IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 June 2023, accepted 12 July 2023, date of publication 19 July 2023, date of current version 27 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3296789

RESEARCH ARTICLE

An Ensemble-Based Parallel Deep Learning Classifier With PSO-BP Optimization for Malware Detection

MOHAMMED NASSER AL-ANDOLI[1], KOK SWEE SIM[1], (Senior Member, IEEE), SHING CHIANG TAN[2], PEY YUN GOH[2], (Senior Member, IEEE), AND CHEE PENG LIM[3]

[1]Faculty of Engineering and Technology, Multimedia University, Melaka 75450, Malaysia
[2]Faculty of Information Science and Technology, Multimedia University, Melaka 75450, Malaysia
[3]Institute for Intelligent Systems Research and Innovation, Deakin University, Geelong Waurn Ponds, VIC 3216, Australia

Corresponding author: Kok Swee Sim (kssim@mmu.edu.my)

This research is supported by Fundamental Research Grant Scheme (FRGS), FRGS/1/2019/ICT02/MMU/02/2 and Multimedia University Grant, MMUI/220154.

ABSTRACT Digital networks and systems are susceptible to malicious software (malware) attacks. Deep learning (DL) models have recently emerged as effective methods to classify and detect malware. However, DL models often relies on gradient descent optimization in learning, i.e., the Back-Propagation (BP) algorithm; therefore, their training and optimization procedures suffer from several limitations, such as high computational cost and local suboptimal solutions. On the other hand, ensemble methods overcome the shortcomings of individual models by consolidating their strengths to increase performance. In this paper, we propose an ensemble-based parallel DL classifier for malware detection. In particular, a stacked ensemble learning method is developed, which leverages five DL base models and a neural network as a meta model. The DL models are trained and optimized with a hybrid optimization method based on BP and Particle Swarm Optimization (PSO) algorithms. To improve scalability and efficiency of the ensemble method, a parallel computing framework is exploited. The proposed ensemble method is evaluated using five malware datasets (namely, Drebin, NTAM, TOP-PE, DikeDataset, and ML_Android), and high accuracy rates of 99.2%, 99.3%, 98.7%, 100%, and 100% have been achieved, respectively. Its parallel implementation also significantly enhances the computational speed by a factor up to 6.75 times. These results ascertain that the proposed ensemble method is effective, efficient, and scalable, outperforming many other compared methods in malware detection.

INDEX TERMS Ensemble method, malware detection, deep learning, parallel processing, backpropagation algorithm, particle swarm optimization.

I. INTRODUCTION

In today's digital era, while the rapid development of computing technologies makes our lives convenient and easy, they are always open to cyber-attacks. The number of cyber-attacks and the associated damages have increased significantly, which has become one of the major threats in recent years [1]. Cyber-attacks cause trillions of dollars in damage to the global economy [2], [3]. Malicious software (malware) is developed by cyber-criminals to carry out unwanted activities on victims' computers. There are various types of malware, e.g. viruses, ransomware, worms, etc., which are capable of stealing important data, damaging computer systems and making themselves unseen in the target's system. Furthermore, malware proliferates by using users' trust as an infection vector [2], [3]. Owing to serious

The associate editor coordinating the review of this manuscript and approving it for publication was Taous Meriem Laleg-Kirati.

and detrimental effects of malware, the malware detection and security sector generates billions of dollars, and continues to grow every year.

In malware detection, suspicious activities and files can be classified as benign or malicious, depending on various characteristics such as infection capability, malware structure, and speed rate [4]. Threats and malware attacks should be quickly detected as soon as they infect the computing systems, in order to protect legitimate users. For this purpose, researchers have developed classification methods for detecting malware [5]. In general, malware detection techniques can be categorized into behavior, signature, model-checking, and heuristic approaches [2]. Nonetheless, many of the techniques perform poorly in detecting complex variants of malware [3], [6].

In recent years, deep learning (DL) has been widely applied to a variety of tasks, e.g., complex network analysis [7], computer vision [8], emotion recognition [9]. Malware detection is of no exception, as DL models have been employed to address the limitations of existing malware detection methods [10]. DL models offer several advantages over traditional machine learning (ML) models, such as the automatic generation of high-quality features and the capability to handle large datasets [11]. On the other hand, ensemble approaches can be used with ML/DL models to enhance their performance in classification tasks [12], [13]. The main principle is to merge multiple models and leverage the strengths of each model to compensate for individual disadvantages, thereby leading to improved performance [12], [13].

In the literature, various DL models, such as Graph Convolutional Network (GCN) and Recurrent Neural Network (RNN) [5], [14], hybrid DL-ML method [15], ensemble-based classification with DL [12], have been developed for malware detection. Even though useful results have been reported, they are neither efficient nor accurate enough for complex and large-scale malware detection tasks. DL models typically use the traditional optimization techniques, i.e., gradient descent with the Back-Propagation (BP) algorithm [7], for network learning. The BP algorithm suffers from several limitations, such as high computational costs and local suboptimal solutions [7], [16]. In addition, not many DL and optimization techniques in the literature leverage parallel processing to facilitate data processing and learning. These shortcomings make the current DL models ineffective in practice, particularly in dealing with big data challenges.

To undertake the aforementioned issues, we propose an ensemble-based parallel DL classifier for malware detection in this paper. The proposed ensemble method undergoes feature learning and classification in a parallel computing platform. Each DL constituent uses a hybrid Particle Swarm Optimization (PSO) and BP for parameter tuning, with an aim to find optimal solutions and improve overall performance. PSO has an ability to search for global optimal solutions [17], [18], while BP can effectively find accurate local solutions [19]. Therefore, a hybrid PSO-BP scheme entails complementary advantages of global and local search algorithms to optimize DL model parameters efficiently in a parallel computing environment. The proposed method, denoted as Ensemble Classification with Deep Learning Parallelism (ECDLP), has the following contributions:

- An ensemble DL method with parallel processing (i.e., is proposed for malware detection, leveraging the strength of individual DL models to improve classification performance;
- A hybrid optimization method based on a hybrid PSO-BP algorithm is introduced for optimization of individual DL model parameters, leveraging the complementary global search capability of PSO and local search capability of BP to achieve optimal solutions;
- A parallel computing scheme is developed to improve the scalability and efficiency of the proposed method.

The remaining sections of this paper are organized as follows. In Section II, we review the relevant studies in the existing literature. Section III provides a detailed explanation of the ECDLP method. The experimental results, analysis, and discussion are presented in Section IV, and concluding remarks are given in Section V.

II. RELATED WORK

Malware data require a careful analysis to identify malicious behaviors, for example, malware infection risk, structure, and spread speed. Several approaches and techniques have been suggested for malware detection, most of which involve ML with signature, behavior, heuristics, and model checking techniques. DL approaches have emerged recently to undertake malware detection [3].

Signature techniques first create a set of bits to represent the program structure, e.g., the malware structure. Next, the techniques analyse suspicious files and detect their signatures for classifying them into either benign or malware [20]. However, signature techniques are not scalable, and they also cannot detect complex malware variants [21]. Behaviour-based techniques initially monitor the characteristics of data samples (e.g., from a program) before identifying them as benign or malware [2]. Detecting changes in Application Program Interface (API), calls registry, system calls, file activities and computer networks are used to trace malware behaviours [2]. A graph model was developed by Kolbitsch et al. [22] to track malware behaviours by monitoring system calls. Lanzi et al. [23] designed a system centric behavioural model to identify interactions between malware and system resources. Behaviour-based techniques generally yield inferior performance, especially when analysing complex malware variants.

Heuristic approaches combine ML techniques and rule bases to handle malware detection tasks [24]. PSO and Apriori algorithms were combined in [25], in which PSO was used to generate and optimize candidate detectors while Apriori algorithm was employed to establish association rules for detecting malicious Android applications. Fatima et al. [26] developed a malware detection method using the Genetic

Algorithm (GA) to select appropriate features and feed them to an ML classifier.

Model checking methods use linear temporal logic for extracting representative malicious features by encoding them as flow relations, i.e. feature dependencies [2]. Song and Touili [27] developed a model checking method named ''pushdown'', in which all software and executable programs were converted into a pushdown system. The pushdown system used the stack computation tree predicate logic to represent malicious behaviours and exploit the associated predicates across the stack to detect malicious behaviours. Overall, malware detection based on model checking and heuristic methods are effective in detecting certain unknown malware variants. However, they are less effective to detect malware that uses obfuscation and packaging techniques.

Even though DL has been widely used in various domains including malware detection, there is room to further develop effective DL-based solutions for improving malware detection rates [3]. A DL malware detection framework was developed in [28] to analyze and identify dynamic and static malware. DL and image processing techniques were integrated to detect malware. Another DL framework was developed in [5] to learn and identify malware in Android devices. GCN was employed to identify the semantic and sequential patterns automatically, while the deep semantic information was determined by using an RNN model. In [29], an automatic detection method utilising Stacked Auto-encoder (SAE) and Deep Belief Network (DBN) was presented to detect unknown and new Android malware. In [30], a new DL-based malware detection method was proposed to overcome the issue of overfitting. Transfer learning was applied using pre-trained ShuffleNet and DenseNet-201 models as feature extractors. An Optimal-Error-Correction-Output-Coding ensemble model, called ECOC, consisting of Support Vector Machines (SVMs) was employed as the final classifier. The SVM parameters were optimized with a grid search approach. A lightweight DNN-based model was developed in [31] for malware classification. Denoted as IMCLNet, the method could effectively detect malware using advanced evasion techniques. As indicated in the study, IMCLNet was able to extract features directly from images without requiring additional feature engineering or domain knowledge for malware detection. In [32], a GCN was utilized to detect system call relationships and identify malicious Android software. Likewise [33] introduced an Android malware detection method utilizing a Graph Neural Network (GNN) with Jumping-Knowledge (JK) and its variations, such as GIN-JK and GraphSAGE-JK. Additionally, [34], employed a GNN for Android malware classification to create an API graph embedding technique, alongside the use of a Generative Adversarial Network (GAN) to target the graph-based GNN Android malware classifier.

Recently, a hybrid DL-ML model was devised in [15] to detect malware and prevent dissemination. A 12-layer DL model was combined with ML, e.g. Random Forest (RF), and voting was performed to reach a final decision. In [35], a semi-supervised DL method was designed to detect obfuscated malware and identify its family. DL processing, image transformation, and feature engineering were leveraged to accomplish such task. Similarly, a CNN-based model was introduced in [36] to classify variants of malware families. In this method, malware families were identified using a fine-tuned CNN architecture, and raw malware binaries were transferred into colour images. In [37], a hybrid architecture combining Long Short-Term Memory (LSTM) and CNN was proposed for detecting malicious software. The classification of malware was accomplished through the utilization of Call-Graph analysis, dynamic analysis, and static analysis. Similarly, [38] conducted an exploratory analysis to identify significant features that enhance the performance of malware detection. Additionally, [39] developed a method for Android malware detection, which involved Dalvik-Opcode features and permission features.

Ensemble methods overcomes the disadvantages of individual models by consolidating their strength in problem solving. Various malware detection methods based on ensemble learning have been proposed in the literature. In [13], an ensemble learning method for Android malware identification was developed. The method combined Naive Bayesian (NB), Decision Tree (DT), RF, and decision table ML algorithms was formed. In the method, intents and permissions were combined to employ a collaborative approach to enhance the classification performance of malware. A meta-ensemble model for Android malware detection was developed in [40]. It applied static analysis in identifying malicious applications. An ensemble approach was introduced in [41] to average the output of several classification models, including Logistic Regression (LR), NB, RF, K Nearest Neighbors (KNN), Stochastic Gradient Descent (SGD), and Support-vector machine (SVM). It was used for analysing Android software behaviours with an integration of static analysis and dynamic instrumentation. Similarly, an extrinsic random-based ensemble method for Android malware detection was proposed in [42]. In the method, the output of ML models, i.e., LR, SGD, and multilayer perceptron (MLP), was averaged to obtain a positive or negative prediction for each data samples (i.e., benign or malware). In [43], a prediction model for Android malware was developed based on ML algorithms using ensemble learning comprising RF, DT, LR, SVM, and XGboost was developed. The best algorithm was selected for malware detection. Louk and Tama [44] compared tree-based ensemble learning methods for analyzing and detecting Portable Executable (PE) malware and evaluating different methods, including RF and XGBoost, with accuracy and recall as the performance indicators. In [45], a multi-faceted deep Generative Adversarial Network model was suggested for an Android malware detection. On the other hand, various ML algorithms, i.e., RF, DT, LR, and SVM, were evaluated in [46] by using an ensemble learning for

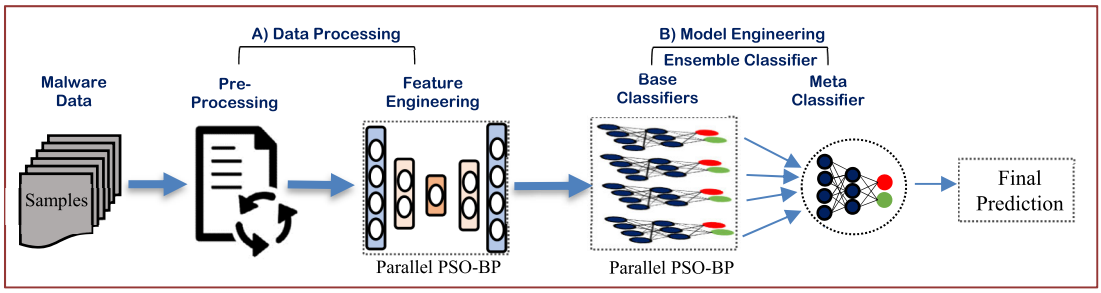


FIGURE 1. Outline of the proposed malware detection methodology (ensemble classification-based parallel deep learning with PSO-BP optimization for malware detection).

Android malware identification. Overall, the abovementioned methods demonstrate that ensemble learning is useful to improving the performance of individual malware detection models.

In the current literature, DL models are often not scalable to handle large-scale data because their architectures are complex. Moreover, the reported DLs typically utilize gradient optimization, which may lead to suboptimal generalization. To address this issue, this paper proposes an ensemble DL model that optimizes its parameters through a hybrid PSO-BP algorithm. To enhance efficiency in processing speed, a parallel processing platform is designed to house the ensemble DL model for malware detection. Compared with other methods reported in the literature, the proposed ECDLP model offers a new malware detection solution by integrating three distinctive characteristics within its framework, namely ensemble learning, hybrid optimization, and parallel DL processing. The efficiency and effectiveness of ECDLP are demonstrated through an empirical evaluation and performance comparison study, as detailed in Section IV.

III. METHODOLOGY

The methodology of ECDLP, which is shown graphically in Fig. 1, is explained in this section. In general, ECDLP is an ensemble-based parallel DL classifier with a hybrid PSO-BP optimizer for data classification. ECDLP involves two main phases: data preparation and model training. In the first phase, a dataset is pre-processed into a useful format for further analysis. Feature engineering then is carried out by using a DL method trained with a hybrid PSO-BP algorithm in a parallel processing platform. In the second phase, which is the main phase, malware detection is performed by training an ensemble model consisting of several base learners and a meta-learner. In ECDLP, the base learners are a series of Deep Neural Networks (DNNs), while the meta learner is a simple Neural Network (NN). Both base and meta learners are trained with the PSO-BP optimizer in a parallel processing platform.

A. DATA PROCESSING

1) PRE-PROCESSING

This study used numerical feature vectors to represent the malware samples from benchmark datasets, namely Drebin, NTAM, DikeDataset, Ml_Android, and TOP-PE. Data cleansing was performed to address missing values, although the amount was small. This served as a precautionary measure to ensure high-quality data representation for achieving improved model performance.

The Python library (Numpy) was employed for data cleansing. The pre-processing task involved replacing all missing values in a dataset with the average feature values. Note that averaging data summarizes the associated numerical features and determines their representative values without significantly affecting the sample representation, especially for behavioural features in malware detection. A z-score normalization process [47] is carried out, as defined in Eq. (1):

$$z - score = \frac{(x - \mu)}{\sigma} \tag{1}$$

where x, μ,and σ indicate to the original value, mean value, and standard deviation, respectively.

After z-score normalization, the next step is to reduce the large number of features in data samples. Some malware data, such as programs, have a large feature dimension. This causes DL-based models to be ineffective in learning. Therefore, the number of features is reduced to an affordable level pertaining to the computing power, which helps minimize computation time and facilitate the formation of a compact DL model. This task is accomplished by aggregating a set of neighbouring features and combine them to single feature [48]. The generation of adjacent features is controlled by a hyper-parameter F, which ranges from 1 to a, where a represents the original features. The new feature dimension (F_{size}) is expressed as follows:

$$F_{size} = \frac{a}{F} \tag{2}$$

After obtaining the normalized and reduced features, the new representation has F_{size} features, instead of the original set of features a, where $F_{size} < a$.

2) DEEP LEARNING-BASED FEATURE ENGINEERING

To further improve the quality of features, a DL-based feature engineering process to extract features and represent them in a lower-dimensional space is conducted. Feature engineering

is accomplished with an unsupervised DL model, i.e., Deep Auto-encoder (DAE). DAE learns useful features and transforms them from a high dimensional representation to a low-dimensional one. DAE typically comprises two stages: encoding and decoding. In the first stage, the data sample representation x is encoded in the hidden layer based on $h = f(W_1x + b_1)$, where f is the encoder function, x indicates the original sample, h refers to the extracted information, W_1 and b_1 are the adjustable parameters of the encoding layer. The decoding stage aims to generate a data representation (Y) similar or close to the original representation. This representation is generated from the output of the hidden layer using the g function, i.e.,$Y = g(W_2h + b_2)$, where W_2 and b_2 are the adjustable parameters of the decoding layer. Both stages with $f$and g perform a nonlinear mapping with the *sigmoid*and *ReLU* (rectified linear unit) functions [49]. DAE seeks to find a latent representation (h) such that h retains useful features of data samples (x), where the feature dimension of h is smaller than that of x. Its adjustable parameters are denoted by $\theta = [W_1, b_1, W_2, b_2]$. After accomplishing feature engineering and identifying the meaningful features, they are sent to the ensemble DL-based classifier for malware detection.

The process of feature extraction in a DAE occurs through its training and optimization process [50]. The aim is to minimize the reconstruction error between the input data sample and its reconstructed output by adjusting the DAE weights in both the encoding and decoding parts. Weights in the encoding layer are responsible for learning and generating useful features from the input data such that the most relevant and important features are extracted. The encoding layer is responsible to capture critical features and using them to reconstruct the input. The optimization process directs the extraction and learning of important features based on the reconstruction error between the input and its reconstructed output. To enhance the optimization efficiency, a hybrid PSO and BP algorithm is devised.

Specifically, the adjustable parameters in a DAE are optimized using a two-step method, i.e., PSO for global search and BP for local search, as shown in Fig. 2. PSO generates a pool of particles that utilize the output of BP for optimization. These particles are distributed across the search space, allowing them to exchange information and identify the optimal solution for further fine-tuning by BP. A detailed explanation of this hybrid algorithm is presented in Section III-B1.a

B. MODEL ENGINEERING

1) DEEP LEARNING-BASED MALWARE CLASSIFICATION

Once the samples features are prepared, they are propagated to another DL model, which is a modified version of DAE, to perform classification. The input features are handled by a DNN, and a prediction is generated at its last layer with ($n * d$), where d indicates the labels (target classes). The DNN hidden layers use functions $h = f(W_1x + b_1)$, $h_l = f_l(W_2h_l + b_2), \ldots o = f_l(W_{l+1}h_l + b_{l+1})$, where x represents the sample features, h represents the hidden layer, o represents the target class, and l represents the number of layers. The DNN adjustable parameters are indicated by θ. The DNN layers are dense, and a ReLU is utilized in all hidden layers. To implement ensemble learning, the proposed method incorporates two activation functions, namely the sigmoid function and the softmax function, in the last hidden layer of the base learners.

When designing DL models, two concerns are optimization and computational cost. ECDLP offers two advantages, i.e. hybrid PSO-BP method for achieving optimization, and parallel processing for improving computational speed. The details of ECDLP are explained in the next sub-sections.

a: HYBRID PSO-BP OPTIMIZATION

A hybrid meta-heuristic, i.e. PSO, and gradient, i.e. BP, optimization method can yield better solutions than those from individual methods [51]. PSO enables information sharing and promotes constructive collaboration among particles, resulting in the attainment of a global optimal solution [52]. In addition, PSO is simple and has proved its capability in solving complex optimization problems in various domains. On the other hand, BP is able to produce an non-linear mapping through local search [19]. In this research, we use the original version of PSO to perform global search, which is simple and fast in operation. Thus, the hybrid PSO-BP algorithm leverages the fast search capability of PSO in identifying global optimum solutions, which are then fine-tuned with BP. In addition, the potential drawback owing to simplicity of the original PSO algorithm is mitigated by forming an ensemble method. Good results in terms of accuracy and stability have been achieved, in line with those reported in the literature [51], [53], making the integration of PSO and BP in an ensemble framework a suitable choice for malware detection, as demonstrated in the empirical study.

ECDLP consists of DL-based feature engineering (Section III-A2) and DL-based classification modules (Section III-B1). Both modules utilize PSO and BP algorithms to optimize their parameters. A Weighted Cross Entropy (WCE) loss function (J_θ) is employed to refine parameter θ of DAEs. ECDLP is trained in such a way to minimize the loss function, as defined in Eq. (3).

$$J_\theta = \frac{1}{n}\sum_{i=1}^{n}[X_i \log(Y_i) + (1 - X_i)\log(1 - X_i)], \quad (3)$$

Eq. (3) is used to guide training of DAE through unsupervised (feature engineering) learning and supervised (classification) learning, respectively, where n,X_i are the number of samples and the original samples. In the feature engineering module, Y_i denotes the regenerated data sample, whereas in the classification module, Y_i represents the predicted class label (i.e., benign or malware). Parameter θ is updated iteratively through a local search process with BP,

as follows:

$$\theta_{\alpha}^{ij} = \theta_{\alpha}^{ij} - \gamma \frac{\partial}{\partial \theta_{\alpha}^{ij}} J_{\theta}(X, Y) \tag{4}$$

$$\begin{aligned} \frac{\partial}{\partial \theta_{\alpha}^{ij}} J_{\theta}\left(X, g(f(X))\right) &= \sum_{i=1}^{N} \frac{\partial}{\partial \theta_{\alpha}^{ij}} J_{\theta}\left(X_i, g(f(X_i))\right) \\ &= \sum_{i=1}^{N} \frac{\partial}{\partial z_{\alpha}^{j}} J_{\theta}\left(X_i, g(f(X_i))\right) \frac{\partial}{\partial \theta_{\alpha}^{ij}} z_{\alpha}^{j} \\ &= \sum_{i=1}^{N} \delta_{\alpha}^{j} X_i^T \end{aligned} \tag{5}$$

where N indicates to neurons, γ refers to the learning rate, and α refers to the activation values of the H hidden layer and Y output layer.

A set of DL models is created and executed using the entire data samples to optimize and tune the parameters of all replica DL models. The output of replica DL models is $\boldsymbol{M_{\theta}} = (M_{\theta 1}, M_{\theta 2}, \ldots, M_{\theta m})$, m indicates the number of DL replicas. $\boldsymbol{M_{\theta}}$ is calculated with Eqs. (3-5), i.e., local optimization with BP. Effective local solutions are found with BP, since it has fine-tuning optimization capabilities. PSO works with BP to search for optimal solutions. After performing local optimization with BP, $\boldsymbol{M_{\theta}}$ is re-used for PSO optimization.

The general idea of hybrid PSO-BP optimization is depicted in Fig. 2. Firstly, the PSO algorithm obtains $\boldsymbol{M_{\theta}}$ (parameters of DL replicas) optimized by BP. Next, a pool of particles P_s is generated, where P_s corresponds to the local replicas. Each particle P_i uses BP in its local optimization and represents one replica. Particles P_s are distributed to several regions of the search space. These particles have the capability to exchange information to each other, in order to search for the optimal solution. To update the position of each particle P_i, it moves with velocity $V_{i\theta}$ over iteration t based on Eqs. (6) and (7). BP updates the local positions (i.e., solutions) of the particles during local optimization using Eqs. (3) to (5), where the output, i.e., $\boldsymbol{M_{\theta}}$=$(M_{\theta 1}, M_{\theta 2}, M_{\theta 3}, \ldots M_{\theta m})$, is sent to PSO, as defined in Eqs (6) to (9):

$$\begin{aligned} V_{\theta i}^{(t+1)} &= \lambda v_{\theta i}^{(t)} + c_1 r_1 \left[p_{\theta i}^{best(t)} - M_{\theta i}^{(t)} \right] \\ &\quad + c_2 r_2 \left[g_{\theta}^{best(t)} - M_{\theta i}^{(t)} \right], \end{aligned} \tag{6}$$

$$i \in [1, P_s], M_{\theta i}^{(t+1)} = M_{\theta i}^{(t)} + V_{\theta i}^{(t+1)}, i \in [1, P_s] \tag{7}$$

For each particle (P_i,) the best local solution is updated with the following equation.

$$P_{\theta i}^{best} = M_{\theta i} | f\left(M_{\theta i}\right) = \min_{c=1,2,3,\ldots,t+1} \left\{ f\left(M_{\theta i,c}\right) \right\} \tag{8}$$

In Eqs. (6) to (8), r_1 and r_2 are two randomly chosen numbers in the interval [0, 1];$M_{\theta i}$ is the output of a local replica;λ is the inertia of the P_s movement; c_1 is the cognitive parameter and c_2 is the social parameter; while t, v, g^{best}, andp^{best} are the number of iterations, velocity, global best solution, and local best solution, respectively.

There are two ways to find the best global solution, either based on the average of the best local solutions p^{best} of all particles or the best local solution with the highest rank. In this research, the best solution is selected for minimization, as shown in Eq. (9).

$$g_{\theta}^{best} = \min \left(\begin{bmatrix} P_{\theta i}^{best} | f(P_{\theta i}^{best}) = \min\limits_{i=1,2,3,\ldots,P_s} \left\{ f(P_{\theta i}^{best}) \right\} \end{bmatrix}, \begin{bmatrix} \frac{1}{P_s} \sum_{i=1}^{P_s} P_{\theta i}^{best} \end{bmatrix} \right) \tag{9}$$

The fitness function is computed by calculating the average values of the loss function (i.e., WCE) from the local DL-replicas ($J_{\theta M}$=$(J_{\theta 1}, J_{\theta 2}, J_{\theta 3}, \ldots, J_{\theta m})$) in Eq. (3), as given in Eq. (10).

$$f(J_{\theta M}) = \frac{1}{m} \sum_{i=1}^{m} J_{\theta i} \tag{10}$$

where m refers to the DL-replicas. Once the fitness function is minimized and improved, $P_{i\theta}^{best}$ and g_{θ}^{best} are updated with Eqs. (6) to (9).

b: PARALLEL DEEP LEARNING

To improve efficiency of ECDLP, it is developed in a parallel processing platform. It operates at two levels: high-level parallelism with multi-processors/machines and low-level parallelism with multi-threads. Data parallelism is employed to build the platform. Parallel processing is carried out pertaining to the tasks of each base learner. In other words, there is no parallelization across the base learners, but in the tasks of each base learner. Fig. 2 shows the parallel processing method of the base learner, i.e., DL-based hybrid optimization with PSO and BP algorithms. Using the developed parallel processing method, many executers, i.e. cores/processors/computers, are first created based on the available resources. For each base learner, multiple DL replicas (M) are created. Each executer (E) receives a replica of the DL model. The dataset is then partitioned into small subsets and stored in local memory for parallelize processing. Parameter θ is shared with all replica DL models and stored in a server, i.e., global memory. Each E addresses one replica of the DL model M_i. Finally, the parallel replicas of DL models are synchronized and their outputs are combined to update the shared parameter.

In the high-level parallelism, a dataset is partitioned into chunks (C), and a number of executors are established according to the available resources, i.e., machines, processors, or cores. Executers (E) represent these resources, and they receive chunks of data with *C/E*. A set of DL replicas M is generated, and is allocated as particles P_s. The particles evolve through PSO in a high-level parallelism mode with multi cores/processors. Each individual particle, P_i, that corresponds to M_i carries out a search process and updates its local solution through parallel processing. The global

memory stores C and g^{best}, and shares them with connected E. The processing resources are allocated according to a ratio of 1: m, where m is the number of DL replicas. As an example, if we have a parallel system with 2 machines, each machine has 7-CPU cores, and consider $m = 3$; each replica of DL model uses 4 CPU cores resources. The low-level parallelism mode, i.e. multi-threads, is carried out with CPU cores for BP local optimization, including matrix multiplication [54]. After that, all DL replicas are synchronized to yield the results of local replicasM_θ. The fitness function of PSO is computed with Eq. (10). Next, the global best solution g^{best} is updated and sent to the global memory, while the local best solution, P_i^{best}, is sent to the local memory of each particle, e.g., M_i, for analysis.

2) STACKED ENSEMBLE MODEL FOR CLASSIFICATION

Model averaging is an ensemble technique in which multiple sub-models help produce a better prediction. Improvement is achieved by weighting the contributions of each sub-model to the generated prediction based on the expected performance of each individual model. The ensemble-based averaging model can be extended by employing a model to learn how to merge the contributions of each individual model. This approach is denoted as a stacked ensemble technique or stacked generalization.

In this study, we propose a stacked ensemble method combining heterogeneous classifiers. Fig. 3 shows the proposed stacked ensemble DL model architecture, which consists of five DNN models as the base learners and single NN as the meta learner. Specifically, the meta learner is a feedforward NN with a hidden layer and an output layer activated using a sigmoid function and trained using BP. Each base learner is a DNN classifier (as described in Section III-B1). All five DNN base learners (Fig. 3), whereby they utilize a sigmoid softmax functions in their operation, undergo the feature engineering and classification processes in parallel.

The architectures of the base learners, i.e., DNNs, utilize different activation functions and numbers of neurons, in order to ensure diversity among the base models used in ECDLP. Specifically, we use sigmoid and softmax activation functions, and each base model has a unique number of neurons, while maintaining consistent input and output layers. In addition, the base models are trained on different subsets of the dataset to further increase diversity of the ensemble model.

DNNs are selected as the base learners because they are effective in learning complex data representations and achieving high performance on various tasks, including malware detection [2], [28]. Additionally, they are highly parallelizable and can be optimized efficiently on digital hardware, making them well-suited for use in our parallel computing platform [10]. Furthermore, the evaluation indicates the effectiveness of the proposed ensemble model in achieving the best performance in comparison with those from other ensemble machine learning algorithms.

Once the base learners, i.e., the DNN models, have completed their operations, the results are merged together. Specifically, a concatenation process is carried out to merge all outputs into single vector A. Vector A is sent to the meta learner, which is used as features to perform classification and produce the final prediction.

The ECDLP operations are summarized in Algorithm 1.

Algorithm 1 ECDLP

Input: Data samples
Output: Classification of samples as malware or benign
Processing data:
  Pre-processing (Refer to Section III-A1)
  Feature engineering**:** (Refer to Section III-A2)
    for i=1 to t
      For all (m in M): ***do*** in ***parallel***
        Generate DL models
        Optimize DL models with PSO-BP (Section III-B1.a)
        Extract useful features of samples
      end parallel
    end for
 Stacking ensemble DL for malware detection and classification:
  a) Base classifiers
    for i=1 to e (Generate and train DL models) (Section III-B1)
      for i=1 to t
        For all (m in M): ***do*** in ***parallel***
          Generate replicas of DL models
          Optimize DL models with PSO-BP (Section III-B1.a)
          Classify samples as benign or malware
        end parallel
        end for
      end for
    Send the prediction values to the meta classifier
  b) Meta classifier
    Receive the prediction values from base classifiers and combine them into a vector A
    Generate meta learner (NN with hidden layer and output layer uses sigmoid function)
    Use A as input of the meta learner
    Train the meta learner and make the final prediction
Return malware and benign samples.

C. TIME COMPLEXITY ANALYSIS

In this section, we present the time complexity analysis of ECDLP and assess its efficiency. ECDLP consists of two optimization phases covering local optimization with BP and global optimization with PSO. During BP-based local optimization, the training steps comprise a feed-forward

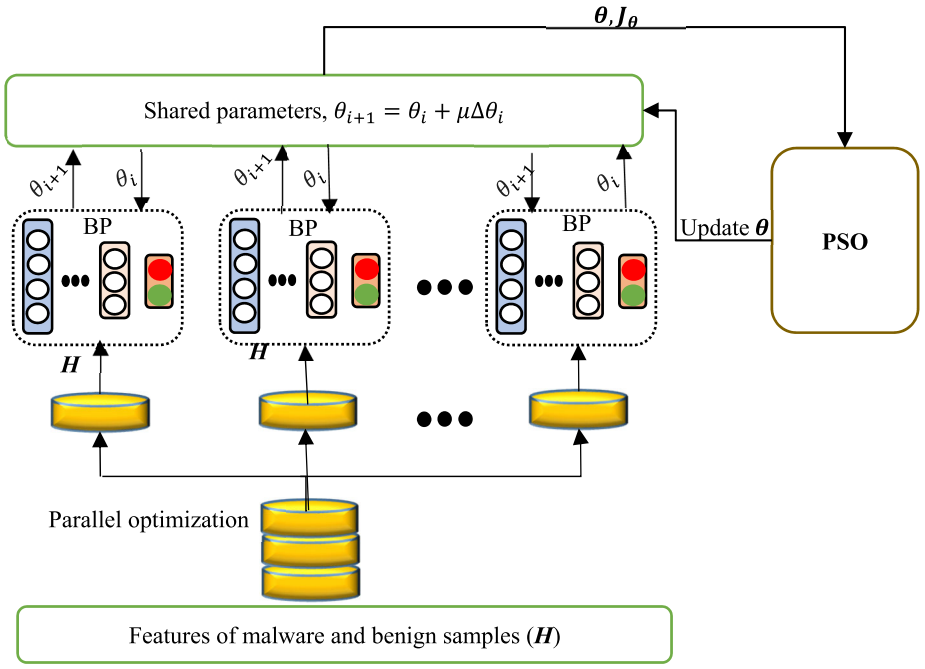


FIGURE 2. Parallel processing method of the base learner model (DL-based hybrid optimization with particle swarm optimization and backpropagation algorithms).

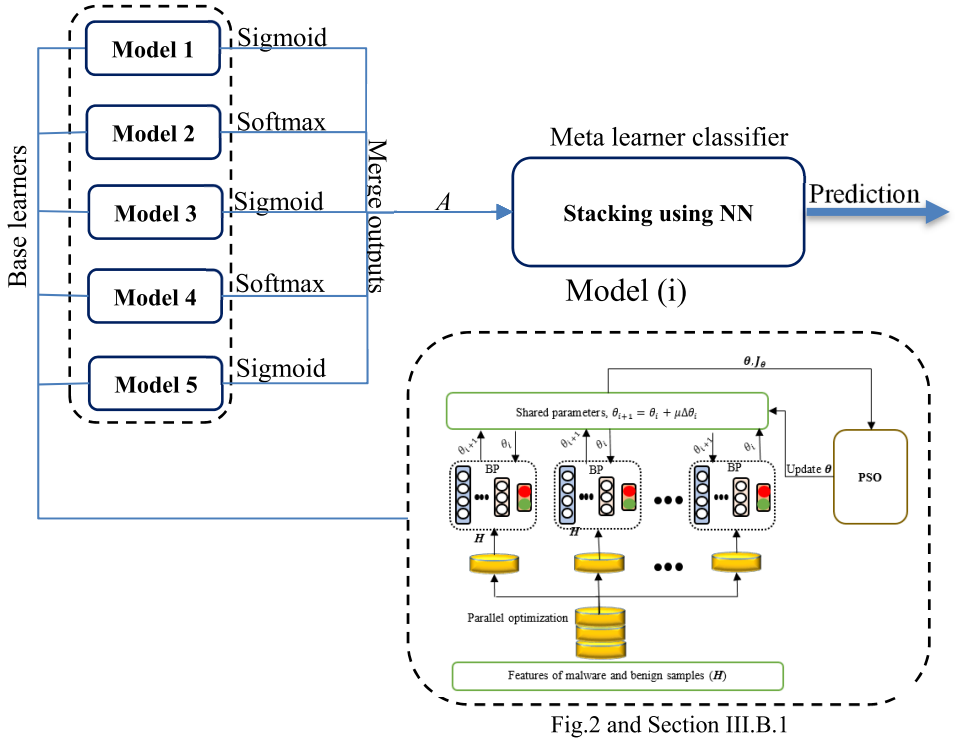


FIGURE 3. ECDLP: Stacking ensemble DL model architecture consisting of five DL models as base learners and a NN as a meta learner.

process J_θ and a back-ward process∂J_θ. As such, the time complexity is $T\left(J_\theta + \partial J_\theta\right) = O\left(2nho + 2nho\right) = O\left(4nho\right) = O\left(nho\right)$, where n and h indicate the numbers of training samples and hidden neurons, respectively, while o represents the number of neurons in the input/output layers. Assume the ECDLP architecture comprises multiple hidden

layers, l, and the DL model requires t iterations for training, the time complexity becomes $O(tlnho)$. The time complexity for PSO to undergo global optimization is $O(P_sD)$, where P_s denotes the swarm size, and $D$is the dimension of adjustable parameters. Since the time complexity of $D$is approximately $O(nho)$, the overall time complexity of PSO is$O(P_snho)$. The time complexity of single DL model in ECDLP is $O(P_stlnho)$. Given that ECLDP uses an ensemble method (E), this leads to time complexity of $O(EP_stlnho)$. However, as the number of models within the ensemble method is constant, where $E$is usually is set to a small number; the overall time complexity of ECDLP is approximately $O(P_stlnho)$.

We also compare the time complexity of the proposed ECDLP model with other ML and DL models, including SVM, Random Forest, CNN-LSTM, DBN-SAE, and ECDLP. The time complexity of SVM is $O(n^3)$ where n is the sample size, while that of Random Forest is $O(nm\log(k))$, where n is the sample size, m is the feature size, and k is the number of trees. The time complexity of CNN-LSTM is $O(mnp)$, where m is the number of time steps, n is the sample size, and p is the number of parameters, leading to $O(n^3)$. The time complexity of DBN-SAE is $O(nmho + mnp)$, where h is the number of hidden layers and o is the number of output nodes. For the proposed ECDLP model, its time complexity is $O(P_stlnho)$, where P_s and $t$are the content sizes and l is the number of deep layers. If $n$is greater than any parameters of the mentioned models, ECDLP remains within an acceptable range of complexity between nho and that of SVM (n^3).

IV. IMPLEMENTATION AND RESULTS

This section presents a set of experiments to validate the performance of ECDLP[1]. The obtained results are analyzed and discussed comprehensively.

A. DATASETS

In this study, we use five malware datasets for performance evaluation, as follows:

1) *Drebin*[2]. This is a common dataset used to identify and detect Android malware. It consists of 4,500 benign and 5,500 malware samples from 179 various malware families.
2) *Network Traffic Android Malware (NTAM)*[3]. This is a network layer feature set for malware detection. It comprises 3,141 malware samples and 4,704 benign samples.
3) *DikeDataset*[4]. It consists of 11,923 samples, comprising 10,841 malicious and 1,082 benign instances. These samples are extracted from Object Linking and Embedding (OLE) and Portable Executable (PE) files.
4) *Ml_Android*[5]. It consists of 10,0000 Android package kit (apk) samples, with 50,000 benign and 50,000 malware samples. The data samples contain behavioral features of malware.
5) *TOP-PE*[6]. It consists of 47,580 samples, with 45,651 malware and 1,929 benign samples. The top 1,000 features have been extracted from the Cuckoo-Sandbox report, and the benign samples have been collected from Windows 7-x86 and portableapps.com.

B. EXPERIMENTAL SETUP

Two machines have been used to analyze and verify the distributed computing and parallel implementation of ECDLP. The processors are Intel Core-i7, and the operating system is Ubuntu 20.04. One machine has 8GB RAM and the other has 4 GB. The Python Ray Library [55] is utilized to implement parallelization.

The DL model hyper-parameters are set as follows: the number of iterations, learning rate, and batch size are set to 200, 0.001, and 256, respectively. As for PSO, $c1 = 0.5$,$c2 = 0.3$, the inertia weight $= 0.9$,and the number of particles $= 20$. The position of each particle is randomly initialized within the range [0, 1]. The hyper-parameter F in Eq. (2) is set to 1 for Ml_Android and DikeDataset, while for Drebin and TOP-PE is set to 4 and 5, respectively. The ECDLP method is coded in the Python language. Its parameters are chosen after several trails, which allow ECDLP to yield the best results. Each dataset is split into a training set and a test set according to the ratio of 4 to 1.

For performance comparison, six ML algorithms are devised as ensemble classifiers, including Ensemble Random Forest (ERF), Ensemble Decision Tree (EDT), Ensemble Gradient Boosting (EGB), Ensemble K Nearest Neighbors (EKNN), Ensemble Ada Boost (EAB), and Ensemble Support Vector Machine (ESVM). In addition, RF, DT, GB, KNN, AB, and SVM are integrated to create a heterogeneous ML-based ensemble classifier, called HML-EC. Moreover, three existing DL based on a single classifier are used for malware detection, i.e., DBN-SAE [29], CNN-ML [28], and CNN-LSTM [37].

As the Drebin dataset is commonly utilized for malicious software detection, we have conducted another evaluation for comparing ECDLP with various ML and DL methods, which are based on single and ensemble classifiers. Specifically, seven state-of-the-art single classifier methods published between 2020-2022 have been used, i.e., GCN-JK [33], GCN-AMD [32], GIN-JK [33],VGAEMalGAN [34], SAGE-JK [33], DeepDiveDrebin [38], and FMulAMD [39]. Furthermore, nine state-of-the-art ensemble methods have been used, i.e., FSEC-MD [40], PIndroid [13], TA-AMD [41], EAMP-EML [43], ERE-AMD [42], MDGAN-MD [45], Stacking DT-SVM-LR [46], Blending DT-SVM-LR [46].

[1] "https://github.com/MNAl-Andoli/ECDLP"
[2] "https://www.sec.cs.tu-bs.de/~danarp/drebin/"
[3] "https://www.kaggle.com/datasets/xwolf12/network-traffic-android-malware"
[4] "https://github.com/iosifache/DikeDataset"
[5] "https://github.com/mburakergenc/Malware-Detection-using-Machine-Learning"
[6] "https://ieee-dataport.org/open-access/malware-analysis-datasets-top-1000-pe-imports"

TABLE 1. Comparison with three DL and seven ensemble ML methods performed on (a) the Drebin and (b) NTAM datasets.

	(a) Drebin				(b) NTAM			
Algorithm	Accuracy	Precision	Recall	F1-measure	Accuracy	Precision	Recall	F1-measure
ERF	96.3	96.8	96.7	96.7	96.9	97.6	96.0	96.8
EGB	90.6	89.0	94.5	91.7	96.7	96.1	97.2	96.6
EDT	95.4	94.3	97.3	95.7	95.1	99.3	88.3	93.5
EAB	90.9	91.2	92.3	91.8	96.1	95.6	96.5	96.1
ESVM	95.6	98.7	92.8	95.8	96.7	96.8	96.6	96.7
EKNN	88.4	96.6	80.8	88.4	95.0	95.6	94.3	94.5
HML-EC	96.7	97.0	96.9	96.9	97.7	98.0	97.4	97.6
CNN [28]	95.0	95.4	93.5	97.5	98.1	97.8	98.4	98.0
DBN-SAE [29]	94.7	95.1	93.7	96.6	95.5	96.8	94.4	95.6
CNN-LSTM [37]	96.5	96.4	97.9	97.7	98.7	98.5	98.8	98.7
ECDLP	**99.2**	**99.23**	**99.4**	**99.3**	**99.3**	**99.2**	**99.3**	**99.5**

TABLE 2. Comparison with three DL and seven ensemble ML methods performed on (a) the TOP-PE and (b) DikeDataset datasets.

	(a) TOP-PE				(b) DikeDataset			
Algorithm	Accuracy	Precision	Recall	F1-measure	Accuracy	Precision	Recall	F1-measure
EDT	95.4	95.6	99.5	97.5	96.5	97.8	98.3	98.0
ERF	96.4	96.7	99.5	98.1	96.8	98.2	98.3	98.2
EGB	95.2	95.2	98.6	97.7	94.3	96.3	97.4	96.8
EAB	96.5	97.3	99.1	98.2	95.7	97.2	98.1	97.6
EKNN	94.1	94.4	97.5	98.4	96.1	97.1	98.6	97.8
ESVM	95.9	96.3	99.3	96.7	97.4	98.0	99.1	98.6
HML-EC	97.5	97.9	99.5	98.7	97.4	98.4	98.9	98.7
CNN [28]	97.6	98.7	99.3	98.2	97.8	98.6	98.2	98.3
DBN-SAE [29]	95.0	93.1	95.3	91.2	95.4	96.8	98.1	97.5
CNN-LSTM [37]	97.7	98.8	99.2	98.4	98.3	98.2	98.4	98.3
ECDLP	**98.7**	**98.9**	**99.8**	**99.3**	**99.5**	**99.3**	**99.7**	**99.6**

C. EVALUATION METRICS

Accuracy, precision, recall, and F1-measure metrics are used for performance evaluation and comparison. These metrics are computed based on True Positive (*TP*), True Negative (*TN*), False Positive (*FP*), and False Negative (*FN*) rates, as follows:

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

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

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

$$F1 - measure = \frac{2 \times (Recall \times \Pr ecision)}{Recall + \Pr ecision} \tag{14}$$

Speedup indicator is computed, as in Eq. (15).

$$Speedup(A, B) = \frac{Method(A)}{Method(B)}(time), \tag{15}$$

D. PERFORMANCE COMPARISON

The experimental results are summarized in Tables 1-6, where the bold scores indicate the best performance. Table 1 presents the results of ECDLP, seven ensemble ML and three DL methods on the (a) Drebin and (b) NTAM datasets. For the Drebin dataset, ECDLP outperforms all other methods (EDT, EGB, ERF, EKNN, EAB, ESVM, HML-EC, CNN, DBN-SAE, and CNN-LSTM) in all four metrics. HML-EC, ERF, and CNN-LSTM harvest the best performance among the comparison methods. On the other side, EAB and EKNN have the lowest performance in all metrics. The results also demonstrate that ensemble ML algorithms provide a good performance, as compared with those of DL-based single classifiers, i.e., CNN, DBN-SAE, and CNN-LSTM. HML-EC outperforms all ensemble-based ML classifiers, including EDT, EGB, ERF, EKNN, EAB, and ESVM. Again, the NTAM results in Table 1(b) clearly indicate that ECDLP obtains the best accuracy, precision, recall, and F1-measure scores.

The TOP-PE and DikeDataset results of ECDLP and compared methods are presented in Table 2(a) and (b), respectively. ECDLP achieves the best performance, outperforming all DL and ensemble ML methods. CNN and CNN-LSTM report the best performance among the compared methods, which are close to those of ECDLP. EDT and EKNN yield the lowest performance in all four indicators.

Referring to the Ml_Android dataset (Table 3), ECDLP performs excellently with a perfect (100%) score in all four metrics, which is on par with ERF, EKNN, and HML-EC. According to Table 3, while CNN-LSTM achieves 100% accuracy on the Mal_Android dataset, its performance is not consistently the highest across different datasets. This is particularly clear when the dataset involved is complex with high-dimensional features, e.g. the Drebin dataset (Table 1(a)). Comparatively, our proposed ECDLP method demonstrates a superior performance in four metrics across on all five datasets. To mitigate its complexity as compared with single models, ECDLP leverages parallel computing for ensemble learning, which enhances its computational efficiency and scalability for tackling large-scale datasets.

Referring to the Dikedataset, NTAM, and TOP-PE datasets (Section IV-A), they exhibit imbalanced data distribution. This can lead to bias in performance with a high accuracy rate, due to correct classification of the majority-class samples. To avoid this issue, we apply a random over-sampling technique [56] to increase the number of minority-class samples, leading to a balance between both majority- and minority-class samples. As shown in Tables 1(b) and 2(a and b), our proposed model performs well in terms of accuracy, precision, recall, and, most importantly, F1-measure, which is a useful metric insensitive to imbalanced data distribution.

Overall, as shown in Tables 1 to 3, ECDLP depicts a high performance in terms of accuracy, precision, recall, and F1-measure. There are several reasons pertaining to the effectiveness of ECDLP. Firstly, ECDLP comprises two DL models, one for feature engineering and another for data classification. Secondly, ECDLP uses optimization based on PSO and BP for devising DL models. This allows ECDLP to find the global optimum and enhance generalization. Thirdly, ECDLP employs the ensemble approach to leveraging the strengths of individual models to achieve the best performance.

To ascertain the results in Tables (1) to (3), two non-parametric statistical tests have been conducted with Friedman and Nemenyi post-hoc tests. The Friedman test computes the average ranked performance with the four indicators at a level with α =0.05 (i.e., 95% certainly level) with respect to ensemble ML methods consisting of ERF, EDT, EAB, EGB, ESVM, EKNN, and HML-EC methods (Tables 1 to 3) and DL methods comprising DBN-SAE [29], CNN [28], and CNN-LSTM [37] (Tables 1 to 3). Table 4 presents the p-values obtained from the Friedman test, which indicate a significant difference in performance between EML and DL at a significance level of α =0.05.

TABLE 3. Comparison with three DL and seven ensemble ML methods performed on the ML_Android dataset.

Algorithm	Accuracy	Precision	Recall	F1-measure
EDT	99.2	98.8	99.6	99.2
ERF	**100**	**100**	**100**	**100**
EGB	99.8	99.6	**100**	99.8
EAB	94.1	98.2	92.8	93.8
EKNN	**100**	**100**	**100**	**100**
ESVM	99.8	99.8	99.8	99.8
HML-EC	**100**	**100**	**100**	**100**
CNN [28]	99.9	99.8	99.9	99.9
DBN-SAE [29]	94.5	97.4	96.9	97.1
CNN-LSTM [37]	**100**	**100**	**100**	**100**
ECDLP	**100**	**100**	**100**	**100**

TABLE 4. Friedman test results of the ECLDP method.

Methods	Accuracy	Precision	Recall	F1-Measure
EML	0.000746	0.002239	0.013029	0.001357
DL	0.00204	0.009537	0.005846	0.002038

Since the Friedman tests show that the results of ECDLP, EML, and DL methods are significantly different, we apply the Nemenyi post-hoc test to find the difference in performance. Figs. 4 and 5 present the Nemenyi test results. It is evident that ECDLP harvests the highest performance rank across all four indicators for both EML and DL methods. HML-EC achieves the highest rank among EML methods and closely matches the performance of ECDLP. This can be attributed to its utilization of heterogeneous EML models. Among DL models, CNN-LSTM attains the highest rank, yet it still falls short compared to ECDLP.

We conduct two additional studies using the Drebin dataset to further validate the effectiveness of ECDLP, i.e., (1) a comparison between ECDLP and state-of-the-art based on single classifiers, as in Table (5); and (2) a comparison between ECDLP and state-of-the-art based on ensemble classifiers, as in Table (6). Referring to Tables (5) and (6), ECDLP outperforms all compared methods based on single and ensemble classifiers in terms of all four metrics. The performance improvements of ECDLP pertaining to single classifiers are better than those of ensemble methods.

In addition, we have included three optimization-based techniques published in 2022 for performance comparison [57], [58]. In [57], a binary-PSO optimization was combined with DL, and the method achieved an accuracy of up to 94.92% with the Drebin dataset. In [59], the proposed ensemble-based genetic optimization produced an accuracy rate of 94.15% with the Drebin dataset. In [58], two swarm intelligence optimization techniques (i.e, Bald Eagle Search and Sailfish Optimization) were employed in conjunction

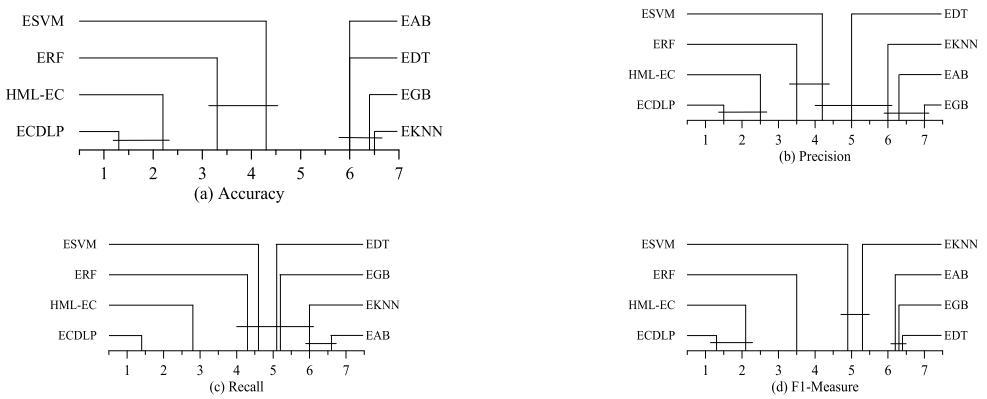


FIGURE 4. Average rankings Performance between ECLDP and the EML- methods.

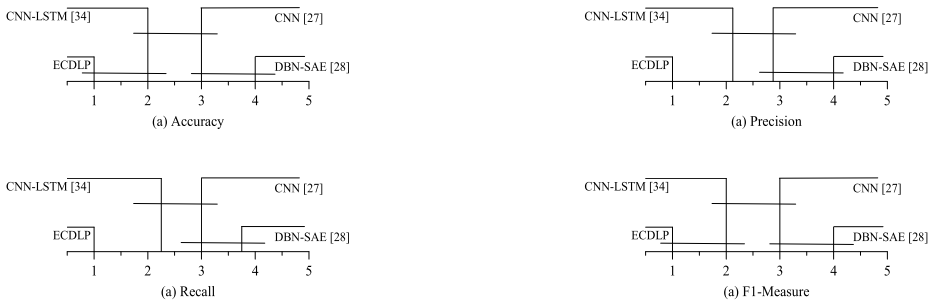


FIGURE 5. Average rankings Performance between ECLDP and the DL- methods.

TABLE 5. Performance comparison with single methods on the Drebin dataset.

Method	Accuracy	Precision	Recall	F1-measure
SAGE-JK [33]	95.24	95.35	95.24	95.0
GCN-AMD [32]	92.30	91.50	92.30	93.30
GCN-JK [33]	96.88	97.01	96.88	97.0
VGAEMalGAN [34]	97.68	95.27	91.08	93.13
GIN-JK [33]	95.03	95.07	95.03	95.0
FMulAMD [39]	96.00	97.00	96.00	95.00
DeepDiveDrebin [38]	96.8	97.8	89.4	93.4
ECDLP	99.2	**99.23**	**99.4**	**99.3**

TABLE 6. Performance comparison with ensemble-based methods on Drebin dataset.

Method	Accuracy	Precision	Recall	F1-measure
EAMP-EML [43]	**99.3**	99.0	99.0	99.0
PIndroid [13]	98.4	97.5	97.5	97.5
MDGAN-MD [45]	96.2	95.1	94.6	94.7
Blending DT-SVM-LR [46]	97.7	96.0	98.0	97.0
Stacking DT-SVM-LR [46]	98.0	97.0	98.0	97.0
TA-AMD [41]	98.2	-	-	-
FSEC-MD [40]	97.6	-	-	-
ERE-AMD [42]	99.1	-	-	-
ECDLP	99.2	**99.23**	**99.4**	**99.3**

with other ML models, which recorded an accuracy rate of 98.92% with the Drebin dataset. Meanwhile, our proposed ECDLP method is able to yield 99.2% accuracy on the Drebin dataset, outperforming all three aforementioned methods.

E. RUNTIME, SPEED-UP, AND SCALABILITY EVALUATION

In this section, the ECDLP efficiency is evaluated with three metrics, namely runtime, speedup, and scalability. Firstly, the runtime results of ECDLP, seven EML methods (i.e., ERF, EDT, EGB, EKNN, EAB, and ESVM, and HML-EC), and three DL methods (i.e., CNN, DBN-SAE, and CNN-LSTM), from experiments with five malware datasets using different computing processors are computed.

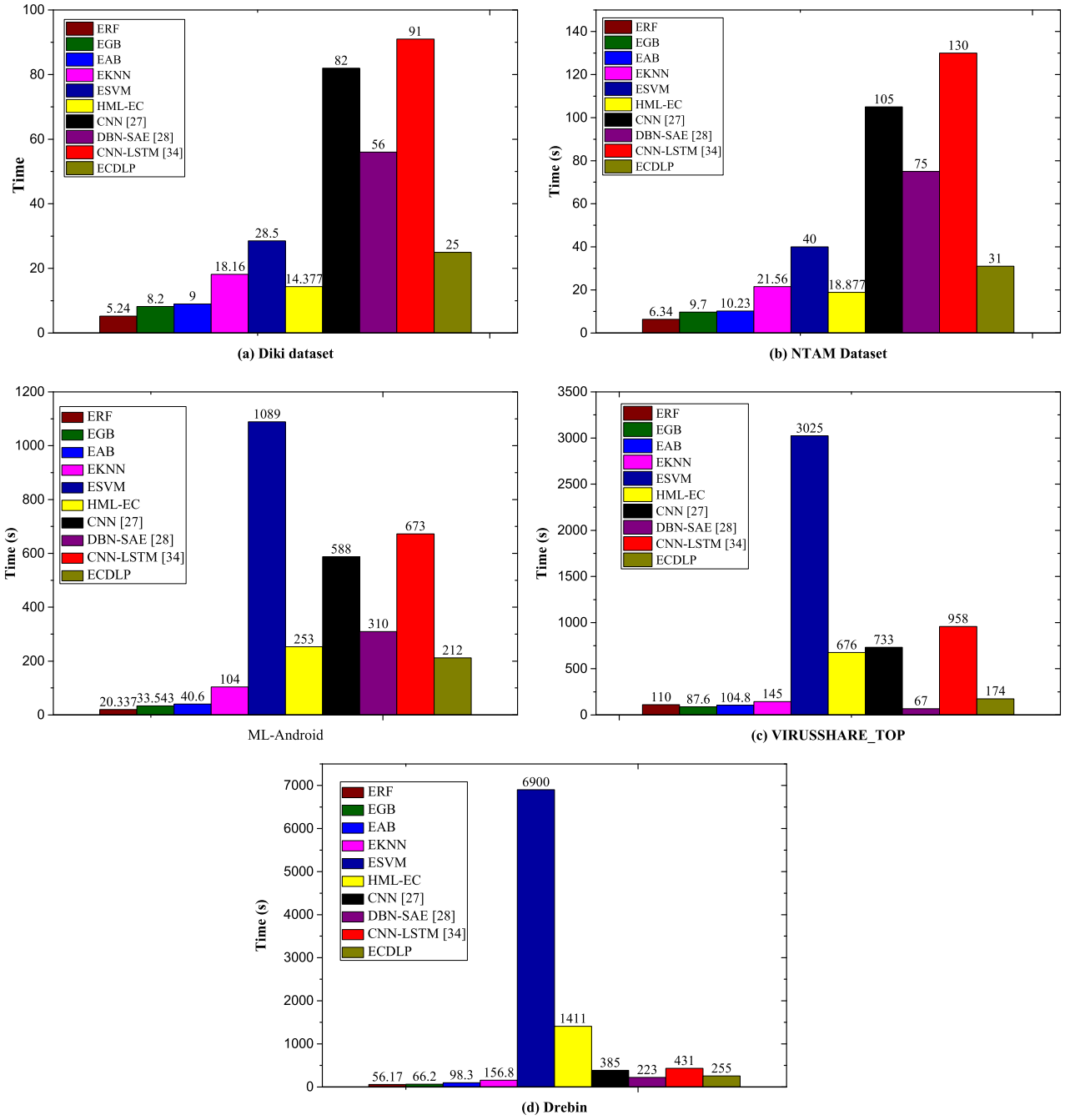
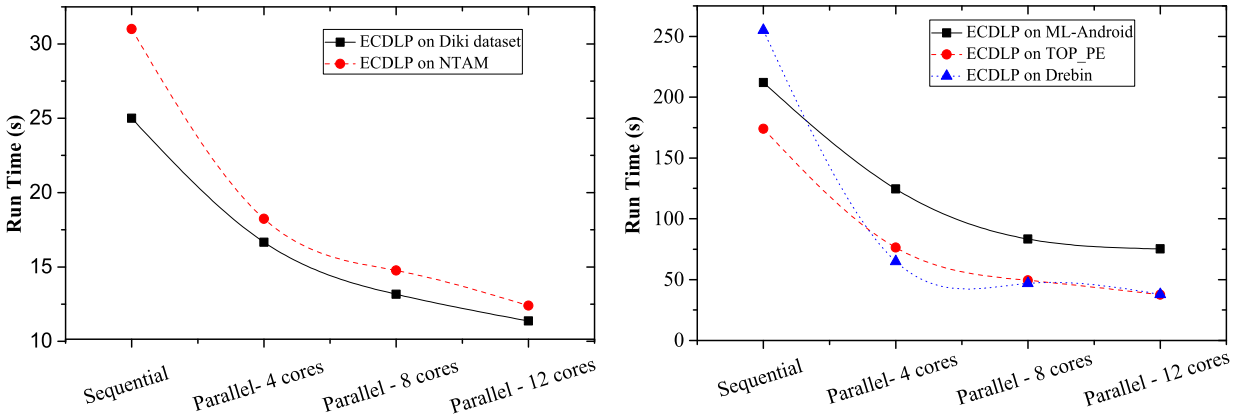


FIGURE 6. Run time of the proposed ECDLP method, the seven EML and the three DL methods.

The experiments have been initially performed sequentially with one computing processor, because the comparison methods are implemented in a sequential way, instead of in a parallel platform. Fig. 6 shows the results in terms of computational time from ECDLP, seven EML, and three DL methods. The longest time was consumed by ESVM with large datasets, such as Mal-Android, Top-PE, and Drebin. However, it performs well with acceptable runtime when dealing with medium sized datasets such as DikiDataset an NTAM. ERF exhibits the shortest runtime consumption. Other EML methods consume a shorter run time than those of DL methods, including ECDLP. ECDLP records shorter runtimes than those of DL methods in all datasets. It also consumes shorter runtimes as compared with those from ESVM in large-scale datasets, i.e., Ml_Android, TOP-PE, and Drebin.

In summary, while ECDLP achieves the best results pertaining to all four indicators (accuracy, precision, recall, and F1 measure), it still lacks efficiency in terms of runtime in sequential implementation. Therefore,



(a) Diki and NTAM datasets

(b) ML-Android, TOP-PE, and Drebin datasets

FIGURE 7. Run time of ECDLP with three parallel settings and sequential implementation.

a parallel processing platform is developed to overcome this limitation.

The experiment has been further expanded to execute ECDLP in a parallel processing platform. Two computing machines with 2 CPUs of 12 cores are used. Figs. 7 (a) and 7 (b) depict the ECDLP runtime results graphically using sequential and three parallel implementations over medium-sized (DikiDataset and NTAM) and large-sized (TOP-PE, ML_Android, and Drebin) datasets. As can be observed in Fig 7 (a) and 7 (b), the sequential implementation takes the longest computation time on all datasets, while parallel implementation with 12-cores consumes the shortest time. As expected, the computation time of ECDLP is reduced when a parallel processing platform is adopted.

Fig. 7 (a) and 7 (b) depict the computational time of ECDLP in a sequential implementation, as well as in *parallel-4-cores, parallel-8-cores, and parallel-12-cores* settings. As shown in Figs. 7 (a) and 7 (b), *parallel-4-cores*consume the longest runtime, while *parallel-12-cores* records the shortest runtime. The results in Figs. 7 (a) and 7 (b) indicate that ECDLP allows the inclusion of additional resources (i.e., cores, processors, machines) to reduce its computational time. From the results in Figs. 7 (a) and 7 (b), the ECDLP runtime improves significantly with the incorporation of new resources, indicating the scalability capability of ECDLP.

Speed-up and scalability are next evaluated to further gauge the efficiency of ECDLP. Eq. (15) is used to calculate the relative speed-up performance between two methods, while scalability indicates how the performance of the parallel method is affected by adding new resources in a parallel processing platform, i.e., computing cores, processors, or machines.

Fig. 8 depicts the speed-up rates of ECDLP on three parallel platforms, i.e., *parallel-4-cores, parallel-8-cores, and parallel-12-cores*), for processing the five datasets. Speed-up of these parallel platforms is computed by comparing them with those from the sequential implementation, which

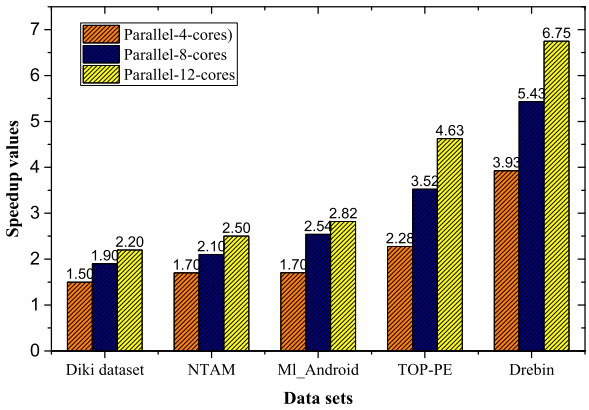


FIGURE 8. Speedup of ECDLP with three parallel platforms and five malware datasets.

is denoted as *Method (A)* in Eq. (14). The results in Fig. 8 indicates that ECDLP with *Parallel-12-cores* achieves the highest speed-up rates, i.e., in the order of 2.2x, 2.5x, 2.82x, 4.63x, 6.75x as compared with those of sequential implementation in processing the DikiDataset, NTAM, Ml_Android, TOP-PE, and Drebin datasets, respectively. The results in Fig. 8 indicate that ECDLP with parallel-4-cores records the lowest speed-up performance. All parallel processing platforms outperform sequential processing in handling all five datasets. A higher speed-up can be achieved when handling larger datasets, as shown in Fig 8.

The trade-off between effectiveness and efficiency refers to the balance between the quality of results and the resources (e.g. time, computing power) required to achieve the results. Pertinent to complex DL models with multiple layers with ECDLP, a trade-off between effectiveness and efficiency in ECDLP performance is exhibited. Indeed, comparing ECDLP with several ML models (RF, KNN, and EAB), there is a compromise between effectiveness and efficiency. While ECDLP operates not as fast as other models, it is able to yield

better results. Compared with other models such as SVM and CNN-LSTM, ECDLP depicts both high effectiveness and efficiency. To mitigate the computational load of ECDLP, its speed efficiency can be greatly improved with the use of parallel computing, as demonstrated in the results shown in Figs 6 to 8.

V. CONCLUSION

In this paper, we have proposed an effective and efficient ensemble method, denoted as ECDLP, for malware detection. ECDLP first extracts the meaningful features from malware data samples using the DAE model. The data samples with newly extracted features are sent to DNN models to perform classification, i.e., determining whether a sample is either benign or malignant. In this phase, a stacked ensemble learning method has been devised to combine the decisions from DNN models to improve the performance. The stacked ensemble method uses five DNN models as the base learners and one NN as the meta learner. During the training and optimization stage, ECDLP uses a hybrid BP and PSO algorithm that combines local and global search capabilities to identify global optima for improving the performance. Finally, ECDLP exploits parallel computing to improve its scalability and efficiency. Five malware datasets have been used to evaluate the performance of ECDLP on two parallel machines. ECDLP outperforms several ensemble ML and state-of-art DL malware detection methods. The experimental results have demonstrated that ECDLP is able to improve the computational speed up to an order of 6.75 times, ascertaining the usefulness of the developed parallel processing platform.

For future work, we will investigate a new DL architecture for malware detection using various data modalities, such as text and images. Another research direction is to extend the proposed model to handle multi-class problems, such as predicting different malware types and families, and to assess its efficacy in undertaking large scale and complex malware detection tasks in real-world environments. We also aim to explore different DL architectures and other ML algorithms to enhance diversity of the ensemble model. In addition, the current work could be extended to handle big data and execute on large distributed and parallel computing systems, involving multiple interconnected machines in the future.

REFERENCES

[1] A. Jamal, M. F. Hayat, and M. Nasir, "Malware detection and classification in IoT network using ANN," *Mehran Univ. Res. J. Eng. Technol.*, vol. 41, no. 1, pp. 80–91, 2022, doi: 10.22581/muet1982.2201.08.

[2] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020, doi: 10.1109/ACCESS.2019.2963724.

[3] Ö. Aslan and A. A. Yilmaz, "A new malware classification framework based on deep learning algorithms," *IEEE Access*, vol. 9, pp. 87936–87951, 2021, doi: 10.1109/ACCESS.2021.3089586.

[4] R. Komatwar and M. Kokare, "A survey on malware detection and classification," *J. Appl. Secur. Res.*, vol. 16, no. 3, pp. 390–420, 2021, doi: 10.1080/19361610.2020.1796162.

[5] X. Pei, L. Yu, and S. Tian, "AMalNet: A deep learning framework based on graph convolutional networks for malware detection," *Comput. Secur.*, vol. 93, Jun. 2020, Art. no. 101792, doi: 10.1016/j.cose.2020.101792.

[6] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: A survey," *Proc. IEEE*, vol. 108, no. 10, pp. 1825–1848, Oct. 2020, doi: 10.1109/JPROC.2020.2993293.

[7] M. N. Al-Andoli, S. C. Tan, W. P. Cheah, and S. Y. Tan, "A review on community detection in large complex networks from conventional to deep learning methods: A call for the use of parallel meta-heuristic algorithms," *IEEE Access*, vol. 9, pp. 96501–96527, 2021, doi: 10.1109/ACCESS.2021.3095335.

[8] M. Hassaballah and A. I. Awad, *Deep Learning in Computer Vision: Principles and Applications*. London, U.K.: CRC Press, 2020.

[9] S. M. S. A. Abdullah, S. Y. A. Ameen, M. A. Sadeeq, and S. Zeebaree, "Multimodal emotion recognition using deep learning," *J. Appl. Sci. Technol. Trends*, vol. 2, no. 2, pp. 52–58, 2021.

[10] M. N. Al-Andoli, S. C. Tan, K. S. Sim, C. P. Lim, and P. Y. Goh, "Parallel deep learning with a hybrid BP-PSO framework for feature extraction and malware classification," *Appl. Soft Comput.*, vol. 131, Dec. 2022, Art. no. 109756, doi: 10.1016/j.asoc.2022.109756.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] R. Damaševičius, A. Venčkauskas, J. Toldinas, and Š. Grigaliūnas, "Ensemble-based classification using neural networks and machine learning models for windows PE malware detection," *Electronics*, vol. 10, no. 4, p. 485, Feb. 2021, doi: 10.3390/electronics10040485.

[13] 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, doi: 10.1016/j.cose.2017.03.011.

[14] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of Android malware detection with deep neural models," *ACM Comput. Surveys*, vol. 53, no. 6, pp. 1–36, Nov. 2021, doi: 10.1145/3417978.

[15] S. Yoo, S. Kim, S. Kim, and B. B. Kang, "AI-hydra: Advanced hybrid approach using random forest and deep learning for malware classification," *Inf. Sci.*, vol. 546, pp. 420–435, Feb. 2021, doi: 10.1016/j.ins.2020.08.082.

[16] B. Akay, D. Karaboga, and R. Akay, "A comprehensive survey on optimizing deep learning models by metaheuristics," *Artif. Intell. Rev.*, vol. 55, pp. 1–66, Mar. 2021, doi: 10.1007/s10462-021-09992-0.

[17] F. Pan, Q. Zhou, W.-X. Li, and Q. Gao, "Analysis of standard particle swarm optimization algorithm based on Markov chain," *Acta Autom. Sinica*, vol. 39, no. 4, pp. 381–389, 2013, doi: 10.1016/S1874-1029(13)60037-3.

[18] E. T. Mohamad, D. J. Armaghani, E. Momeni, A. H. Yazdavar, and M. Ebrahimi, "Rock strength estimation: A PSO-based BP approach," *Neural Comput. Appl.*, vol. 30, no. 5, pp. 1635–1646, Sep. 2018, doi: 10.1007/s00521-016-2728-3.

[19] U. Bhattacharya and S. K. Parui, "Self-adaptive learning rates in backpropagation algorithm improve its function approximation performance," in *Proc. Int. Conf. Neural Netw.*, 1995, pp. 2784–2788, doi: 10.1109/ICNN.1995.488172.

[20] M. F. Zolkipli and A. Jantan, "A framework for malware detection using combination technique and signature generation," in *Proc. 2nd Int. Conf. Comput. Res. Develop.*, May 2010, pp. 196–199, doi: 10.1109/ICCRD.2010.25.

[21] J. Scott, "Signature based malware detection is dead," Inst. Crit. Infrastruct. Technol., Washington, DC, USA, Jun. 2017.

[22] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-Y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. USENIX Secur. Symp.*, vol. 4, no. 1, 2009, pp. 351–366.

[23] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "AccessMiner: Using system-centric models for malware protection," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, Oct. 2010, pp. 399–412, doi: 10.1145/1866307.1866353.

[24] K. Alzarooni, "Malware variant detection," Doctoral thesis, Dept. Comput. Sci., Univ. College London, London, U.K., 2012.

[25] O. S. Adebayo and N. Abdul Aziz, "Improved malware detection model with apriori association rule and particle swarm optimization," *Secur. Commun. Netw.*, vol. 2019, pp. 1–13, Aug. 2019, doi: 10.1155/2019/2850932.

[26] A. Fatima, R. Maurya, M. K. Dutta, R. Burget, and J. Masek, "Android malware detection using genetic algorithm based optimized feature selection and machine learning," in *Proc. 42nd Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2019, pp. 220–223, doi: 10.1109/TSP.2019.8769039.

[27] F. Song and T. Touili, "Pushdown model checking for malware detection," *Int. J. Softw. Tools Technol. Transf.*, vol. 16, no. 2, pp. 147–173, Apr. 2014, doi: 10.1007/s10009-013-0290-1.

[28] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019, doi: 10.1109/ACCESS.2019.2906934.

[29] S. Hou, A. Saas, L. Chen, Y. Ye, and T. Bourlai, "Deep neural networks for automatic Android malware detection," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Jul. 2017, pp. 803–810, doi: 10.1145/3110025.3116211.

[30] W. K. Wong, F. H. Juwono, and C. Apriono, "Vision-based malware detection: A transfer learning approach using optimal ECOC-SVM configuration," *IEEE Access*, vol. 9, pp. 159262–159270, 2021, doi: 10.1109/ACCESS.2021.3131713.

[31] B. Zou, C. Cao, F. Tao, and L. Wang, "IMCLNet: A lightweight deep neural network for image-based malware classification," *J. Inf. Secur. Appl.*, vol. 70, Nov. 2022, Art. no. 103313, doi: 10.1016/j.jisa.2022.103313.

[32] T. S. John, T. Thomas, and S. Emmanuel, "Graph convolutional networks for Android malware detection with system call graphs," in *Proc. 3rd ISEA Conf. Secur. Privacy (ISEA-ISAP)*, Feb. 2020, pp. 162–170, doi: 10.1109/ISEA-ISAP49340.2020.235015.

[33] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "Graph neural network-based Android malware classification with jumping knowledge," 2022, *arXiv:2201.07537W*.

[34] R. Yumlembam, B. Issac, S. M. Jacob, and L. Yang, "IoT-based Android malware detection using graph neural network with adversarial defense," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8432–8444, May 2023, doi: 10.1109/JIOT.2022.3188583.

[35] A. Darem, J. Abawajy, A. Makkar, A. Alhashmi, and S. Alanazi, "Visualization and deep-learning-based malware variant detection using OpCode-level features," *Future Gener. Comput. Syst.*, vol. 125, pp. 314–323, Dec. 2021, doi: 10.1016/j.future.2021.06.032.

[36] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138, doi: 10.1016/j.comnet.2020.107138.

[37] S. Hosseini, A. E. Nezhad, and H. Seilani, "Android malware classification using convolutional neural network and LSTM," *J. Comput. Virology Hacking Techn.*, vol. 17, pp. 1–12, Apr. 2021, doi: 10.1007/s11416-021-00385-z.

[38] N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "A deep dive inside DREBIN: An explorative analysis beyond Android malware detection scores," *ACM Trans. Privacy Secur.*, vol. 25, no. 2, pp. 1–28, May 2022, doi: 10.1145/3503463.

[39] H. Bai, N. Xie, X. Di, and Q. Ye, "FAMD: A fast multifeature Android malware detection framework, design, and implementation," *IEEE Access*, vol. 8, pp. 194729–194740, 2020, doi: 10.1109/ACCESS.2020.3033026.

[40] L. D. Coronado-De-Alba, A. Rodríguez-Mota, and P. J. Escamilla-Ambrosio, "Feature selection and ensemble of classifiers for Android malware detection," in *Proc. 8th IEEE Latin-Amer. Conf. Commun. (LATINCOM)*, Nov. 2016, pp. 1–6, doi: 10.1109/LATINCOM.2016.7811605.

[41] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha, "Two anatomists are better than one—dual-level Android malware detection," *Symmetry*, vol. 12, no. 7, p. 1128, Jul. 2020, doi: 10.3390/sym12071128.

[42] N. Potha, V. Kouliaridis, and G. Kambourakis, "An extrinsic random-based ensemble approach for Android malware detection," *Connection Sci.*, vol. 33, no. 4, pp. 1077–1093, Oct. 2021, doi: 10.1080/09540091.2020.1853056.

[43] N. A. Sarah, F. Y. Rifat, M. S. Hossain, and H. S. Narman, "An efficient Android malware prediction using ensemble machine learning algorithms," *Proc. Comput. Sci.*, vol. 191, pp. 184–191, 2021, doi: 10.1016/j.procs.2021.07.023.

[44] M. H. L. Louk and B. A. Tama, "Tree-based classifier ensembles for PE malware analysis: A performance revisit," *Algorithms*, vol. 15, no. 9, p. 332, Sep. 2022, doi: 10.3390/a15090332.

[45] F. M. Alotaibi and Fawad, "A multifaceted deep generative adversarial networks model for mobile malware detection," *Appl. Sci.*, vol. 12, no. 19, p. 9403, Sep. 2022, doi: 10.3390/app12199403.

[46] M. S. Rana and A. H. Sung, "Evaluation of advanced ensemble learning techniques for Android malware detection," *Vietnam J. Comput. Sci.*, vol. 7, no. 2, pp. 145–159, May 2020, doi: 10.1142/S2196888820500086.

[47] D. G. Zill, *Advanced Engineering Mathematics*. Burlington, MA, USA: Jones & Bartlett Publishers, 2020.

[48] N. Spolaôr, E. A. Cherman, M. C. Monard, and H. D. Lee, "A comparison of multi-label feature selection methods using the problem transformation approach," *Electron. Notes Theor. Comput. Sci.*, vol. 292, pp. 135–151, Mar. 2013, doi: 10.1016/j.entcs.2013.02.010.

[49] O. Irsoy and E. Alpaydın, "Unsupervised feature extraction with autoencoder trees," *Neurocomputing*, vol. 258, pp. 63–73, Oct. 2017, doi: 10.1016/j.neucom.2017.02.075.

[50] W. H. L. Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Autoencoders," *Mach. Learn.*, vol. 15, pp. 193–208, Nov. 2019, doi: 10.1016/B978-0-12-815739-8.00011-0.

[51] M. N. Al-Andoli, S. C. Tan, and W. P. Cheah, "Distributed parallel deep learning with a hybrid backpropagation-particle swarm optimization for community detection in large complex networks," *Inf. Sci.*, vol. 600, pp. 94–117, Jul. 2022, doi: 10.1016/j.ins.2022.03.053.

[52] K. E. Parsopoulos and M. N. Vrahatis, "On the computation of all global minimizers through particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 211–224, Jun. 2004, doi: 10.1109/TEVC.2004.826076.

[53] A. Ismail, D.-S. Jeng, and L. L. Zhang, "An optimised product-unit neural network with a novel PSO–BP hybrid training algorithm: Applications to load–deformation analysis of axially loaded piles," *Eng. Appl. Artif. Intell.*, vol. 26, no. 10, pp. 2305–2314, Nov. 2013, doi: 10.1016/j.engappai.2013.04.007.

[54] V. Kelefouras, A. Kritikakou, and C. Goutis, "A matrix–matrix multiplication methodology for single/multi-core architectures using SIMD," *J. Supercomput.*, vol. 68, no. 3, pp. 1418–1440, Jun. 2014, doi: 10.1007/s11227-014-1098-9.

[55] P. Moritz, R. Nishihara, S. Wang, and A. Tumanov, "Ray: A distributed framework for emerging AI applications," in *Proc. 13th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2018, pp. 561–577.

[56] Z. Zheng, Y. Cai, and Y. Li, "Oversampling method for imbalanced classification," *Comput. Inform.*, vol. 34, no. 5, pp. 1017–1037, 2016.

[57] R. M. Sharma and C. P. Agrawal, "A BPSO and deep learning based hybrid approach for Android feature selection and malware detection," in *Proc. IEEE 11th Int. Conf. Commun. Syst. Netw. Technol. (CSNT)*, Apr. 2022, pp. 628–634, doi: 10.1109/CSNT54456.2022.9787671.

[58] S. Naick, P. Bethapudi, and S. P. R. Reddy, "Malware detection in Android mobile devices by applying swarm intelligence optimization and machine learning for API calls," *Int. J. Intell. Syst. Appl. Eng.*, vol. 10, no. 3S, pp. 67–74, 2022.

[59] 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, doi: 10.3390/su142114406.

MOHAMMED NASSER AL-ANDOLI received the B.Sc. degree in computer information systems from Mutah University, Jordan, in 2011, the M.Sc. degree in computer science from the Jordan University of Science and Technology, Jordan, in 2016, and the Ph.D. degree in information technology from Multimedia University (MMU), Malaysia, in 2022. He is currently a Postdoctoral Researcher with MMU. His main research interests include malware analysis, complex network analysis, machine learning, high-performance computing, deep learning, and parallel computing.

KOK SWEE SIM (Senior Member, IEEE) is currently a Professor with Multimedia University, Bukit Beruang, Melaka, Malaysia. He is also working closely with various local and overseas institutions and hospitals. He has filed more than 18 patents and 70 copyrights. He has received many international and local awards. He was a recipient of the Japan Society for the Promotion of Science (JSPS) Fellowship, Japan, in 2018. He received the Top Research Scientists Malaysia (TRSM) from Academic Science Malaysia, in 2014, and the Korean Innovation and Special Awards in 2013, 2014, and 2015. He also received the TM Kristal Award and the International Championships of World Summit on the Information Society (WSIS) Prizes, in 2017, 2018, 2019, 2020, and 2021.

SHING CHIANG TAN received the B.Tech. (Hons.) and M.Sc. (Eng.) degrees from Universiti Sains Malaysia, Malaysia, in 1999 and 2002, respectively, and the Ph.D. degree from Multimedia University, Malacca, Malaysia, in 2008. He is currently a Professor with the Faculty of Information Science and Technology, Multimedia University. His current research interests include computational intelligence, deep learning and its applications, data classification, condition monitoring, fault detection and diagnosis, stroke rehabilitation, and biomedical disease classification and optimization. He was a recipient of the Matsumae International Foundation Fellowship, Japan, in 2010.

PEY YUN GOH (Senior Member, IEEE) received the B.I.T. degree (Hons.) majoring in business information systems, in 2004, the M.Phil. degree in management, in 2007, and the Ph.D. degree in IT, in 2018. Her research field changes from management to IT in order to support her career interest in IT. She is a Lecturer with the Faculty of Information Science and Technology, Multimedia University (MMU), Malaysia. She is currently attached to an artificial intelligent cluster, which is one of the special interest groups within FIST. Her research interests include knowledge discovery, neural networks, pattern recognition, soft computing, and adversarial machine learning.

CHEE PENG LIM received the Ph.D. degree from the University of Sheffield, U.K., in 1997. He is currently a Professor at Deakin University, Australia. He has published more than 550 technical papers in books, international journals, and conference proceedings. His research interests include computational intelligence, data analytics, pattern classification, and multi-objective optimization.