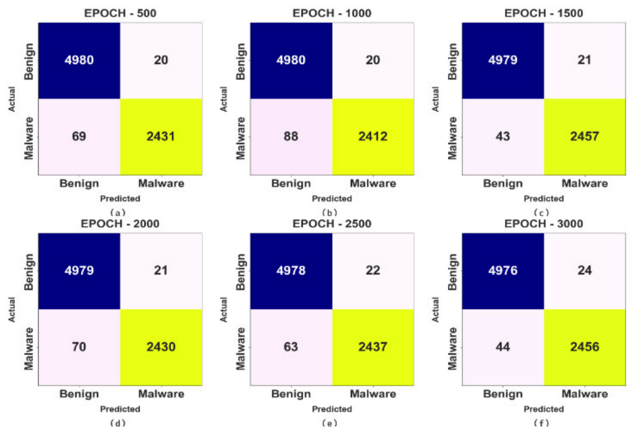

FIGURE 3. Confusion matrices of AAMD-OELAC system (a-f) Epochs 500-3000.

TABLE 2. Malware classifier outcome of AAMD-OELAC system with distinct epochs.

Class	$Accu_y$	$Prec_n$	$Reca_l$	F_{score}	MCC
Epoch - 500					
Benign	99.60	98.63	99.60	99.11	97.33
Malware	97.24	99.18	97.24	98.20	97.33
Average	98.42	98.91	98.42	98.66	97.33
Epoch - 1000					
Benign	99.60	98.26	99.60	98.93	96.76
Malware	96.48	99.18	96.48	97.81	96.76
Average	98.04	98.72	98.04	98.37	96.76
Epoch - 1500					
Benign	99.58	99.14	99.58	99.36	98.08
Malware	98.28	99.15	98.28	98.71	98.08
Average	98.93	99.15	98.93	99.04	98.08
Epoch - 2000					
Benign	99.58	98.61	99.58	99.09	97.27
Malware	97.20	99.14	97.20	98.16	97.27
Average	98.39	98.88	98.39	98.63	97.27
Epoch - 2500					
Benign	99.56	98.75	99.56	99.15	97.45
Malware	97.48	99.11	97.48	98.29	97.45
Average	98.52	98.93	98.52	98.72	97.45
Epoch - 3000					
Benign	99.52	99.12	99.52	99.32	97.96
Malware	98.24	99.03	98.24	98.63	97.96
Average	98.88	99.08	98.88	98.98	97.96
Malware	98.41	99.07	98.41	98.74	98.11
Average	98.97	99.13	98.97	99.05	98.11

The HPO method has derived a fitness function (FF) to get better classifier outcomes. It determined a positive value that represented the candidate solutions' superior outcome. The minimal classifier error rate can be the FF, as shown in Eq. (24).

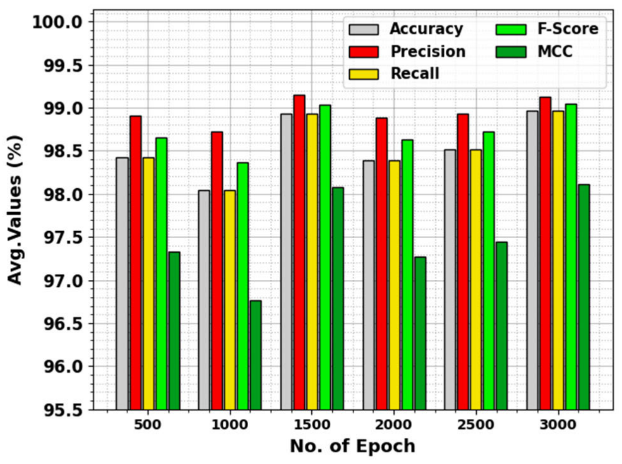

FIGURE 4. The average outcome of the AAMD-OELAC system with distinct epochs.

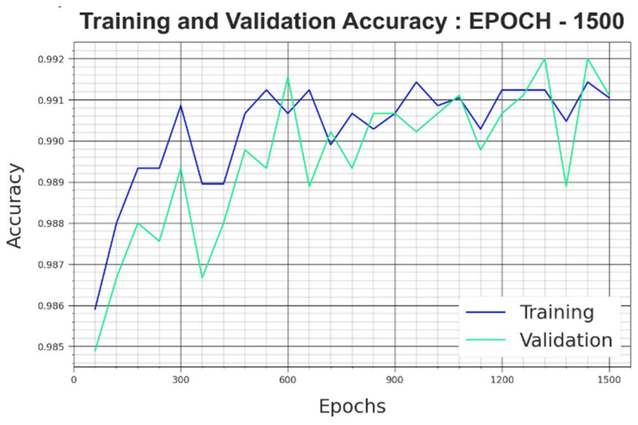

FIGURE 5. Accuracy curve of AAMD-OELAC system on epoch 1500.

$$\begin{aligned} fitness\,(x_i) &= ClassifierErrorRate\,(x_i) \\ &= \frac{number\ of\ misclassified\ samples}{Total\ number\ of\ samples} * 100 \end{aligned} \tag{24}$$

IV. PERFORMANCE VALIDATION

In this study, the malware detection outcomes of the AAMD-OELAC technique are examined on the Andro-AutoPsy Dataset [29], [30]. It holds 7500 samples with two classes, as given in Table 1.

The confusion matrices of the AAMD-OELAC method on the malware detection process are demonstrated in Fig. 3. The figure states that the AAMD-OELAC technique properly categorized the benign and malware samples.

The malware classification results of the AAMD-OELAC technique are investigated under several epochs in Table 2 and Fig. 4. The result identified that the AAMD-OELAC technique properly categorized benign and malware samples under all epochs. For example, with 500 epochs, the

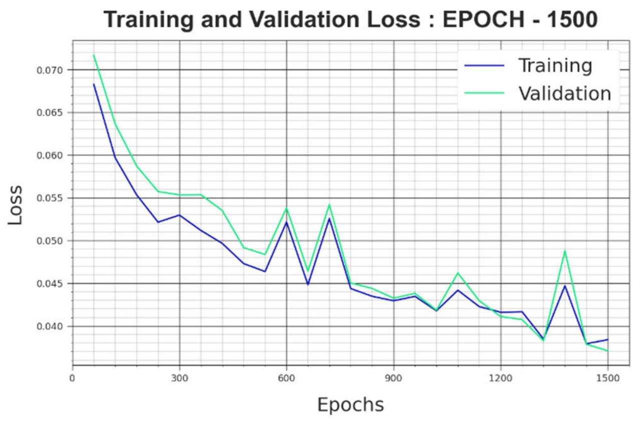

FIGURE 6. Loss curve of AAMD-OELAC system on epoch 1500.

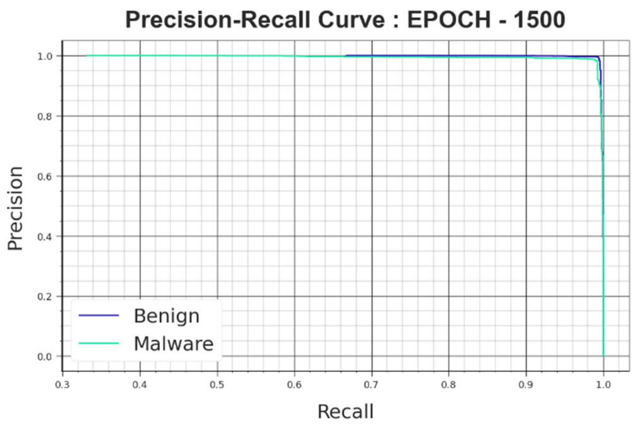

FIGURE 7. PR curve of AAMD-OELAC system on epoch 1500.

AAMD-OELAC method gains average $accu_y$ of 94.42%, $prec_n$ of 98.91%, $reca_l$ of 98.42%, F_{score} of 98.66%, and MCC of 97.33%. In the meantime, with 1500 epochs, the AAMD-OELAC approach gains average $accu_y$ of 98.93%, $prec_n$ of 99.15%, $reca_l$ of 98.93%, F_{score} of 99.04%, and MCC of 98.08%. Finally, with 2000 epochs, the AAMD-OELAC method gains average $accu_y$ of 98.39%, $prec_n$ of 98.88%, $reca_l$ of 98.39%, F_{score} of 98.63%, and MCC of 97.27%. At last, with 3000 epochs, the AAMD-OELAC approach gains average $accu_y$ of 98.97%, $prec_n$ of 99.13%, $reca_l$ of 98.97%, F_{score} of 99.05%, and MCC of 98.11%.

Fig. 5 exhibits the accuracy of the AAMD-OELAC approach in the process of the training and validation on epoch 1500. The figure specified that the AAMD-OELAC technique attained greater accuracy values over increasing epochs. Also, the higher validation accuracy over training accuracy depicted that the AAMD-OELAC approach learns productively on epoch 1500.

The loss analysis of the AAMD-OELAC method in the training and validation is shown on epoch 1500 in Fig. 6. The figure pointed out that the AAMD-OELAC method reached closer values of training and validation loss. The AAMD-OELAC method learns productively on epoch 1500.

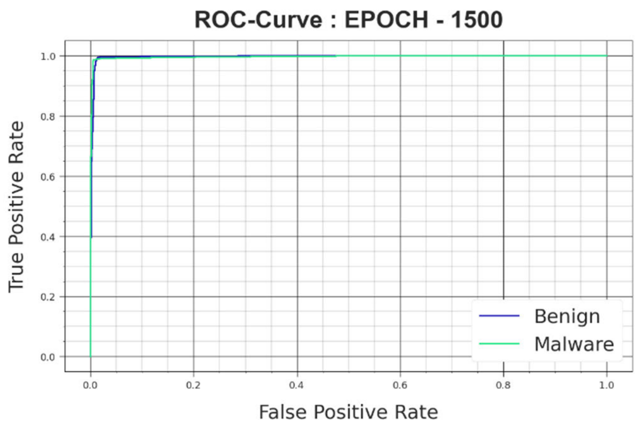

FIGURE 8. ROC curve of AAMD-OELAC system on epoch 1500.

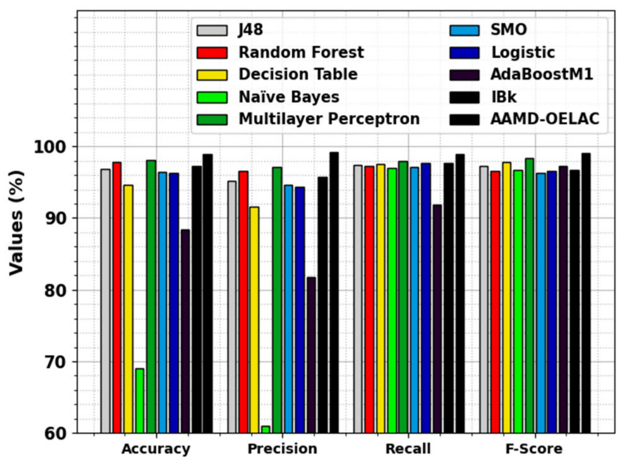

FIGURE 9. Comparative outcome of AAMD-OELAC approach with other methods.

TABLE 3. Comparative result of the AAMD-OELAC method with other approaches.

Algorithm	$Accu_y$	$Prec_n$	$Reca_l$	F_{score}
J48	96.80	95.20	97.42	97.24
Random Forest	97.80	96.60	97.22	96.62
Decision Table	94.60	91.60	97.52	97.77
Naïve Bayes	69.10	61.00	96.92	96.71
Multilayer Perceptron	98.10	97.10	97.90	98.35
SMO	96.40	94.60	97.07	96.26
Logistic	96.30	94.40	97.74	96.58
AdaBoostM1	88.40	81.70	91.83	94.25
IBk	97.20	95.70	97.66	96.68
AAMD-OELAC	98.93	99.15	98.93	99.04

A brief precision-recall (PR) curve of the AAMD-OELAC method is given on epoch 1500 in Fig. 7. The results pointed out that the AAMD-OELAC technique results in higher values of PR. Further, the AAMD-OELAC technique can reach higher PR values in all classes.

TABLE 4. CT outcome of AAMD-OELAC approach with other methods.

Algorithm	Time Costs (s)
J48	00.28
Random Forest	00.68
Decision Table	07.48
Naïve Bayes	00.28
Multilayer Perceptron	15.19
SMO	13.44
Logistic	00.95
AdaBoostM1	00.45
IBk	00.30
AAMD-OELAC	00.08

In Fig. 8, a ROC study of the AAMD-OELAC technique is revealed on epoch 1500. The figure described that the AAMD-OELAC algorithm resulted in improved ROC values. Furthermore, the AAMD-OELAC method can extend enhanced ROC values on all classes.

Table 3 and Fig. 9 compare the overall outcomes of the AAMD-OELAC method with other models. The outcomes identified that the NB approach has poor performance, whereas the AdaBoostM1 model gains slightly enhanced results. Along with that, the J48, RF, DT, MLP, SMO, logistic, and IBk models accomplish moderately closer performance. However, the AAMD-OELAC technique offers better results with increased accu_y of 98.93%, prec_n of 99.15%, reca_l of 98.93%, and F_score of 99.04%.

Finally, the computational time (CT) analysis of the AAMD-OELAC technique is compared with recent approaches in Table 4. The outcomes exhibited that the AAMD-OELAC technique reaches the least CT value of 8s. At the same time, the existing models have reached increased CT values. These results highlighted that the AAMD-OELAC technique shows maximum performance over other models on malware classification.

V. CONCLUSION

In this study, we have developed the design of the AAMD-OELAC technique for an accurate and automated Android malware detection process. The intention of the AAMD-OELAC approach focused on the automatic recognition and classification of Android malware. To achieve this, the AAMD-OELAC technique encompasses data preprocessing, ensemble classification, and HPO-based parameter tuning. For the Android malware detection process, the AAMD-OELAC technique follows an ensemble learning process using three ML models namely LS-SVM, KELM, and RRVFLN. Finally, the HPO algorithm is exploited for the optimal parameter tuning of the three DL models and it helps in accomplishing improved malware detection results. To portray the supremacy of the AAMD-OELAC method, a wide-ranging experimental analysis is conducted. The simulation results portrayed the supremacy of the AAMD-OELAC technique over other existing approaches. Future work could focus on developing more advanced techniques to capture and analyze fine-grained behaviours, enabling better detection of sophisticated malware. In addition, future work could explore privacy-preserving approaches such as secure multi-party computation or federated learning, which enable collaborative malware detection without compromising user privacy.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through large group Research Project under grant number (RGP2/56/44). Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R361), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. We Would like to thank SAUDI ARAMCO Cybersecurity Chair for funding this project. This study is partially funded by the Future University in Egypt (FUE). This study is supported via funding from Prince Sattam bin Abdulaziz University project number (PSAU/2023/R/1444).

REFERENCES

[1] H. Rathore, A. Nandanwar, S. K. Sahay, and M. Sewak, "Adversarial superiority in Android malware detection: Lessons from reinforcement learning based evasion attacks and defenses," *Forensic Sci. Int., Digit. Invest.*, vol. 44, Mar. 2023, Art. no. 301511.

[2] H. Wang, W. Zhang, and H. He, "You are what the permissions told me! Android malware detection based on hybrid tactics," *J. Inf. Secur. Appl.*, vol. 66, May 2022, Art. no. 103159.

[3] A. Albakri, F. Alhayan, N. Alturki, S. Ahamed, and S. Shamsudheen, "Metaheuristics with deep learning model for cybersecurity and Android malware detection and classification," *Appl. Sci.*, vol. 13, no. 4, p. 2172, Feb. 2023.

[4] M. Ibrahim, B. Issa, and M. B. Jasser, "A method for automatic Android malware detection based on static analysis and deep learning," *IEEE Access*, vol. 10, pp. 117334–117352, 2022.

[5] L. Hammood, İ. A. Doğru, and K. Kılıç, "Machine learning-based adaptive genetic algorithm for Android malware detection in auto-driving vehicles," *Appl. Sci.*, vol. 13, no. 9, p. 5403, Apr. 2023.

[6] P. Bhat and K. Dutta, "A multi-tiered feature selection model for Android malware detection based on feature discrimination and information gain," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 10, pp. 9464–9477, Nov. 2022.

[7] D. Wang, T. Chen, Z. Zhang, and N. Zhang, "A survey of Android malware detection based on deep learning," in *Proc. Int. Conf. Mach. Learn. Cyber Secur.* Cham, Switzerland: Springer, 2023, pp. 228–242.

[8] Y. Zhao, L. Li, H. Wang, H. Cai, T. F. Bissyandé, J. Klein, and J. Grundy, "On the impact of sample duplication in machine-learning-based Android malware detection," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 3, pp. 1–38, Jul. 2021.

[9] E. C. Bayazit, O. K. Sahingoz, and B. Dogan, "Deep learning based malware detection for Android systems: A comparative analysis," *Tehnički vjesnik*, vol. 30, no. 3, pp. 787–796, 2023.

[10] H.-J. Zhu, W. Gu, L.-M. Wang, Z.-C. Xu, and V. S. Sheng, "Android malware detection based on multi-head squeeze-and-excitation residual network," *Expert Syst. Appl.*, vol. 212, Feb. 2023, Art. no. 118705.

[11] K. Shaukat, S. Luo, and V. Varadharajan, "A novel deep learning-based approach for malware detection," *Eng. Appl. Artif. Intell.*, vol. 122, Jun. 2023, Art. no. 106030.

[12] J. Geremias, E. K. Viegas, A. O. Santin, A. Britto, and P. Horchulhack, "Towards multi-view Android malware detection through image-based deep learning," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, May 2022, pp. 572–577.

[13] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, ''MAPAS: A practical deep learning-based Android malware detection system,'' *Int. J. Inf. Secur.*, vol. 21, no. 4, pp. 725–738, Aug. 2022.

[14] S. Fallah and A. J. Bidgoly, ''Android malware detection using network traffic based on sequential deep learning models,'' *Softw., Pract. Exper.*, vol. 52, no. 9, pp. 1987–2004, Sep. 2022.

[15] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, and S. Son, ''De-LADY: Deep learning-based Android malware detection using dynamic features,'' *J. Internet Serv. Inf. Secur.*, vol. 11, no. 2, p. 34, 2021.

[16] W. Wang, M. Zhao, and J. Wang, ''Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network,'' *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 8, pp. 3035–3043, Aug. 2019.

[17] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, ''Efficient-Net convolutional neural networks-based Android malware detection,'' *Comput. Secur.*, vol. 115, Apr. 2022, Art. no. 102622.

[18] M. Masum and H. Shahriar, ''Droid-NNet: Deep learning neural network for Android malware detection,'' in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 5789–5793.

[19] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, ''PIndroid: A novel Android malware detection system using ensemble learning methods,'' *Comput. Secur.*, vol. 68, pp. 36–46, Jul. 2017.

[20] A. Guerra-Manzanares, H. Bahsi, and M. Luckner, ''Leveraging the first line of defense: A study on the evolution and usage of Android security permissions for enhanced Android malware detection,'' *J. Comput. Virol. Hacking Techn.*, vol. 19, no. 1, pp. 65–96, Aug. 2022.

[21] A. Taha and O. Barukab, ''Android malware classification using optimized ensemble learning based on genetic algorithms,'' *Sustainability*, vol. 14, no. 21, p. 14406, Nov. 2022.

[22] K. Sabanci, M. F. Aslan, E. Ropelewska, and M. F. Unlersen, ''A convolutional neural network-based comparative study for pepper seed classification: Analysis of selected deep features with support vector machine,'' *J. Food Process Eng.*, vol. 45, no. 6, Jun. 2022, Art. no. e13955.

[23] A. Batouche and H. Jahankhani, ''A comprehensive approach to Android malware detection using machine learning,'' in *Information Security Technologies for Controlling Pandemics*. USA: Springer, 2021, pp. 171–212.

[24] O. N. Elayan and A. M. Mustafa, ''Android malware detection using deep learning,'' *Proc. Comput. Sci.*, vol. 184, pp. 847–852, Jan. 2021.

[25] S. S. Sammen, M. Ehteram, Z. Sheikh Khozani, and L. M. Sidek, ''Binary coati optimization algorithm- multi- kernel least square support vector machine-extreme learning machine model (BCOA-MKLSSVM-ELM): A new hybrid machine learning model for predicting reservoir water level,'' *Water*, vol. 15, no. 8, p. 1593, Apr. 2023.

[26] M. A. Khan, S. Kadry, P. Parwekar, R. Damaševicius, A. Mehmood, J. A. Khan, and S. R. Naqvi, ''Human gait analysis for osteoarthritis prediction: A framework of deep learning and kernel extreme learning machine,'' *Complex Intell. Syst.*, vol. 2021, pp. 1–19, Jan. 2021.

[27] Z. Zhou, M. Liu, W. Deng, Y. Wang, and Z. Zhu, ''Clothing image classification with DenseNet201 network and optimized regularized random vector functional link,'' *J. Natural Fibers*, vol. 20, no. 1, Apr. 2023, Art. no. 2190188.

[28] D. Wen, S. Zheng, J. Chen, Z. Zheng, C. Ding, and L. Zhang, ''Hyperparameter-optimization-inspired long short-term memory network for air quality grade prediction,'' *Information*, vol. 14, no. 4, p. 243, Apr. 2023.

[29] *Andro-AutoPsy*. Accessed: Feb. 12, 2023. [Online]. Available: https://ocslab.hksecurity.net/andro-autopsy

[30] J.-W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, ''Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information,'' *Digit. Invest.*, vol. 14, pp. 17–35, Sep. 2015.