Penerbit Akademia Baru

Journal of Advanced Research Design

Journal homepage:
https://akademiabaru.com/submit/index.php/ard
ISSN: 2289-7984

Enhanced Cybersecurity Efficiency and Android Malware Classification towards Carbon Emission Reduction: Two Phase Ensemble Approach

Xuxu[1], Kah Hou Teng[1,2,*], Botao Wang[3], Zhuolin[3], Swee Pin Yeap[2,4], Andy Shaw[5]

[1] Faculty of Engineering, Technology and Built Environment, UCSI university, Kuala Lumpur, 56000, Malaysia
[2] UCSI-Cheras Low Carbon Innovation Hub Research Consortium, Kuala Lumpur, Malaysia
[3] Information Communication Company of Faculty of Engineering Technology & Information Communication Company of Hubei Electric Power Company STATE GRID Corporation of China Wuhan, China
[4] Department of Chemical and Petroleum Engineering, Faculty of Engineering, Technology and Built Environment, UCSI University, 56000, Kuala Lumpur, Malaysia
[5] Built Environment and Sustainable Technologies Research Institute, Liverpool John Moores University, United Kingdom

ARTICLE INFO

Article history:
Received 25 March 2025
Received in revised form 5 June 2025
Accepted 10 September 2025
Available online 25 September 2025

Keywords:
Android Malware Classification, Ensemble, Genetic Algorithm, Machine Learning, SDG

ABSTRACT

The environmental impact of cybersecurity, especially on the agenda of energy consumption and carbon emissions, is becoming one of the concerns in the cybersecurity industry. Automatic Android classification plays a vital role in combating the rapidly growing number of Android malware variants. This paper describes a highly accurate ensemble classification approach for detecting malicious Android apps. The design of this two-phase ensemble considers diversity as a key aspect. In the first phase, a large number of discriminative features for classification are extracted from Android application package (APK) files, and a Genetic Algorithm(GA) based feature subset selection procedure is applied on different types of base classifiers. In the second phase, an initial pool of classifiers is constructed by varying parameters of base classifiers, and a heuristic search process is conducted aiming at pruning the learning models in the initial pool. The results show that evaluation with 1554 malware apps and 2400 free popular apps reported a detection accuracy of 97.6% and an ROC curve (AUC) value of 99.5% that is better than the existing static analysis-based method. The integration of these technologies into malware detection processes not only bolsters security but also supports environmental sustainability in information technology (IT) practices. These actions would drive green IT towards sustainable development goals (SDG).

1. Introduction

Cybersecurity is having an increasing impact on the environment, with rising energy use and carbon emissions being a major concern throughout the nation . Digital grid applications, which rely on cloud platforms and other mobile internet technologies, are one of the green ways to address this problem [1]. They greatly increase IT efficiency and reduce carbon emissions. However, these technologies face challenging security issues. The integration of several research studies on machine learning-aided methods, deep neural networks, and optimised integrated learning to achieve higher efficiency and accuracy in malware classification is applied. The growth of Android malware necessitates the development of efficient detection strategies. The contribution of machine learning

*Corresponding author.
E-mail address: TengKH@ucsiuniversity.edu.my

and deep learning techniques to Android malware classification is increasing, which organisational IT departments can use to improve green IT practices [2]. Milosevic, Dehghantanha, and Choo introduce Machine Learning Aided Systems for Android Malware Classification that demonstrate accurate static Android malware analysis and effective computation, ultimately reducing energy use in organisational IT operations [3] . The efficiency of Deep Neural Networks for Android app classification is proven, indicating a reduction of the necessary computational resources [4]. Taha and Barukab propose an optimised ensemble learning technique using genetic algorithms that enhances Android malware detection accuracy but reduces computational resources due to high malware detection accuracy [5]. Better green IT practices can be made possible by these computational methods, which will make the digital grid IT environment more sustainable [6].

The rapid growth in Android platform usage over the past several years has led to an increase in targeting these devices by malware authors. Android-targeted malware now accounts for the vast majority (97%) of the mobile threat landscape. Android malware is adapting and evolving, embracing more sophisticated tactics to target users. Consequently, recent studies have proposed many methods for detecting Android malware using machine learning techniques. Malicious and benign Android apps are represented by vectors of features, which are obtained by static analysis or dynamic execution of apps. Static features can be extracted directly from intermediate code representations obtained through decompiling the Android application package (APK) file. Dynamic features are collected by monitoring the runtime behaviours of the apps [7]. These features are used to train classifiers that are able to learn a generalised description of Android malware. By applying this knowledge in the detection phase, an unknown instance of Android malware can be classified as malicious or benign. However, the performance of existing Android classification approaches is still not satisfactory [8].

Diversity can usually be introduced into the classifier in several ways: (1) manipulate the data samples for each ensemble member, (2) use different feature subsets for each classifier in the training of base classifiers, and (3) choose different types of classifiers with variant parameters as building the initial pool of base classifiers. Thus, the task of building an effective ensemble to improve the detection of Android malware can be broken down into two challenging research questions: (1) How to find a set of feature subsets for each base classifier so as to ensure high individual accuracy and high diversity among those base classifiers? and (2) How to build a good performance (i.e., accuracy) ensemble system from a large collection of diverse base classifiers obtained from the first stage?

This paper introduced an innovative two-phase ensemble approach and showed its superiority in the malware detection tasks to mitigate the detection of Android malware. In an empirical evaluation on a total of 1554 Android malware samples, this approach is shown to be highly effective, enabling a detection accuracy of 97.6% and an area under the ROC curve (AUC) value of 99.5%. It offers hope for developing fast and scalable tools in classifying a large number of Android malware variants. Furthermore, this paper aims to contribute by providing a comprehensive experimental study to compare various data mining techniques (different individual classifiers and well-known ensembles) with the paper-proposed ensemble for malware detection. A set of 2,400 benign samples and 1,554 malicious were extracted from Android apps. Following on this, a two-phase ensemble approach for Android malware classification is incorporated. In phase I, It proposed a novel genetic algorithm (GA)-based feature subset selection method for each base classifier. On top of that, a pool of classifiers by varying parameters of base classifiers to obtain an optimal pruned ensemble for efficient malware classification was constructed by a heuristic ensemble selection algorithm in phase II. Finally, the two-phase ensemble approach enhances overall classification performance in terms of accuracy,

receiver operating characteristic (ROC) curves, and area under the ROC curve (AUC) demonstrated in this paper.

2. Literature Review

There have been intensive works in applying machine learning techniques to the malware classification problem. Shhadat *et al.* propose a more enhanced feature set is proposed using random forests to reduce the number of features, and an improvement in accuracy is achieved by applying several machine learning algorithms on the benchmark dataset [9]. Binary Classification with Decision Trees, Multiple Classification with Random Forests, and Binary Classification with Bernoulli's Plain Bayes, which achieved an accuracy of more than 91%. Li *et al.* proposed a meta-feature mining algorithm based on MetaNET, which can discover potential relationships between samples belonging to the same category and use meta-features to identify complex Android malware with excellent robustness and stability [10]. Kolter and Maloof extracted byte sequences from executables, converted them into n-grams, and trained several classifiers [11].

On the other hand, several machine learning-based approaches have been proposed to detect Android malware. Aslan *et al.* proposed a hybrid architecture that integrates two extensive pre-trained network models in an optimised manner, which was tested on three datasets, Malimg, Microsoft BIG 2015 and Malevis, through data collection, designing a deep neural network architecture, training the proposed deep neural network architecture, and evaluating the trained deep neural network [12]. Shabtai *et al.* trained machine learning models using features of permission usage in Android apps [13] to evaluate their models by measuring the true positive ratio, accuracy and the area under the ROC curve (AUC) for different classifiers. Vinayakumar, R, *et al.* proposed ScaleMalNet, a scalable deep learning network architecture for malware detection, which is able to leverage big data techniques to process large numbers of malware samples, classify executables as either malware or legitimate using static and dynamic analysis, and categorise malware executables into corresponding malware families [14]. The architecture is capable of processing large numbers of malware samples using big data techniques, classifying executables into malware or legitimate files using static and dynamic analysis, and categorising malware executables into corresponding malware families. Xu *et al.* propose a dynamic-static mixed-mode malicious webpage detection system, which adopts a secondary cascade detection method. In the rule-based classifier attribute extraction, the idea of associated text tracking and merging is applied to significantly improve the accuracy of the static detection algorithm; the designed lightweight virtual machine is used to replace the traditional system-level virtualised detection environment, which makes the dynamic detection system have a higher task throughput rate [15]. Peiravian *et al.* extracted a heterogeneous feature set and processed each feature independently using multiple kernel learning algorithms [16]. Similarly, Gascon *et al.* extracted function call graphs from Android apps and mapped call graphs to a graph kernel. A support vector machine (SVM) was then trained to distinguish between malware and benign apps. Most of these works have trained a single or a set of machine learning algorithms. One machine learning model, which performed best against a predetermined set of criteria, was then chosen to conduct classification [17].

On top of single-model methods, several researchers have examined ensemble methods, which combine a collection of training models in malware detection. The main idea of an ensemble approach is to combine a set of weak classifiers to obtain a better composite classifier. Guo *et al.* proposed a malware classification approach based on an ensemble classifier. To choose the best classification features, the information-gain feature selection method was conducted on the byte n-gram-based features. Then, a probabilistic neural network (PNN) was applied to construct an

individual classifier for detection. Finally, the D-S theory of evidence was used to combine the contribution of each individual classifier to make a final decision [18]. Ye *et al.* proposed an interpretable string-based malware detection system, SBMDS, to classify file samples and predict the exact types of malware. SBMDS adapted an SVM ensemble with a bagging technique to improve the system's effectiveness and efficiency. They achieved an accuracy of 93% that outperformed other classifiers, such as a single SVM, a Naive Bayes ensemble with bagging, and a J4.8 version of a decision tree ensemble with bagging [19]. In addition, Sami *et al.* employed a static approach, where API calls were extracted from binary files and trained an ensemble of Random Forests [19]. Menahem *et al.* performed a comparative study on ensemble methods for malware detection by combining five different base classifiers. The following combining algorithms were examined: majority voting, distribution summation, Naive-Bayes combination, Bayesian combination, performance weighting, stacking, and Troika [20]. The authors aimed to find the best ensemble method for detecting malware in terms of accuracy, AUC, and execution time. Up to date, numerous ensemble-based approaches have been executed in Android malware detection by involving weighted predictions of classifiers and generating base learners from main classifiers. Three types of combination schemes (i.e., majority voting, stacking, and variant stacking) were applied to construct the ensemble system [21].

3. Methodology

3.1 Feature Extraction and Dataset

The dataset used for this study consists of 1554 Android Malware samples and 2400 benign Android apps in Android application package (APK) format. These Android malware samples are collected from three different sources: Contagio Mobile, the Android Malware Genome Project and sharing from antivirus vendors. These sources were selected to ensure diversity and representativeness of malware variants across different time periods and attack types. To validate authenticity, the malware samples were cross-verified using VirusTotal, and only those with consistent family classifications by at least two out of three major antivirus engines (AVG, ESET NOD32, and Bitdefender) were retained. Malware family names were assigned based on majority agreement among these engines. This dataset covers 107 distinct Android malware families, spanning a decade from 2011 to 2021.

For the benign class, 2400 samples were selected from the top 100 free applications across 12 categories in the Google Play Store (April 2021). To further ensure their benign nature, all applications were re-analysed using VirusTotal, and only those with no malware flags were included. Feature extraction was automated using custom Python scripts built on the Androguard library. The extraction process followed a three-stage pipeline: (1) APKs were decompressed to access manifest files and resource folders; (2) permission declarations and component configurations were parsed from the manifest; (3) assembly code (smali files) was scanned to identify sensitive APIs, shell commands, embedded URLs, and suspicious strings indicative of malicious behaviour. Additionally, structural features such as file sizes, number of methods, presence of native binaries, and embedded archives were extracted.

Feature selection emphasised indicators empirically linked to malicious behaviour and commonly used by human malware analysts. These criteria were derived from recent malware analysis reports, security whitepapers, and manual inspection of known malware families. Ultimately, 252 discriminative features were selected for modelling, balancing comprehensiveness with redundancy avoidance.

Table 1
A List of Typical Extracted Features

Category	Feature	Category	Feature
	Runtime.exec()		RECEIVE_BOOT_COMPLETED
	GetDeviceId		READ_SMS
Risky API	GetLastKnownLocation()	Requested	ACCESS_COARSE_LOCATION
	GetDeclaredMethod()	permission	READ_LOGS
	SmsMessage.createFromPdu		READ_HISTORY_BOOKMARKS
Native code	Su, sh, chmod, chown, ps	Statistical feature	Size of apk, presence of zip
Native lib Loading	System.loadLibrary ()	Suspicious string	Rageagainstthecage, GingerBreak
	Android.os.Build.MODEL		Ration of total valid method names
VM detection	Settings.Secure.getString	Obfuscation	MessageDigest.getInstance
Use reflection	Reflect.Method.invoke		MessageDigest.digest

An ARFF file is built for the use of Weka software with the extracted data, as shown in Table 1 [22]. All attributes have binary values: 'y' and 'n', indicating the existence or absence of the feature, respectively. There are 189 kinds of permissions, including 148 Android system permissions and 41 custom-defined permissions, which are all included in the feature set and have been found in the malware dataset. The rest of the feature set consists of potential risky APIs, shell commands, and attributes of the APK file, among others. Table 1 lists some of these extracted features as examples. To better mitigate mobile malware threats, It made the Android malware samples used in this work and all extracted feature sets available to the research community on GitHub.

3.2 Ensemble and Diversity

An ensemble is used to improve the performance of Android malware detection in terms of accuracy or other measurable merits (e.g., ROC, AUC). Both theoretical and empirical research has shown that ensemble methods can generate more accurate classification results than individual classifiers. To promote the diversity of each individual model, several methods were employed to achieve this goal by combining them into an ensemble. Diversity can be naturally obtained by using different types of base classifiers. The intuition is that each base classifier has an explicit or implicit bias, leading them to prefer certain generalisations over others.

The use of varied feature subsets by each ensemble member helps enhance diversity. Additionally, Diversity is introduced into the ensemble by identifying a subset of features for each base classifier, thus encouraging divergence between them. The core idea is that different feature subsets offer distinct perspectives on the data, resulting in diverse individual learners. This approach also helps reduce the dimensionality of base classifiers and lowers the overall computational complexity of the ensemble. Unlike traditional feature selection algorithms that focus solely on optimizing accuracy, in ensemble methods, both accuracy and diversity are crucial considerations. To address the first research question raised in section 1, It proposes a novel feature subset selection method based on genetic algorithm (GA) for each base classifier.

Another method for creating diverse individual learners is by varying the parameter settings of the base classifiers. Hence, the third approach to generating diversity involves building a pool of training models using the base classifiers from the previous step, each trained with different parameter configurations. For example, J48 was trained by adjusting parameters such as confidence factor and minimum number of instances per leaf. Once this pool of trained models is created, instead of combining all of them, It selects a subset of models to form the ensemble. This selection process is crucial for two main reasons: efficiency and predictive performance. A smaller ensemble

reduces computational complexity compared to a larger one, and empirical studies by Dietterich suggest that a pruned ensemble may be more accurate than the original [23]. To address the second research question raised in section 1, an ensemble pruning approach based on the method proposed by Caruana *et al.* [24].

In this paper, It used the divergence metric, which is an intuitive measure of diversity between a pair of classifiers, to evaluate the diversity among base classifiers. Consider two classifiers, Ci and Cj, and a 2x2 table that summarises their outputs as shown in Table 2. The entries in the table are the probabilities for the pair of correct or incorrect outputs of classifiers. For instance, the N10 indicates the number of instances in which classifier Ci has correctly classified the instances, but Cj has misclassified these instances.

Table 2
Relationship between a pair of classifiers

	D_{k} correct(1)	D_{k} wrong(0)
D_{i} correct(1)	N^{11}	N^{10}
D_{i} wrong(0)	N^{01}	N^{00}
Total, $N = N^{11} + N^{10} + N^{01} + N^{00}$		

The disagreement between two classifiers is measured by Eq. (1)：

$$Dis_{i,j} = \frac{N^{01}+N^{10}}{N^{11}+N^{10}+N^{01}+N^{00}} \quad (1)$$

This measure is equal to the probability that the two classifiers will disagree on their decisions. For an ensemble consisting of L base classifiers, it will generate $\frac{L(L-1)}{2}$ pairwise diversity values. The disagreement diversity among the whole set of classifiers is then defined as an average over all the pairs of disagreement below Eq. (2):

$$Dis = \frac{2}{M(M-1)\sum_{i=1}^{M}\sum_{j=i+1}^{M} dis_{i,j}} \quad (2)$$

The diversity increases with increasing values of the disagreement measure in the range from 0 to 1. In GA-based Feature Selection, it is used as a component of the fitness function guiding the process of feature subset selection for individual classifiers.

3.3 System Architecture

As shown in Figure 1, it is describe the workflow of the proposed ensemble classification approach, which is composed of two major phases: (1) the creation of a set of optimal feature subsets for individual classifiers, and (2) the construction of an optimal ensemble to yield the final prediction. In the first phase, a large number of discriminative features for classification are extracted from APK files, followed by the application of a GA-based feature subset selection on different types of base classifiers. In the second phase, an initial pool of classifiers is constructed by varying the parameters of base classifiers. Subsequently, a heuristic search process is conducted, aiming at pruning the learning models in the initial pool.

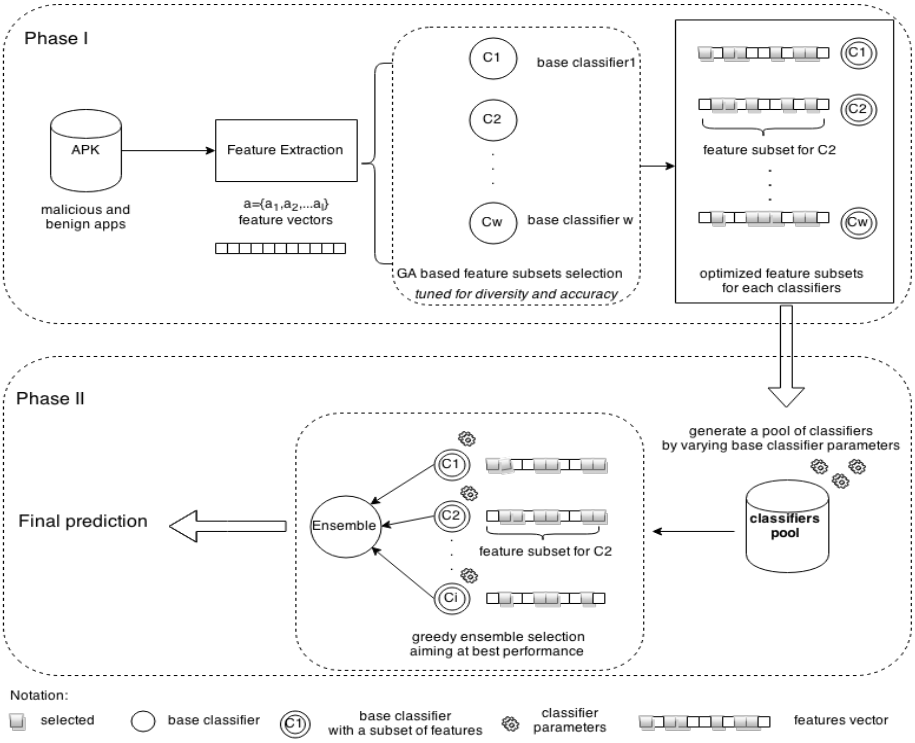

Fig. 1. The System Overview Architecture

3.4 GA-based Feature Selection

The primary purpose of feature selection is to design a more compact classifier with as little performance degradation as possible. The problem of feature selection becomes even more cumbersome in the context of ensemble because a set of subsets $S = \{S_1, S_2, \ldots, S_k\}$, ($S_i \subsetneq W$, where W is a set with n features), is to be chosen instead of a single subset.

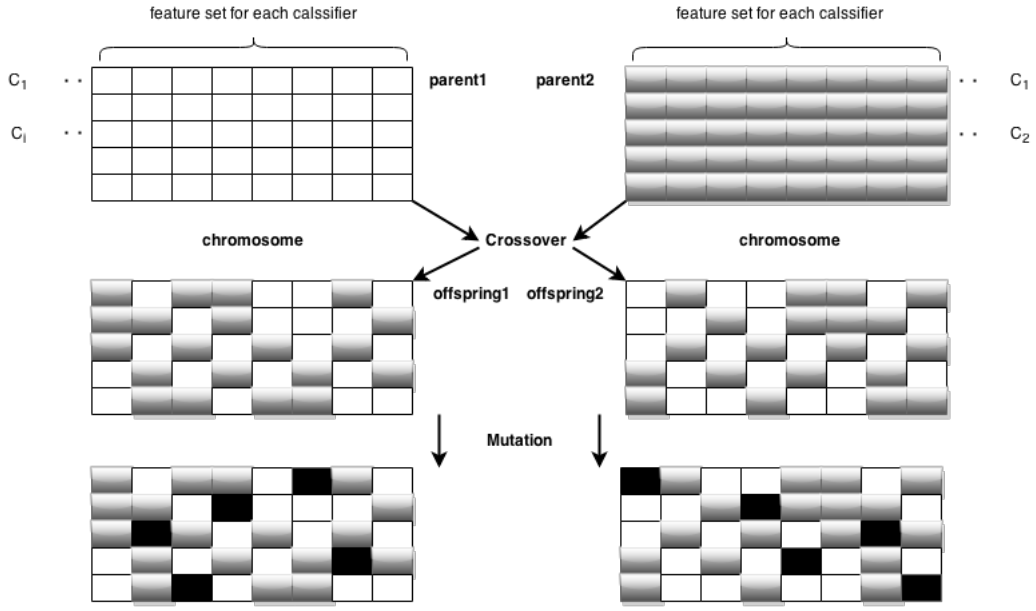

Fig. 2. GA based feature subsets selection

Figure 2 illustrates the process of ensemble feature subset selection. Feature subset selection can be considered a multi-criteria optimisation problem in a vast search space. The criteria to be optimised include the classification accuracy of each classifier and the diversity among individual classifiers. Genetic algorithms [17] offer a particularly effective approach to these kinds of multi-criteria optimisation challenges in high-dimensional search spaces. Therefore, this paper proposes a GA-based method for feature subset selection. Genetic algorithms are adaptive search methods inspired by the evolutionary process of biological populations [25]. In GA, a population consists of a set of candidate solutions, with potential solutions encoded as the chromosomes of individuals. A fitness function evaluates the quality of a solution during the evolutionary process. The fitter members from the previous generation and their offspring, created through crossover and mutation processes, compose each succeeding generation. After many generations, the chromosome solution achieving the best fitness value is considered the optimal solution.

Algorithm 1: FSS_GA_BC(G,p,C)

Input: G: number of generations; p: population size; C: number of classifiers
Output: The largest element in the set

1 Let P_g denote the population in generation g, and let $P_g.i$ be *ith* classifier of that population
2 $P_0 \longleftarrow$ initialize population
3 **for** $g \leftarrow 1$ ***to*** G **do**
4 **for** $i \leftarrow 0$ ***to*** p **do**
5 compute: $Fitness_i = W_a \times \frac{\sum_1^n accuracy_i(P_g.i)}{n} + W_d \times dis(P_g.i)$
6 **repeat**
7 select two parents P_1 and P_2 from P
8 crossover the pair with probability p_c
9 mutate the offspring
10 **until** *p of offspring have been created*
11 replace the current population and update P_{g+1}
12 **return** *the best chromosome*

Fig. 3. Algorithm FSS_GA_BC

The genetic algorithm used for feature subset selection for individual classifiers is similar to a standard GA. Figure 3 shown an outline of the proposed algorithm, FSS_GA_BC (GA based Feature Subset Selection for base classifiers) . Initialize population by randomly choosing the number of features included in each feature subset. Following that, the base classifier Ci, which is represented as i th row in a chromosome, the size of each feature subset Ni is independently chosen from a uniform distribution between 1 and the number of features extracted in the malware dataset. Ni are randomly selected and set them to the value 1, which means that the corresponding features are included in the classifier training set. The fitness of each chromosome is calculated in lines 4-5. In this step, after each base classifier evaluating selected feature set, they return the classification accuracies and diversity to the fitness function. After the whole population has been evaluated, in lines 7-9, Firstly selecting the preferred chromosome with a high fitness score, and then conduct the crossover and mutation operations on selected chromosomes with a predefined P_c (probability of crossover) and P_m (probability of mutation), respectively. The evolutionary process will repeat until terminating generation is reached and the final chromosome would be the optimal set of feature subsets for each individual classifier.

3.4.1 Fitness Function

The classification accuracy and diversity are the two criteria used to design for current model’s fitness function. The fitness of a chromosome is defined as Eq. (3):

$$\text{Fitness}_i: W_a \text{ X } \frac{\sum_1^n \text{accuracy}_i(P_g,i)}{n} + (\ W_d \text{ X dis}(P_g, i) \quad (3)$$

The fitness function is weighted by two predefined parameters W_a and W_d for the classification accuracy and diversity among the base classifiers, respectively. Furthermore, accuracy_i specify the classification accuracy of the i th base classifier upon the feature subset, which is calculated over the validation dataset. Diversity is calculated using Equation 2, measuring the differences in predictions among the base classifiers. Therefore, a chromosome that exhibits both high classification accuracy and significant disagreement (diversity) among the entire set of classifiers aiming to yield a high fitness value for high accuracy classification.

3.4.2 Crossover

A crossover operator swaps bits between two chromosomes based on a certain probability, P_c While one-point crossover is inspired by biological processes, it encounters a significant limitation with two-dimensional structured chromosome: the inherent two-dimensional information might be disregarded or lost, as this crossover operator does not account for the matrix-like structure of the chromosomes. The chromosome is a matrix with the dimension of $C \times W$. where each row represents a subset of features for an individual base classifier, and the fitness function is calculated for each row. A uniform crossover operator tailored to chromosome's unique encoding generates offspring from the parent chromosomes using a randomly generated crossover mask, treating every feature as a potential crossover point. Consequently, each row of the offspring contains a blend of genes (features) from the corresponding row of each parent, ensuring a mix of attributes without a fixed number of crossing points, but will be averaged at $\frac{C \times W}{2}$. The key advantage of the uniform crossover is its ability to exchange bits instead of entire segments, facilitating the recombination of features within individual classifiers irrespective of their positions in the matrix.

3.4.3 Mutation

After the crossover process, the chromosomes undergo mutation. The purpose of mutation is to prevent the algorithm from becoming trapped in a local optimum, thereby ensuring diversity within the population. Given the various forms of mutation applicable to different chromosome representations, the approach to mutation also takes into account the two-dimensional structure of the chromosome. For each row of matrix with the dimension of $C \times W$, a random Index generated p, where $1 < p < W$, that represent the location selected to be the mutation point that row. The value of the bit at this locus (changing a 0 to 1 and vice versa) with a predetermined probability of mutation.

3.5 Ensemble Selection

In the Phase II, main two steps are overproduction and selection. An initial pool of classifiers $\varsigma = \{C_1, C_{2,\ldots,} C_n\}$ is constructed by varying control parameters of each base classifier C_i which are obtained from algorithm 1 in Phase I. These classifiers in the initial pool were trained on the training data set T and then a library of classifier models was created. Those classifiers derived from the same base classifiers C_i as a group $= C_i : \{C_{i,1}, C_{i,2,\ldots,} C_{i,k}\}$ where every classifiers share the same feature subsets FSC_i are denoted. The initial pool of the training models consists of either homogeneous or heterogeneous models. Models that derive from the same base classifier by varying classifier control

parameters are homogeneous and models that derive from different base classifier are heterogeneous.

Following that, different combinations of the classifier models tested in the pool so as to identify an optimal subset of classifiers ς^* which achieves the best performance measurement on the validation data set. Various techniques, such as majority voting and weighted voting, can be applied to aggregate the results from each classifier. Majority voting approach was incorporated in this study due to its simplicity and effectiveness, where each classifier predicts a class, and the class receiving the majority of votes is chosen by the ensemble. The task of selecting the optimal ensemble has been identified as an NP-complete problem [20], making an exhaustive search for the best subset of classifiers impractical for ensembles comprising a large number of models. As an alternative, Using a heuristic algorithm, the best integration can be efficiently identified from the pool of available classifiers. Following the strategy of Caruana *et al.* [24], who utilized a simple forward selection method to build a high-quality ensemble from an extensive library–thereby optimizing performance metrics such as accuracy, RMS error, and F-score–implement a modified version of Caruana's method for the ensemble selection process.

Before describing the integrated selection algorithm, the notation that will be used in the rest of the paper is provided. Let $T = \{(x_i, y_i), i = 1,2\ldots, N\}$ denote a set of training samples where each sample consists of a feature vector x_i and class label y_i, y_i=0 or y_i=1. Let $E = \{(x_i, y_i), i = 1,2\ldots, N\}$denote a set of evaluation samples where each sample consists of a feature vector x_i and class labely_i, y_i=0 or y_i=1. Also, Let $H = \{h, t = 1,2,\ldots, T\}$be the set of classifiers of an ensemble, where each classifier $h_i$maps an instance x to a class label,x ,.h,(x)=y.

Algorithm 2: Ensemble Selection

$S \leftarrow \emptyset$
Initialize the candidate set:$C \leftarrow H$
Evaluate the performance measure $f_{FSC_i}(C, h_t)$ for all $h_t \in C$
while $C \neq \emptyset$ **do**
    Select an element $s \in C$ with the maximum measure of $f_{FSC_i}(C, s)$
    Incorporate s into the current solution: $S \leftarrow S \cup \{s\}$
    Update the candidate set C
    Reevaluate the performance measure $f_{FSC_i}(C, h_t)$ for all $h_t \in C$
return S

Fig. 4. Algorithm 2 Ensemble Selection

Figure 4 presents heuristic search process in pseudocode. It starts with an empty set of classifiers ().$(S = \emptyset)$ At each iteration, a new element from the set H is incorporated into the partial solution under construction, until a complete feasible solution is obtained. Every element $h_i(h_i \in H)$ will be evaluated by an evaluation function f_{FSC_i} (h_i) Which measures the contribution of adding each element to the current partial solution. The selection of the next element to be incorporated is determined by the evaluation of all candidate elements according to the evaluation function. The classifier that maximizes the performance of the ensemble on the evaluation dataset will be selected for the sub-ensemble set.

The evaluation function influences the selection of each element in the sub-ensemble and, subsequently, the performance of the final ensemble. Given a sub-ensemble S and a training model h, the evaluation function estimates the benefit of incorporating h into S using an evaluation measure, which is calculated on the evaluation dataset E. The choice of both the measure and the dataset used for evaluation is crucial, as it affects the quality of the evaluation function and, consequently, the quality of the selected ensemble.

In the method, the dataset was randomly divided into a training set T (comprising approximately 85% of the available instances) an evaluation set E. The evaluation function calculates a value for each candidate subset of models according to an evaluation measure. This value is computed based on the predictions made by its models on the evaluation dataset E. For example, f_{FSC_i} (C, h_i) could return the accuracy of a candidate ensemble C on the data set E by combining the decisions of the classifiers with the method of voting. Given that candidate ensemble C consists of k classification models$\{C_{,i}C_{,2,}\ldots C_{,k}\}$, each classifier model has feature subsetFSC_i which are chosen by feature subsets selection algorithm in Phase I. Thus, for any instance (x_i, y_i), in E , only those features in the corresponding feature subset of classifier $C_{,i}$is selected when calculating prediction performance on evaluation data set E.

In the ensemble selection phase, classification accuracy is the most important criterion for guiding the selection procedure. In this paper, Using the root mean square deviation (RMSD), which is a good measure of accuracy, as performance measurement. The measure of RMSE of one classification model $h_i$with respect to the current sub-ensemble C and set of evaluation data set E is defined as follows Eq. (4):

$$\{\mathrm{RMSE}\ (C,\ h)\ = \sqrt{\frac{1}{N|C}\sum_{i=1}^{T}\sum_{j=1}^{N} h_i(x_j - y_j)^2 + \frac{1}{N}\sum_{i=1}^{T} h_{\square}(x_j - y_j)^2\}} \tag{4}$$

4. Experimental Evaluation

4.1 Performance Metrics

Instead of relying solely on classification accuracy as the evaluation criterion, It employs a set of evaluation metrics, including receiver operating characteristics (ROC) graphs, F-measure, and area under the curve (AUC), to assess the performance of the proposed ensemble for Android malware classification. The ROC (Receiver Operating Characteristic) curve is a graph produced by plotting the true positive rate against the false positive rate for a binary classifier as its discrimination threshold varies. The use of the receiver operating characteristic (ROC) curve allows for a visualisation of the performance of a classifier, depicting the trade-off between the detection rate and the false alarm rate. While the ROC curve is a two-dimensional expression of classifier performance, the area under this curve (AUC) provides a single scalar value for comparing classifiers. The greater the AUC, the better the classifier is at obtaining true positives with fewer false positives.

Another important metric that is isused in this research work is the F-value, which considers false positives and false negatives along with true positives. It incorporates two other measures: precision, which gives us the measure of the classifier's correctness in predicting an actual positive, and recall, which provides us the measure of the percentage of positives identified correctly. Equations (5)-(7) calculate precision and recall.

$$\mathrm{prescision} = \frac{\mathrm{TruePositive}}{\mathrm{TruePositive} + \mathrm{FPostive}} \tag{5}$$

$$\mathrm{recall} = \frac{\mathrm{TruePositive}}{\mathrm{TruePositive} + \mathrm{FalseNegative}} \tag{6}$$

As a composite statistic, f-value is then calculated from precision and recall to summarize the effects of the two types of errors:

$$\mathrm{F-value} = 2\frac{\text{precision recall}}{\text{precision recall}} \tag{7}$$

4.2 Experimental Setting

In this work, It utilized five different inducers as the base classifiers: multilayer perceptron [26], Locally Weight Naïve Bayes (LWL) [27], Naive-Bayes [28], Decision Table [29], and J48 algorithm [30]. It's noteworthy that each inducer belongs to a different family of classifiers. For instance, the multilayer perceptron is a feed-forward artificial neural network classifier, LWL falls under lazy classifiers, Decision Table under rules, Naive-Bayes under Bayesian classifiers, and J48 under decision tree classifiers. The Weka machine learning library served as the source for these base classifiers. To generate 129 models comprising the initial ensemble, It ran each base classifier with various parameters on the training set. The parameters adjusted (while others were left at their default values in Weka) included:

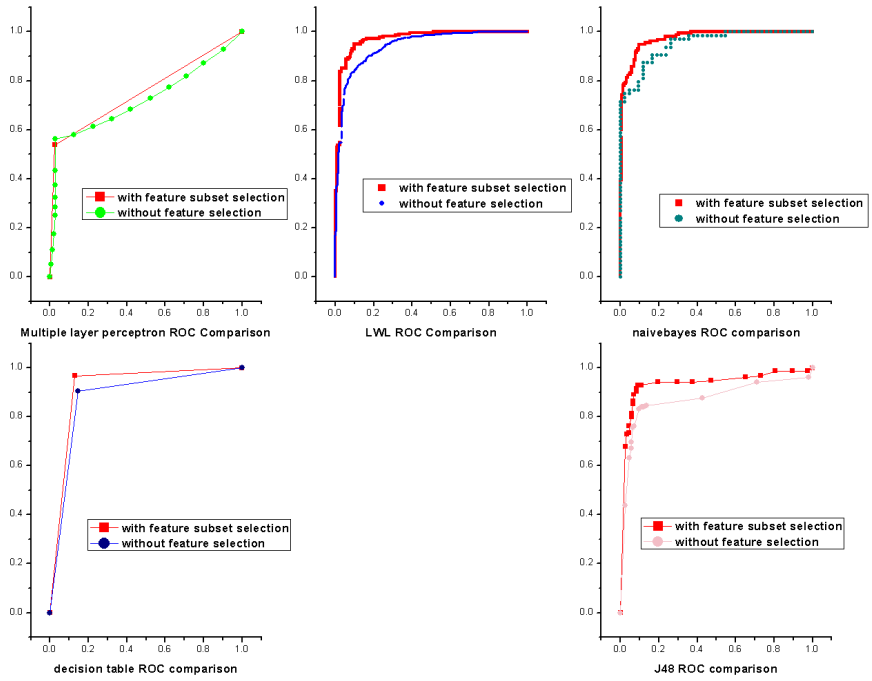

Fig.5. ROC curves for all 5 individual classifiers with or without feature subset selection

The main practices are as follows in Table 3:

Table 3
ROC curves for all 5 individual classifiers Key practices

Inducers	Key practices
Multilayer perceptrons	configured one hidden layer with 8 node values {1, 2, 4, 8, 16, 32, 64, 128}, 4 momentum term values {0.0, 0.2, 0.5, 0.8}, and 2 learning rate values {0.6, 0.9}.
LWL	selected 5 values for the number of neighbors parameter {-1, 0, 1, 2, 3} and 5 values for the weighting method {0, 1, 2, 3, 4}.
Naive Bayes	built one model with default parameters and another with kernel estimation.
Decision Table	used 3 values for the direction of search parameters {0, 1, 2} and 5 values for the number of non-improving nodes to consider before terminating the search {1, 2, 3, 4, 5}.
J48	applied 6 values for the confidence factor {0.2, 0.25, 0.3, 0.4, 0.45, 0.5} and 4 values for the minimum number of instances per leaf {2, 3, 4, 5}.

The population size selection requires balancing search‐space coverage against the computational cost of fitness evaluations. In the experiments, we determined the appropriate population size through empirical tuning. Initially, three population sizes are selected: 50, 100, and 200. For each population size, the genetic algorithm (GA) is executed under a fixed evaluation budget, with the number of generations adjusted such that total fitness evaluations are performed. Upon the completion of each run, both the best fitness value achieved and the total runtime are recorded. Finally, compare the outcomes and select the population size that provides the best trade-off between solution quality and computational cost.

Following the experimental results, and set the population size at 100. focused on tuning the remaining two parameters. Given the complexity of the 252-dimensional feature space and the risk of encountering numerous local optima, deliberately choose a mutation rate of 0.05 to encourage greater population diversity. This adjustment was intended to improve the algorithm's exploratory ability and lower the chances of premature convergence on sub-optimal solutions. For the crossover probability, I selected a widely adopted value of 0.8, which has been shown to work well across different optimisation tasks. the experiments demonstrated that with these settings, the algorithm not only maintains the essential genetic traits of high-quality individuals but also navigates the solution space more effectively, leading to better convergence toward optimal or near-optimal outcomes.

After creating a set of feature subsets for the base classifiers in Phase I, an ensemble selection algorithm was applied to the pool of base classifiers. This process yielded the proposed Android malware classification model. To benchmark the model against the base learning algorithms and various other ensemble methods, It evaluated all classifiers on performance metrics using 10-fold cross-validation, enhancing the reliability of the test results.

Table 4
The performance increased by GA-based feature subset selection

Base classifier	decisionstump		LWL		Naivebayes		Decision table		J48	
num of features	252	97	252	112	252	120	252	85	252	134
Accuracy	0.768	0.775	0.755	0.775	0.838	0.89	0.841	0.881	0.865	0.916
F-value	0.707	0.687	0.691	0.687	0.847	0.881	0.913	0.935	0.861	0.916
Precision	0.956	0.95	0.939	0.950	0.979	0.962	0.864	0.882	0.884	0.916
Recall	0.561	0.537	0.546	0.537	0.746	0.813	0.969	0.993	0.839	0.916
AUC	0.725	0.757	0.941	0.971	0.951	0.974	0.880	0.919	0.872	0.933

Table 5

Comparing the performance of the well known ensembles with the proposed method

Metrics	adaboostM1	bagging	decorate	RandomForest	Phase II ensemble
Accuracy	0.906	0.90	0.923	0.929	0.976
F-value	0.907	0.90	0.923	0.929	0.974
Precision	0.899	0.897	0.923	0.924	0.972
Recall	0.916	0.903	0.923	0.935	0.977
AUC	0.928	0.968	0.977	0.984	0.995

4.3 Experimental Results

In the first experiment, It evaluated the GA-based feature subset selection procedure. The experimental results are summarized in Table 4, presenting the accuracy, F-value, precision, recall, and AUC for each base classifier with and without the GA-based feature subset selection procedure, compared pairwise. The italicized numbers represent the highest values in their respective column groups.It observed that the GA-based feature subset selection significantly improved the classification results compared with the corresponding base classifiers trained solely with all 252 features. Furthermore, an analysis of the ROC curves in Fig. 5 reveals that the results obtained with the GA-based feature subset selection markedly outperform those of the base classifiers trained with all 252 features. This confirms that the GA-based feature subset selection procedure not only reduces the number of selected features but also enhances classification performance.

The second experiment assesses the classification performance of the proposed two-phase ensemble and compares it against baseline ensemble algorithms such as AdaBoostM1 [31] , Bagging [32], DECORATE [33], and Random Forest [34]. Among the baseline algorithms, Bagging, DECORATE, and AdaBoostM1 are well-known ensemble methods that aim to increase accuracy by combining various base inducers into a single classifier. In this paper, the ensemble methods AdaBoostM1, Bagging, and DECORATE utilise the REPTree algorithm as the base classifier. For Random Forests, the number of attributes considered at each node was set to the default value in WEKA software. In contrast to the proposed method, which was trained on selected feature subsets for each base classifier, the baseline ensemble algorithms were trained using all 252 features extracted from the dataset. Table 5 lists the results in terms of accuracy, precision, recall, F-measure, and AUC improvement over the baseline algorithms, with the italicized numbers indicating the highest values in each row. Fig. 6 presents the ROC curves in a comparative manner. The results indicate a superior ROC curve for the proposed two-phase ensemble method compared to the baseline ensembles. Undoubtedly, the performance improvement is attributable to the GA-based feature subset selection method in Phase I, which produces significant diversity among base classifiers. Furthermore, additional improvement is achieved by applying the proposed ensemble selection procedure in Phase II.

For the purpose of the study, a RIGOL-DP832 was used to capture the DELL The PowerEdge R740 server’ s represents the actual energy consumption of 1200W(E_i). As one of the primary parameters of comparison, the energy utilization was captured over 2000 detection sample data points, then iteratively implemented 10,000 times（ C_i）. Use a unique method of evaluating the usage of energy. The formula for this method is as follows Eq. (8):

$$W = \sum E_i \sum T_i \sum C_i \quad (8)$$

- W： Energy Consumption
- E_i: Represents the actual energy consumed
- T_i : The Calculation cycle for a single execution
- C_i : The number of cycles.

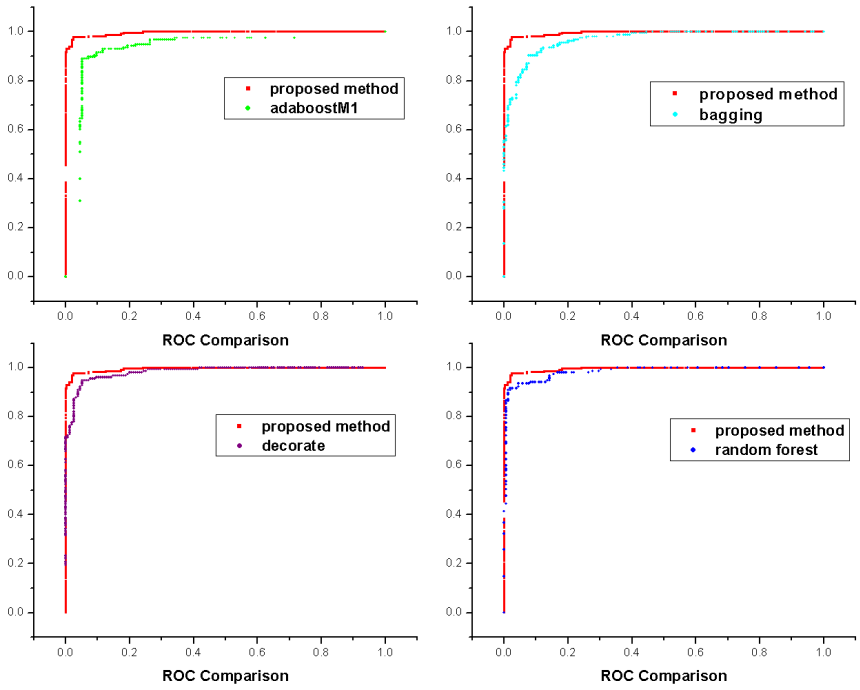

Fig. 6. Classification performance as ROC curve for the method

To compare the performance of the use of energy, two main fields of focus will remain the exact energy utilized and the time used to execute once. The objective is to minimize the extent of these two, thus the overall power usage is minimized. This is especially important in the case of energy-efficient work including data centers and HPC situations. As compared to other techniques used, the present approach demonstrated considerable energy efficiency improvement. The improvement was subsequently reflected in energy used and the required time to exhaust the entire computation cycle. However, it is vital to understand that the implementation of the method could depend on the specific assignment and hardware design. Therefore, it is urgent to conduct an extensive evaluation of the actual position to have a detailed overview of the concept. The absolute energy usage and runtime for each algorithm during one complete training and inference cycle are measured. Table 6 summarizes, for each algorithm–Phase II Ensemble, Random Forest, Bagging, DECORATE, and AdaBoostM1–the mean energy consumed (kWh), its standard deviation over 10 repeated runs, and the mean runtime (s).

Table 6
Comparing the Energy Consumption with the proposed method

	Phase II ensemble	Bagging	Decorate	Random Forest	Adaboost M1
Calculation cycle TI （S）$(\mu \pm \delta)$	6.4 ± 0.8	9.34 ± 0.8	8.7 ± 0.8	8.1 ± 0.8	8.5 ± 0.8
Energy Consumption (KWh)$(\mu \pm \delta)$	21.33 ± 0.8	31.13 ± 0.8	29.2 ± 0.8	27.1 ± 0.8	28.33 ± 0.8

Repeated each experiment 10 times with different random seeds. A paired two-tailed t-test was used to compare the Phase II Ensemble method against each baseline. The significance level was set at 0.05. the method achieved a 21% average improvement in energy efficiency compared to Random Forest. This improvement was statistically significant ($p < 0.01$). The improvements compared to

Bagging, DECORATE, and AdaBoostM1 were also statistically significant. Table 7 provides the confidence intervals for these average energy savings.

Table 7
Paired t-test results and 95% confidence intervals for energy consumption differences

Comparison	Mean Energy	95% CI	p-value
Phase II VS Random Forest	-21.0%	[−24.5, −17.5]	<0.001
Phase II vs Bagging	+31.4%	[+28.0, +34.8]	<0.001
Phase II vs DECORATE	+26.4%	[+23.1, +29.7]	<0.001
Phase II vs AdaBoostM1	+24.7%	[+21.8, +27.6]	<0.001

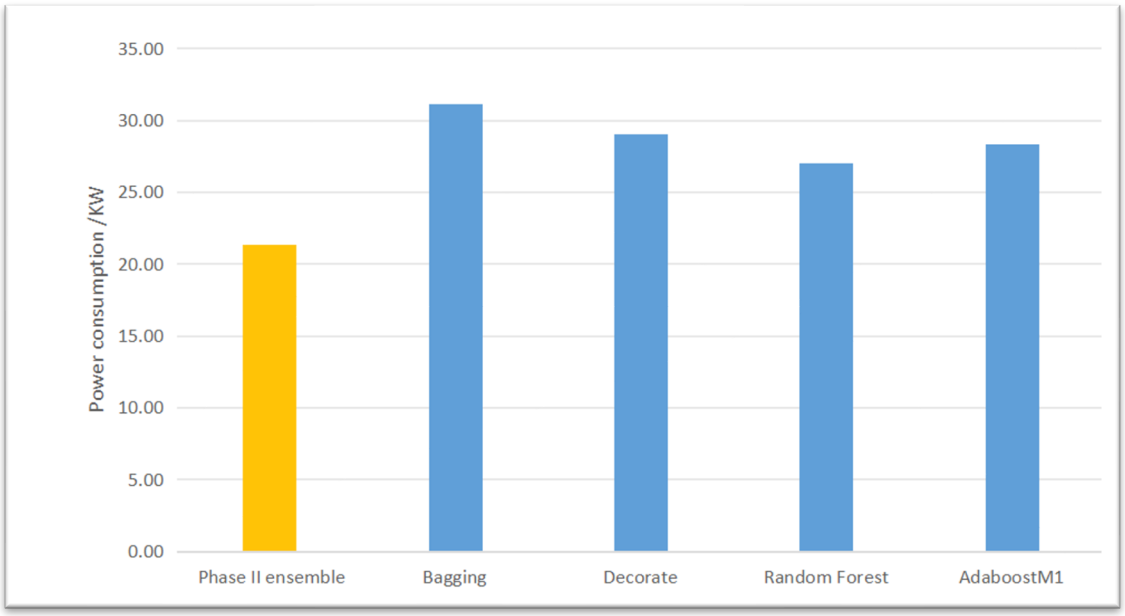

Fig. 7. Energy Consumption for the method and well known ensembles

The results of the energy consumption analysis indicated that the Phase II ensemble method not only enhanced detection accuracy but also required less energy consumption. Compared to the random forest method, the energy consumption of the Phase II ensemble method increased by 21%. It was 31.4% higher than that of the bagging method, 26.4% higher than the decorate method, and 24.7% higher than the adaboostM1 method.

The outcome of the exploration of energy consumption depicts that the ensemble technique of Phase II enhances the level of accuracy in the detection prior to the consumption of more energy. There was an increase of 21% when compared to the method of random forest. This is approximately 31.4% higher, 26.4% higher than decorate, and 24.7% higher when compared to AdaBoostM1. A more comprehensive way of depicting the total energy consumed by varying rather average algorithms can be utilised when conducting similar tasks. This is an essential discovery when choosing the algorithms to take with a higher level of energy consumption. It also suggests the potential to achieve a balance between detection accuracy and energy consumption.

Based on the measurements, the Phase II ensemble consumed 21.33 kWh per full training-inference cycle, compared to 27.10 kWh for the Random Forest baseline, resulting in a per-run energy saving of 5.77 kWh. We can save an estimated 57,000 kWh of energy after 10,000 executions. Based

on a CO_2 emission factor of 0.50 kg CO_2e per kWh, this sum corresponds to a reduction of approximately 28,850 kg of CO_2-equivalent emissions.

When projected to a large-scale deployment scenario, such as one million inferences per month, the Phase II method could potentially save around 5.77 million kWh and avoid approximately 2.885 million kilograms (or about 2,885 metric tonnes) of CO_2e each month. These emission reductions could have a substantial impact on the energy demands of data centres and significantly improve the battery life of edge computing devices. Overall, these results suggest that beyond its accuracy improvements, the Phase II ensemble delivers meaningful energy and carbon savings under the evaluated conditions, although further validation across a broader range of hardware and workloads is advisable.

5. Conclusion

In this paper, the research effort on Android malware classification based on a two-phase ensemble is described. We initially analysed a large collection of Android malware, extracting 252 features that distinguish between two categories of files (benign and malware). Then, a two-phase ensemble is proposed to build a classification model for Android malware. In the first phase, a GA-based feature subset selection is applied to different types of base classifiers. In the second phase, a heuristic-based ensemble selection approach is conducted with the aim of pruning the library of models, which consists of 129 homogeneous or heterogeneous classifiers. Finally, by conducting several experiments, the proposed method is found to be superior to various state-of-the-art ensemble techniques or single inducers available today.

The majority of mobile malware targets the Android platform, so this work focuses on Android malware classification. However, minor changes can adapt the presented method to the iOS platform. Adapting the approach to other platforms, including desktop OS, may thus be an intriguing direction for future work. Moreover, future work may include the involvement of more inducers for base classifiers (only 5 were used in this study), while there are plenty more to choose from.

This paper demonstrates experimentally that machine learning and deep learning techniques are effective in Android malware classification and also focuses on the application of green IT software engineering: better detection accuracy, lower energy efficiency, and the possibility of deploying security applications in cloud platforms or data centres in large-scale digital grids, and the integration of these techniques into the malware detection process strengthens the security and contributes to the environmental sustainability in IT practices.

Acknowledgement

The authors would like to acknowledge the Ministry of Higher Education (MoHE) Malaysia for supporting this project (in part) through Fundamental Research Grant Scheme (FRGS) - project code FRGS/1/2023/TK08/UCSI/02/1 and UCSI-Cheras Low Carbon Innovation Hub Research Consortium for providing the dataset.

References

[1] Anyachebelu, N. C., et al. 2025. “The Role of IoT and Cybersecurity in Sustainable Mining and Materials Processing: A Pathway to Climate Change Mitigation.” *Metallurgical and Materials Engineering* 31 (1): 460–74.

[2] Chee Kel, S., and A. Shahrum Shah. 2024. “Prediction of Monthly Total Sales for a Company Using Deep Learning.” *Journal of Advanced Research Design* 101 (1): 1–20. https://doi.org/10.37934/ard.101.1.120.

[3] Milosevic, N., A. Dehghantanha, and K.-K. R. Choo. 2017. “Machine Learning Aided Android Malware Classification.” *Computers & Electrical Engineering* 61: 266–74. https://doi.org/10.1016/j.compeleceng.2017.02.013.

[4] Nix, R., and J. Zhang. 2017. "Classification of Android Apps and Malware Using Deep Neural Networks." In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. https://doi.org/10.1109/IJCNN.2017.7966078.
[5] Taha, A., and O. Barukab. 2022. "Android Malware Classification Using Optimized Ensemble Learning Based on Genetic Algorithms." *Sustainability* 14 (21): 14406. https://doi.org/10.3390/su142114406.
[6] Faizal, M. A. M., and N. A. A. Salim. 2024. "Green Strategies during Lower Occupancy: The Best Practices of Facilities Management for Energy Optimization in Commercial Office Buildings." *Journal of Advanced Research Design* 117 (1): 57–67. https://doi.org/10.37934/ard.117.1.5767.
[7] Pinheiro Henriques de Araújo, I., et al. 2024. "Antimalware Applied to IoT Malware Detection Based on Softcore Processor Endowed with Authorial Sandbox." *Journal of Computer Virology and Hacking Techniques* 20 (4): 729–49. https://doi.org/10.1007/s11416-024-00526-0.
[8] Abd Razak, N. H., S. Baharom, and G. K. Yi. 2025. "Comparative Study on Code Complexity Metric to Guide Action Selection of Automated GUI Testing Based on Q-Learning Algorithm." *Semarak Engineering Journal* 8 (1): 23–36. https://doi.org/10.37934/sej.8.1.2336b.
[9] Shhadat, I., A. Hayajneh, and Z. A. Al-Sharif. 2020. "The Use of Machine Learning Techniques to Advance the Detection and Classification of Unknown Malware." *Procedia Computer Science* 170: 917–22. https://doi.org/10.1016/j.procs.2020.03.110.
[10] Li, Z., et al. 2024. "metaNet: Interpretable Unknown Mobile Malware Identification with a Novel Meta-Features Mining Algorithm." *Computer Networks* 250: 110563. https://doi.org/10.1016/j.comnet.2024.110563.
[11] Kolter, J. Z., and M. A. Maloof. 2004. "Learning to Detect Malicious Executables in the Wild." In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/1014052.1014105.
[12] Aslan, Ö., and A. A. Yilmaz. 2021. "A New Malware Classification Framework Based on Deep Learning Algorithms." *IEEE Access* 9: 87936–51. https://doi.org/10.1109/ACCESS.2021.3089586.
[13] Shabtai, A., Y. Fledel, and Y. Elovici. 2010. "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning." In *2010 International Conference on Computational Intelligence and Security*. IEEE. https://doi.org/10.1109/CIS.2010.77.
[14] Vinayakumar, R., et al. 2019. "Robust Intelligent Malware Detection Using Deep Learning." *IEEE Access* 7: 46717–38. https://doi.org/10.1109/ACCESS.2019.2906934.
[15] Xu, X., and J.-F. Yu. 2011. "Research of Detection Algorithm Based on Tracking-and-Merging." *Computer Engineering and Design* 32 (4).
[16] Peiravian, N., and X. Zhu. 2013. "Machine Learning for Android Malware Detection Using Permission and API Calls." In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. IEEE. https://doi.org/10.1109/ICTAI.2013.53.
[17] Gascon, H., et al. 2013. "Structural Detection of Android Malware Using Embedded Call Graphs." In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. https://doi.org/10.1145/2517312.2517315.
[18] Guo, W., et al. 2024. "MalOSDF: An Opcode Slice-Based Malware Detection Framework Using Active and Ensemble Learning." *Electronics* 13 (2): 359. https://doi.org/10.3390/electronics13020359.
[19] Ye, Y., et al. 2009. "SBMDS: An Interpretable String-Based Malware Detection System Using SVM Ensemble with Bagging." *Journal in Computer Virology* 5: 283–93. https://doi.org/10.1007/s11416-008-0108-y.
[20] Aswini, A., and P. Vinod. 2014. "Android Malware Analysis Using Ensemble Features." In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. https://doi.org/10.1007/978-3-319-12060-7_20.
[21] Xiong, P., et al. 2014. "Android Malware Detection with Contrasting Permission Patterns." *China Communications* 11 (8): 1–14. https://doi.org/10.1109/CC.2014.6911083.
[22] Hall, M., et al. 2009. "The WEKA Data Mining Software: An Update." *ACM SIGKDD Explorations Newsletter* 11 (1): 10–18. https://doi.org/10.1145/1656274.1656278.
[23] Dietterich, T. G. 2000. "Ensemble Methods in Machine Learning." In *International Workshop on Multiple Classifier Systems*. Springer. https://doi.org/10.1007/3-540-45014-9_1.
[24] Caruana, R., et al. 2004. "Ensemble Selection from Libraries of Models." In *Proceedings of the Twenty-First International Conference on Machine Learning*. https://doi.org/10.1145/1015330.1015432.
[25] Londe, M. A., et al. 2025. "Biased Random-Key Genetic Algorithms: A Review." *European Journal of Operational Research* 321 (1): 1–22. https://doi.org/10.1016/j.ejor.2024.03.030.
[26] Verma, B. 1997. "Fast Training of Multilayer Perceptrons." *IEEE Transactions on Neural Networks* 8 (6): 1314–20. https://doi.org/10.1109/72.641454.
[27] Frank, E., M. Hall, and B. Pfahringer. 2012. "Locally Weighted Naive Bayes." *arXiv* preprint arXiv:1212.2487.
[28] Berrar, D. 2025. "Bayes' Theorem and Naive Bayes Classifier." https://doi.org/10.1016/B978-0-323-95502-7.00118-4.

[29] Achari, A. P. S. K., and R. Sugumar. 2025. "Performance Analysis and Determination of Accuracy Using Machine Learning Techniques for Decision Tree and RNN." In *AIP Conference Proceedings*. AIP Publishing LLC. https://doi.org/10.1063/5.0258588.

[30] Bhargava, N., et al. 2013. "Decision Tree Analysis on J48 Algorithm for Data Mining." *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering* 3 (6).

[31] Kalfountzou, E., et al. 2025. "A Comparative Analysis of Machine Learning Algorithms in Energy Poverty Prediction." *Energies* 18 (5). https://doi.org/10.3390/en18051133.

[32] Nie, Y., et al. 2025. "Bagging Machine Learning Algorithms for Rapid Identification, Classification, Evaluation and Upscaling in Unconventional Reservoir." *Geoenergy Science and Engineering* 246: 213545. https://doi.org/10.1016/j.geoen.2024.213545.

[33] Liang, Y., et al. 2020. "Decorin: An Automatic Method for Plane-Based Decorating." *IEEE Transactions on Visualization and Computer Graphics* 27 (8): 3438–50. https://doi.org/10.1109/TVCG.2020.2972897.

[34] Mallala, B., et al. 2025. "Forecasting Global Sustainable Energy from Renewable Sources Using Random Forest Algorithm." *Results in Engineering* 25: 103789. https://doi.org/10.1016/j.rineng.2024.103789.