International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I8P118

Volume 74 Issue 8, 259-277, August 2026

Original Article

Android Malware Detection via Hybrid Static-Dynamic Analysis and Metaheuristic Feature Selection

Kshamta Chauhan[1*], Ekta Gandotra[2]

[1,2]*Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, Solan, H.P., India.*

[1]*Corresponding Author : kshmta.chauhan19@gmail.com*

Received: 03 March 2026 Revised: 13 July 2026 Accepted: 22 July 2026 Published: 29 August 2026

Abstract - *The rapid proliferation of Android applications has significantly increased the exposure of mobile devices to malware, necessitating accurate and robust detection mechanisms. While hybrid malware analysis that combines static and dynamic features has been widely adopted for Android malware detection, the effectiveness of such systems strongly depends on the selection of discriminative features. In this study, we present a systematic optimization and comparative evaluation of metaheuristic feature selection techniques within a hybrid malware analysis framework. Specifically, three metaheuristic feature selection algorithms, namely particle swarm optimization, genetic algorithm, and bat algorithm, are employed to identify the most relevant and discriminative set of features while minimizing redundancy and boosting the accuracy. An extensive set of experiments is carried out on a self-created dataset of Android applications to assess the effectiveness of the suggested approach. To further validate its effectiveness and generalizability, experiments are also performed on a benchmark dataset of Android malware. Five machine learning and three deep learning algorithms are trained with original features and the features selected using optimization algorithms for static, dynamic, and hybrid analysis approaches. Experimental results demonstrate that metaheuristic-based feature optimization consistently improves detection accuracy compared to non-optimized hybrid feature sets. According to experimental results, a gated recurrent unit constructed with features derived from the bat optimization approach for hybrid analysis gives the highest accuracy of 99.45%.*

Keywords - *Android malware, Hybrid malware analysis, Dynamic malware analysis, Machine Learning, Optimization algorithms, Static malware analysis.*

1. Introduction

Globally, Android is the most popular mobile operating system. Because Android is free and open-source, it has become the most widely used smartphone platform in recent years. It provides outstanding functionalities to customers. In 2025, Android dominated 71% of the smartphone market all over the world as compared to the other Operating Systems (OS) [1]. Mobile applications (apps) are being used for business and personal use, i.e., to do phone calls, sending emails, take pictures, record videos, gaming, use GPS, access e-banking facilities, shop online, conduct information searches, etc [2]. While installing these apps, most users grant permissions without giving them a second thought, which results in lax security [3]. The rising popularity of Android devices has resulted in a rise in malware and cyberattacks. Malware is a piece of code that is designed to damage devices and data, whereas anti-malware software is designed to identify malicious software or apps [4]. Android apps are developed by Android developers and then sent to a third party for verification. App developers, brokers, and vendors must ensure that each app meets all security and privacy requirements [5]. Some newly released apps, even in the Google Play Store, have malware that is initially undetectable when they are released in the Android Market [6]. In 2018, a spyware named Desert Scorpion was found on a chat app called Dardesh Instant App, which was present in the Google Play Store, and users had downloaded this app under the impression that all the apps available on the Play Store are secure [7].

A number of methods and techniques have been developed over time in order to detect and recognize malware attacks. Signature-based detection is the most widely used technique, where the signature of the application is matched with those available in the database. Not finding zero-day malware is a problem with this approach [8, 9]. To deal with this issue, researchers have begun to apply Machine Learning (ML) techniques in the detection and categorization of malware. These methods make use of the features obtained from malware analysis. Malware analysis is a procedure that defines the behavior and functionality of a malicious program. It is of two types: static and dynamic malware analysis. Static

malware analysis allows the detection of malicious apps without executing their code. The approach is rather fast but cannot detect and recognize morphing malware and code obfuscation. Dynamic malware analysis inspects the code's behaviour by running it in the sandbox environment. This method can detect unknown malware, but during the execution process, certain parts of the code remain undetected. To address the issue of both techniques, static and dynamic malware analysis features are integrated and used with Deep Learning (DL) and ML algorithms for the detection and classification of Android malware [10]. Moreover, there are insufficient recent datasets and no publicly accessible benchmarks to assess the proposed base techniques.

Three optimization techniques, namely the Bat Algorithm (BA), the Particle Swarm Optimization (PSO), and the Genetic Algorithm (GA), are the most commonly used search strategies [11]. These techniques have the capability to optimize a set of features as well as search for the optimal hyperparameter values. PSO is a simple algorithm that converges quickly. GA is a popular evolutionary method that can search the whole space well. BA is a newer algorithm that balances exploration and exploitation by using frequency and loudness parameters. In this research, we propose an optimized technique that elevates the accuracy and reliability of Android malware detection. It takes advantage of the hybrid set of features that are taken from both static and dynamic malware analysis. These features are then optimized using three optimization techniques, i.e., PSO, GA, and BA. Afterwards, these features are utilized to build ML models for the detection of Android malware. The main contributions of this study are:

- A binary dataset is constructed using static and dynamic malware analysis and shared publicly on Kaggle.
- Three metaheuristic-based feature selection algorithms, i.e., PSO, GA, and BA, are applied to static, dynamic, and hybrid features to identify the most relevant set of features for malware detection.
- A comparative analysis of ML and DL algorithms trained using an original and optimized set of hybrid features is carried out for Android malware detection.

1.1. Research Questions and Hypothesis

Many Android malware detection frameworks have been suggested in the literature, but most of the studies that have been done so far either use hybrid features without systematically optimizing them or only test one optimization strategy on its own.

There has been insufficient focus on elucidating the comparative efficacy of various metaheuristic algorithms in the context of hybrid feature representations. This study establishes the following research questions and hypotheses to systematically and quantifiably address these gaps.

RQ1: Does metaheuristic-based feature optimization enhance the efficacy of Android malware detection in comparison to the original hybrid feature set?

RQ2: Do optimized hybrid features derived from combined static and dynamic analysis enhance classification performance across ML and DL models?

RQ3: Among PSO, GA, and BA, which optimization technique is most effective for feature selection in Android malware detection?

Based on these research questions, the corresponding hypotheses are defined as follows:

H1: The feature optimization using metaheuristics enhances malware classification performance from optimized hybrid features.

H2: The accuracy and F1-score of various classifiers increase with optimized hybrid feature representations.

H3: The BA performs better than PSO and GA for feature selection tasks in terms of the performance of the developed Android malware detecting system.

The organization of the paper is as follows. Section 2 presents the related work on Android malware identification. The methodology adopted in this study is described in Section 3. The results of the experiment are discussed and examined in section 4. Section 5 presents the discussion, followed by the conclusion and future work in section 6.

2. Related Work

This section highlights the research of malware detection using optimization, ML, and ML strategies in static, dynamic, and hybrid malware analysis.

2.1. Static Malware Analysis

Static malware analysis is a method of detecting dangerous patterns in an app by analysing its code. It decompiles the app source code using disassembly techniques to discover dangerous patterns [12]. This subsection presents the work carried out using static malware analysis for the identification and categorization of Android malware.

In [13], the authors investigated the use of ML methods combined with wrapper-based feature selection techniques to distinguish between malicious and benign applications. Using this technique, authors achieved 98.92% accuracy with 79% minimum dimensionality in feature space. Kumar et al. [14] improved Android IoT devices malware detection by combining the features of ML and blockchain technology. The authors used classification and clustering techniques to extract the malware information, and further used the blockchain to

store that information. With the help of blockchain, malware information is passed to the entire network to improve the rate of detection. According to the experimental results, the suggested framework achieved greater malware detection accuracy with fewer false-positive and false-negative rates.

Zhongru Ren et al. [3] suggested using DL to identify Android IoT malware. The authors resampled the classes.dex file of an Android APK and used DL techniques to train the processed data. The experimental results provided an accuracy of 96%. Almomani et al. [15] employed a hybrid crypto-steganography technique to detect malware concealed in multimedia on Android IoT devices. Experimental results provided high efficiency and good performance. Varma et al. [16] proposed the bat optimization algorithm for Android malware detection. To elevate the malware detection findings, the authors employed a wrapper-based feature selection technique. Eslavath Ravi et al. [17] conducted a comparative analysis of malware detection methods that utilize ML and DL. According to their findings, the DL strategy performed better than conventional ML techniques.

Further, they found that if the DL approach is used with optimization techniques, it enhances the performance of the proposed approach. Razak et al. [11] proposed a bio-inspired algorithm for the feature optimization approach to identify malware attacks. Experimental results revealed that the particle swarm optimization techniques gave the best results for selecting the most appropriate features. Mahesh et al. [18] introduced adaptive red fox optimization using the convolutional neural network. In this paper, the authors worked only on two features, i.e., API calls and permissions, for better feature extraction. The proposed approach has been compared with existing malware detection approaches, and results revealed that the proposed method achieved better values for recall, precision, and f-measure. Lingru Cai et al. [19] had given a model that enhanced the feature selection process and optimized the classifiers. When compared to the weighted unaware method, the proposed weighted method outperformed for all performance metrics. Sulaiman et al. [20] presented the whale optimization method to strengthen Android malware categorization performance. In this work, various ML algorithms were used with the whale optimization technique for selecting features.

Although static malware analysis can quickly analyze the code, it cannot identify malicious applications that employ code obfuscation techniques. So, dynamic malware analysis is employed as a solution to overcome this issue.

2.2. Dynamic Malware Analysis

Dynamic malware analysis examines program behavior by running samples in a runtime environment (an emulator or virtual computer). This section discusses dynamic malware analysis research for detecting and classifying Android malware.

Alzubi et al. [21] introduced an optimization technique for harris hawks to detect Android malware. In this paper, the proposed technique was evaluated on five different datasets to check its robustness and performance, and it achieved the best results in most of them. Smmarwar et al. [22] proposed an efficient and optimized model for Android malware detection for sustainable computing. In this framework, Binary Grey Wolf Optimization (BGWO) was used to select the prominent features for static and dynamic analysis. To boost the efficiency of ensemble learning, different base classifiers were trained using hyperparameters. After the experimental results, binary classification obtained an accuracy of 96.95%, and category classification obtained an accuracy of 83.49%. Altyeb Taha et al. [23] had given a model that optimizes ensemble learning based on a genetic algorithm. A genetic algorithm was utilized to improve the parameters of the Random Forest (RF) algorithm and achieved the best Android malware classification accuracy. After evaluating the proposed method, the results proved that the proposed algorithm was superior than the existing method and achieved 94.15% accuracy. Afonso et al. [24] presented an approach that used dynamic features, API calls, and system calls to assess if an Android application is malicious or benign. The author achieved a detection rate of 96.66% using various ML algorithms. Using DL and dynamic features, a new approach has been introduced for the detection of Android malware by Sihag et al. [25]. This proposed technique was tested against 13,522 Android applications and found to give quite accurate results for malware detection.

Dynamic malware analysis is capable of identifying unknown malware. It is quite efficient and capable of identifying the malicious content, but during execution, some parts of the malicious code remain undetected, and it also takes a lot of resources and time.

2.3. Hybrid Malware Analysis

A single method, whether static or dynamic, is insufficient for accurately categorizing malware and benign samples. The researchers started using an integrated set of features from static and dynamic malware analysis to identify and categorize Android malware in order to get around this restriction.

Fei Tong et al. [26] proposed a hybrid approach for the detection of Android malware. In this paper, the author used the dynamic approach for the collection of runtime system calls and the static method to analyse the collected data. After the collection of the data, the author analysed the normal pattern and malware pattern set and evaluated the performance of the proposed techniques, which achieved a better accuracy rate of 90%. In 2017, Altaher et al [27] presented a hybrid approach for the detection of Android malware based on permission and API calls. They used the combination of ANFIS and PSO to enhance the performance and achieved 89% accuracy as compared to other approaches. Dhalaria et

al. [6] introduced the new hybrid approach, which uses the two databases to identify and categorise malware for Android devices. Their proposed technique achieved the highest accuracy for random forest in both datasets using a hybrid approach, i.e., 98.53% and 90.1%. Kabakus et al. [28] introduced a method called mad4a that made use of the advantages of both static and dynamic analysis. They used the permission and API calls to analyse the data and improved the performance of the proposed technique. Farhan Ullah et al. [29] integrated the byte-level image representation with API call graphs (APGs) for Android malware detection. The proposed method outperformed the latest methods with 99.27% accuracy and makes use of a customized dataset created from the CIC-InvesAndMal2019 dataset. Taher et al. [30] used the hybrid analysis model with an improved Harris-Haws Optimizer (HHO) to optimize the neural network model for the detection and classification of Android malware. According to experimental findings, the hybrid approach improved the overall performance as compared with static and dynamic analysis separately. Table 1 summarizes existing optimization techniques used for the detection of Android malware.

Table 1. Summary of existing optimization techniques for Android malware detection

Author	Type of Analysis	Feature Selection Optimization Technique	Dataset	Results (Accuracy)	Limitation
N.B.S. et al. (2024) [13]	Static	Swarm Intelligence	Self-created (43,876)	98.92%	Optimization applied only to static API features; lacks evaluation on hybrid or dynamic data
Varma et al. (2021) [16]	Static	BA	CICAndMal2019 (4,115)	95.92%	Feature selection limited to static features; no comparison with other metaheuristics.
Mahesh & Hemalatha (2022) [18]	Static	Adaptive Red Fox Optimization	Self-created (34,000)	97.29%	Static image-based features only; computational overhead not analyzed
Lingru Cai et al. (2021) [19]	Static	Feature weighting	Drebin (129,013) AMD (24,553)	96.20%	Uses deterministic weighting rather than metaheuristic optimization; no hybrid analysis
Sulaiman et al. (2018) [20]	Static	Whale Optimization Algorithm (WOA)	Self-created (1,500)	98.21%	Evaluation limited to static permissions; absence of dynamic or hybrid features
Alzubi et al. (2022) [21]	Static	Harris Hawks Optimization (HHO)	CICAndMal2017 (10,854)	93.10%	Optimization confined to static features; no multi-optimizer comparison
Smmarwar et al. (2022) [22]	Hybrid	Optimized Ensemble	CICAndMal2017 (5,491)	96.95%	Employs a single optimization method without systematic comparison
Taha & Barukab (2022) [23]	Static	GA	Drebin (15,036)	94.15%	GA applied only to static features; dynamic behavior not considered
Sihag et al. (2021) [25]	Dynamic	-	Self-created (13,533)	98.84%	Dynamic analysis is computationally expensive.
Ullah et al. (2022) [29]	Hybrid	Attention-based learning	CIC-Inve-sAndMal2019 (5,000)	99.27%	High computational cost
Taher et al. (2023) [30]	Hybrid	Fuzzy-metaheuristics optimization	CICAndMal2017 (5,494) Drebin (5,560)	98.10%	Model complexity

The comparison performed in the summary Table 1 indicates that most of the existing Android malware detection studies are based on a single optimization or feature selection technique, applied to static, dynamic, or hybrid features. Furthermore, these studies have combined optimization techniques with either ML or DL models in isolation. In this paper, we explore multiple metaheuristic optimization algorithms, i.e., PSO, GA, and BA, within the same

experimental framework so that a proper investigation of their relative effectiveness can be conducted. Further, the optimized hybrid features are used to build DL and ML models so that the effects of detection performance can be better gauged.

3. Methodology Used

This section describes the methodology for detecting malware on Android devices. It has three stages, i.e., collection of data, preparation of data, and Android malware detection. In the data acquisition phase, malicious and benign Android apps are gathered from different sources, like virusshare [31], apkpure [32], and apk mirror [33].

In the data preparation phase, the MD5 hash algorithm is utilized for eliminating duplicate samples. Avira Antivirus [34] is used to check the maliciousness of Android apps. These apps are then analysed statically and dynamically to extract their features. Redundant and irrelevant features are removed using the feature selection method. In this study, optimization algorithms (PSO, GA, and BA) are used for selecting the best features.

Afterwards, five ML and three DL algorithms are trained using the original set of features and features selected using optimization algorithms to detect the Android malware. The findings are compared using the 5-fold cross-validation method. The suggested approach is summarized in Figure 1. The following is a detailed explanation of each stage:

3.1. Data Acquisition

A total of 4,349 Android apps (malicious and benign) were acquired in late 2020s from various sources, including virusshare, apkpure, and apk mirror. Malicious applications were obtained by registering on the virusshare website, whereas benign apps were gathered from platforms such as apkpure and apk mirror. The Android application package (.apk) file type is used by all of these apps.

3.2. Data Preparation

In the data preparation phase, duplicate samples are removed using the MD5 hash algorithm. 3,755 Android apps are left after removing the duplicates. The Avira Antivirus (AV) tool is used to scan Android apps obtained from the previous stage for labelling. After labelling, 1,900 benign and 1,855 malicious apps are left. The other steps used while preparing the data are as follows:

3.2.1. Feature Extraction

To extract various features from Android apps, both static and dynamic malware analysis methods are used. In static malware analysis, firstly, all the files are collected in a zip format, and then to view those files, we need to unzip them. Baksmali Disassembler [35], String [36], and AXMLPrinter2 [37] are used to extract the four major types of static features, i.e., intents, API calls, command strings, and permissions. Figure 2 depicts the feature extraction process of static features. In dynamic malware analysis, all the Android apps are made to be executed in the sandbox environment created using CuckooDroid [38]. CuckooDroid handles the Android emulator and generates reports after the analysis.

A Linux virtual machine running the Android emulator serves as the guest computer, which isolates and runs Android malware samples. The emulator runs the apps, collects data, and sends it back to CuckooDroid. Each sample is analyzed for up to 190 seconds. Once the analysis is complete, the results are compiled in JSON format. Figure 3 shows the feature extraction process of dynamic features.

Existing literature and the official Android site indicate that dynamic features in malicious apps commonly fall into four categories: cryptographic operations, dynamic permissions, system calls, and information leaks. The characteristics extracted during static and dynamic malware analysis are described in Tables 2 and 3. Among all the benign and malicious apps, 359 static and 322 dynamic features are retrieved. The created dataset is posted on Kaggle for public access (Link: https://www.kaggle.com/datasets/kshmtachauhan/android-malware-dataset).

3.2.2. Feature Selection

Feature selection is a way of reducing the number of input variables. It helps in choosing the most appropriate features because choosing these features can give a good result and boost the classification model accuracy.

To select the best features, three optimization algorithms, i.e., PSO, GA, and BA, are used. Feature optimization is important because it reduces computation time, controls the overfitting problem, speeds up training, and improves data visualization.

The different optimization algorithms employed in this work are discussed as follows:

- PSO: It is a population-based random generation optimization technique that mimics a fish's foraging behaviour, a school of delusional students, or a swarm of birds. It uses a population of particle swarms (swarm size of 20) close to the best field of search space to locate the overall optimal solution [39]. Interia weight is dynamically adjusted from 0.5 to 0.9 and parameter (C1=1.5 and C2=2.0) for 100 iterations. Generations are restarted by searching for optima after a population is started at random. PSO lacks evolution operators like mutation and crossover. Potential solutions, or particles, follow the particles that are currently optimal as they move around the problem space. In the problem space, each particle has a record of its position, which is determined by the best solution it has found so far.

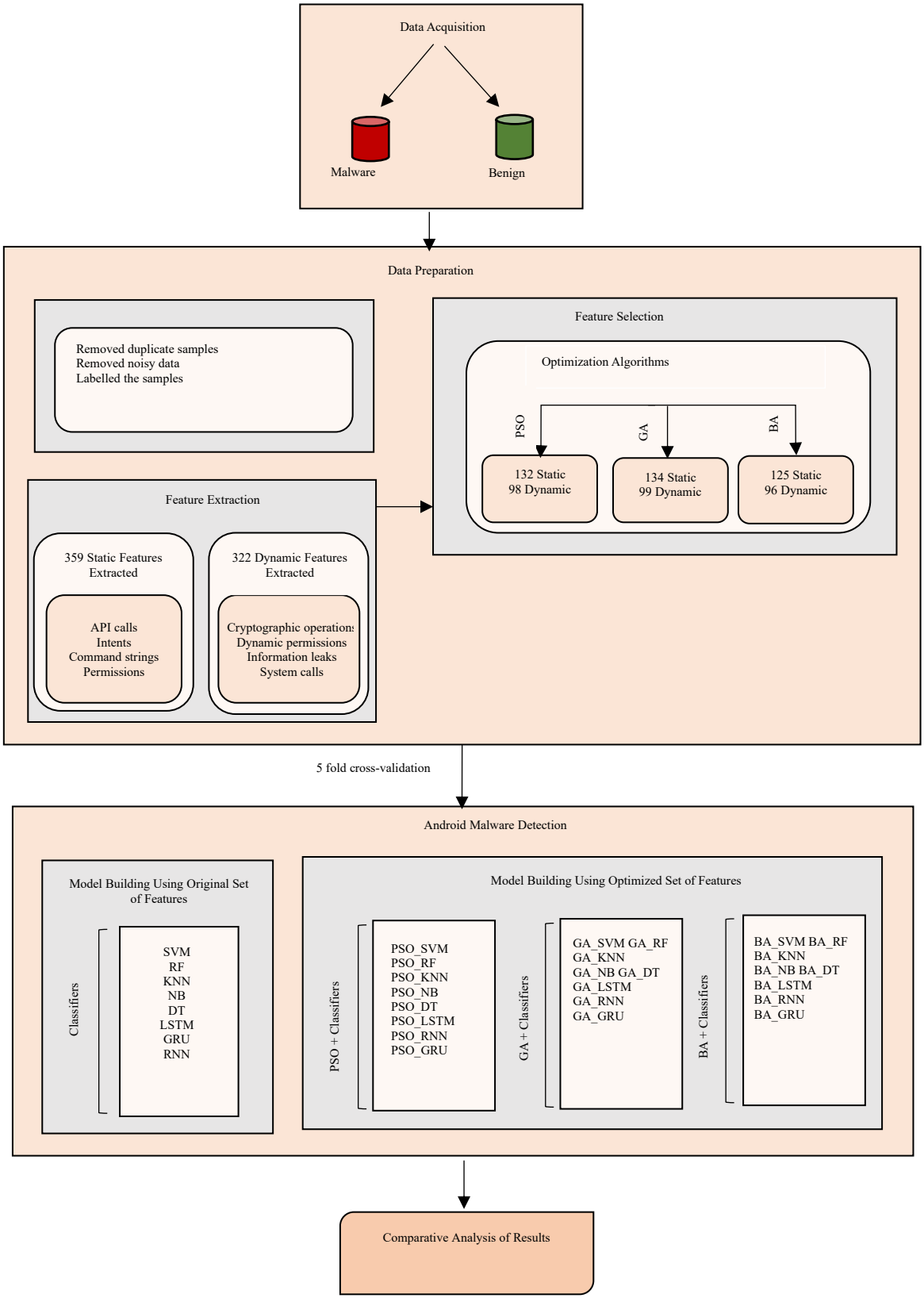

Fig. 1 Workflow of proposed work

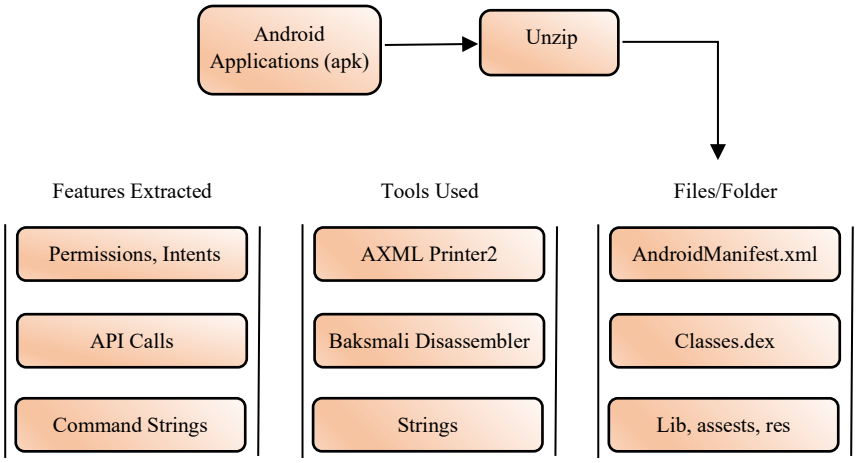

Fig. 2 Static feature extraction procedure

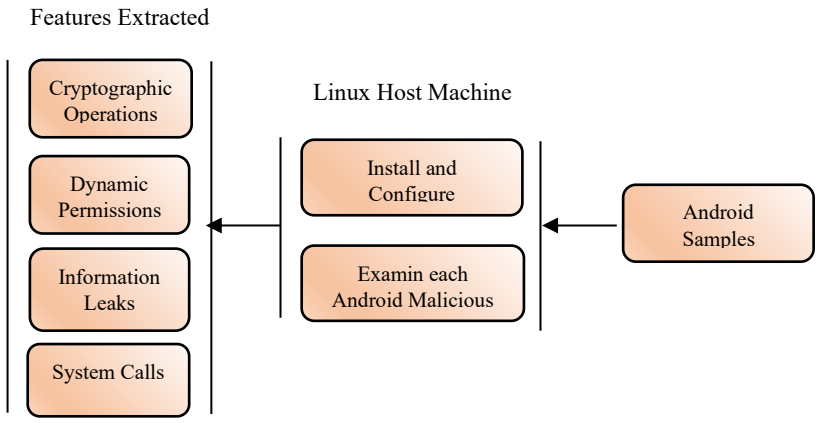

Fig. 3 Dynamic feature extraction procedure

Table 2. Description of features extracted using static malware analysis

Tool used	Features name	Description	No. of features	Example
AXML Printer2	API Calls	API stands for application programme interface. API is used to interact with devices and obtain all information from the server.	52	MessengerService, IRemoteService, onBind, Runtime. load, transact, bindService
AXML Printer2	Permissions	Information about the user is protected through permission.	277	BLUETOOTH_SHARE, FM_RADIO, NOTIFICATIONS, REBOOT, RECOVERY, INTERNET
Baksmali Disassembler	Intents	These are used to perform some activity on the screen that corresponds to the operation the user is requesting.	24	RUN, BATTERY_LOW, CALL_BUTTON, SENDTO, SCREEN_ON, BATTERY_OKAY
String	Command Strings	It examines the command string found in the lib, res, and assets folders.	6	chmod, chown, mount, remount, /system/app, /system/bin

Table 3. Description of features extracted using dynamic malware analysis

Tool used	Features name	Description	No. of features	Example
Cuckoo Droid	Cryptographic Operations	This feature includes encryption, decryption, and key generation, as well as various cryptographic algorithms used to secure data.	78	Encryption_AES, decryption_AES, keyalgo_AES
Cuckoo Droid	Dynamic Permissions	Dynamic permissions are used in the runtime environment and are an important factor in app behaviour analysis.	71	AUDIO_FILE_ACCESS, WRITE_AUDIO_DATA, RECORD_AUDIO, RECEIVE_SMS, WRITE_CONTACT_DATA, SET_TIME
Cuckoo Droid	Information Leaks	Sending confidential data to an unauthorised user causes data leaks.	123	IMEI_SMS, PHONE_NUMBER_File, IMSI_File, IMSI_Network
Cuckoo Droid	System Calls	System calls, which are used to obtain services from the kernel, are always given a higher priority than other operations.	50	SENDTO, CLOSE, CONNECT, RENAME, ACCESS, WRITE

This is called the Pbest value. Additionally, the best value achieved by any of its neighboring particles is known as the Lbest value. Gbest is the best value that is obtained when all neighbours of the particle are also members of the particle's topology.

Every time, the speeds towards the Lbest and Pbest positions change. Acceleration is weighted by a random factor, and independent numbers are generated towards the Pbest and Lbest locations [40]. Steps of the algorithm is defined below.

Algorithm: Particle Swarm Optimization

Input: K (minimum number of iterations), M (population size), M_V (number of variables), L_B (lower bound), C1, C2 (Cognitive factors), U_B (upper bound), itr (iteration)
Output: Gbest (global best), Ftn (fitness function)
Initialize the population
for k = 1 to M, do //*Randomly initialize the velocity and swarm position*//
Y(k,l)=round(L_B(l)+rand()*(U_B(l) - L_B(l)));
end for
for itr = 1 to K, do
for each kϵ[M,1], do
for each lϵ[M_V,1], do
$V_{k,l}^{m+1} = xV_{k,l}^{m} + C_1\ rand[0,1]X\left(Pbest_{k.l}^{m} - Y_{k,l}^{m}\right) + C_2\ rand\ [0,1]\ X\ (Gbest - Y_{k.l}^{m})$ //* Updating swarm velocity *//
$Y_{k,l}^{m+1} = V_{k,l}^{m+1} + Y_{k,l}^{m}$ //* Updating swarm position*//
Modify the $Y_{k,l}^{m+1}$between L_B and U_B
end for
Update Pbest by:
if ftn $(Y_{k,l}^{m+1}) < ftn\left(Pbest_{k,l}^{m}\right)then,$ //* Evaluate the swarm fitness *//
$Pbest_{k,l}^{m} = Y_{k,l}^{m+1}$
end if
if ftn$\left(Pbest_{k,l}^{m}\right) < ftn\ (Gbest)$ then //* Update Gbest *//
Gbest = $Pbest_{k,l}^{m}$
end if
end for
Check the stopping criteria. If itr <K, go to step 6, Otherwise stop
end for

- GA: GA is called a modifying heuristic search algorithm. It is based on genetic principles and natural selection. This method is utilised to get a superior optimisation and search space solution. It is a bio-activated operator similar to mutation, selection, and crossover [41]. The population size used in this algorithm is 50 with 0.8 crossover and 0.01 mutation rate. Steps of the algorithm is given below.

Algorithm: Genetic Algorithm

Input: n (size of population), fi (fitness), Z (Fraction of iteration), l (mutation rate)
Output: X_{bs} (Global best solution)

```
Initialize the population
X_k (k=1,2,….n)                                   //* Set initial population of n chromosomes *//
U=0                                               //* Set the iteration counter *//
Calculate fi for each U∈ X_k
do
    Choose (1- Z) × n members of X_k and insert into X_{k+1}   //* Selection *//
    Choose Z × n members of X_k        //* Apply crossover and insert offspring into X_{k+1} *//
    Choose l × n members of X_{k+1}    //* Apply mutation to the offspring given the probability of mutation *//
    Calculate the fi for each U ∈ X_k. //* New population should replace the existing one *//
    U=U+1                              //* Increment current iteration *//
end do
while the condition not met
return the global best solution X_bs
```

- BA: BA is a frequently used heuristic optimization approach inspired by the echolocation behaviour of bats. It works on the behaviour of bats and how they sense objects and their distance. The population size here is 25 bats, with 0.5 loudness, and 0-2 Hz frequency rate for 100 iterations. Every iteration of this algorithm involves a group of bats trying to discover the optimal answer [16, 42]. Steps of this algorithm are given below.

Algorithm: Bat Algorithm

Input: Xi (bat position), Vi (velocity), fi (frequency), ei (pulse emission rate), li (loudness), n (Max number of iteration), f(fitness function)
Output: Gbest (global best)

```
Initialization of Xi, Vi, fi, ei, and li.        //* Set initial values *//
while (t<n)                                      //* Number of iteration *//
Update the Xi, Vi, and Fi .                      //* Generate new solution *//
    if (rand > ei) then,
        Xnew,                                    //* randomly generate a local solution and select the best one *//
    end if
    fnew                                         //* Generate the new solution *//
    if (rand < li and fnew < f(Xi)) then
        Update fnew
        Increase ei and decrease li
    end if
    find the best Gbest (Xi)                     // * Sort the bats based on fitness function *//
end while
```

PSO, GA, and BA are the three optimization algorithms used to select the optimized set of features. The number of static features selected using PSO, GA, and BA are 132, 134, and 125, respectively. The number of dynamic features selected using PSO, GA, and BA are 98, 99, and 96, respectively. The Table 4 shows the description of the dataset.

3.3. Android Malware Detection

For the detection of Android malware, five ML algorithms are used: Decision Tree (DT), k-Nearest Neighbour (KNN), Random Forest (RF), Support Vector Machine (SVM), and Naive Bayes (NB), as well as three DL algorithms: Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), and Recurrent Neural Network (RNN). The 5-fold cross-validation approach is utilized here, dividing the entire dataset into five equal parts. The data set is separated into five portions, four for training and one for testing the model. This section describes the five ML and three DL algorithms that are built using the original and optimized set of features.

Table 4. Description of the dataset

Benign applications	Malicious applications	No. of static features extracted	No. of dynamic features extracted	No. of static features selected			No. of dynamic features selected		
				PSO	GA	BA	PSO	GA	BA
1900	1855	359	322	132	134	125	98	99	96

3.3.1. Model Building using an Original and Optimized Set of Features

This work utilizes the following ML and DL algorithms:

- SVM: SVM is the most popular and ML algorithm. A hyperplane is employed in SVM to split the data points into two areas. It determines the largest possible distance between all dimensions. [43]. It creates an N-dimensional space if the data is not linearly separable.
- RF: RF is a popular ensemble learning technique that works on a decision tree, and it uses the average of all forecasts to determine the outcome rather than relying on just one tree decision. This method is popular because it trains on a random basis using each piece of data separately [44].
- KNN**:** It is the most basic algorithm of ML that works on the similarity of the features, in which when new data points enter, they are assigned based on their proximity to the points in the training set [45].
- DT: DT is a supervised learning method used in regression and classification models. It displays the prediction result as a tree-like structure. The decision tree displays the outcome in an easy-to-understand visual form [46].
- NB: NB is among the most straightforward and effective classification algorithms; it builds the model rapidly and yields the finest prediction results. The algorithm produces a result based on the probability prediction of an object [47].
- LSTM: LSTM is an improved version of RNN that can remember long-term patterns. To keep, remove, and update information, it uses three gates, i.e., input, output, and forget gates. This helps to overcome the vanishing-gradient problem and retain information for an extended period of time [48].
- RNN: RNN is a neural network designed to analyze sequential data, including time series, text, or behavioral logs. RNNs, in contrast to conventional neural networks, contain a feedback loop that enables information to flow from one stage to the next, providing them with a "memory" of earlier inputs [49].
- GRU: GRU is a powerful recurrent neural network that can handle sequential input effectively. The reset gate, the update gate, and the current memory content define the three basic components of the GRU unit. These gates enable the GRU to capture long-term dependencies in sequences by selectively updating and exploiting information from previous time steps [50].

Three models are build using an original and optimized set of features selected using PSO, GA, and BA. The models built using the optimized set of features obtained through PSO are named as PSO_SVM, PSO_RF, PSO_KNN, PSO_DT, PSO_NB, PSO_GRU, PSO_LSTM, and PSO_RNN. The models built using the optimized set of features obtained through GA are named as GA_SVM, GA_RF, GA_KNN, GA_DT, GA_NB, GA_GRU, GA_LSTM, and GA_RNN. Similarly, the models built using an optimized set of features through BA are BA_SVM, BA_RF, BA_KNN, BA_DT, BA_NB, BA_GRU, BA_LSTM, and BA_RNN.

3.4. Evaluation Parameters Used

The productivity of diverse models employed for malware detection is assessed using many measures, including False Positive Rate (FPR), precision, F-measure, True Positive Rate (TPR), and accuracy. These metrics are delineated utilizing the False Positive (FP), True Positive (TP), False Negative (FN), and True Negative (TN) components of the confusion matrix. In this context, TP represents the instances where the model correctly identifies positive cases, FP indicates the instances where the model erroneously predicts positive cases, TN refers to the instances where the model accurately identifies negative cases, and FN denotes the instances in which the model accidentally classifies negative cases as positive.

- FPR: It is the ratio of negative cases that are misclassified as positive cases in the data. The computation is performed in accordance with Equation (1).

$$FPR = \frac{FP}{TN+FP} \quad (1)$$

- TPR: It is also called recall or sensitivity. The ratio of positive instances that are precisely identified. The computation is performed in accordance with Equation (2).

$$TPR = \frac{TP}{TP+FN} \quad (2)$$

- Precision: It is the ratio of the number of true positive cases anticipated to the total number of cases. The computation is performed in accordance with Equation (3).

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

- F-measure: It is evaluated using both recall and precision. The computation is performed in accordance with Equation (4).

$$\text{F} - \text{measure} = 2\ \frac{Recall \times Precision}{Recall + Precision} \quad (4)$$

The best score for F-measure is 1.

- Accuracy: It is the ratio of accurate predictions to the total number of instances. The computation is performed in accordance with Equation (5).

$$Accuracy\ (\%) = \frac{TP+TN}{TP+TN+FP+FN} * 100 \quad (5)$$

4. Experimental Results

This section elaborates the experimental findings achieved using five ML and three DL algorithms optimized using PSO, GA, and BA on the basis of static, dynamic, and hybrid features.

The classifier's performance is measured by running the experiments on Python 3.9 on AMD Ryzen 5, 64-bit and 8 GB of RAM. The algorithms are executed using the sklearn [51] package in a Python script. Various evaluation metrics, such as accuracy, TPR, FPR, precision, and F-measure, are used in experiments using the 5-fold cross-validation approach.

Table 5 describes the classification results of different ML and three DL algorithms. It displays that GRU provides the best and NB provides the worst classification results in all static, dynamic, and hybrid results. Comparing all the classifiers' outputs in terms of accuracy and precision, GRU gives 98.63% and 0.987, respectively, in the case of the hybrid approach.

Table 5. Classification results of ML and DL algorithms using original set of static, dynamic, and hybrid features

Approach	Classifiers	FPR	TPR	Precision	F-measure	Accuracy (%)
Static	SVM	0.055	0.940	0.950	0.950	95.00
	RF	0.036	0.967	0.967	0.967	96.71
	KNN	0.049	0.957	0.957	0.956	95.64
	DT	0.053	0.957	0.958	0.957	95.71
	NB	0.128	0.861	0.863	0.861	85.42
	RNN	0.033	0.969	0.97	0.969	96.89
	LSTM	0.028	0.976	0.977	0.976	97.62
	GRU	0.031	0.972	0.973	0.972	97.20
Dynamic	SVM	0.034	0.965	0.965	0.965	96.53
	RF	0.029	0.971	0.971	0.971	97.10
	KNN	0.039	0.960	0.960	0.960	96.04
	DT	0.046	0.955	0.956	0.955	95.58
	NB	0.058	0.940	0.941	0.940	94.08
	RNN	0.028	0.973	0.973	0.973	97.22
	LSTM	0.027	0.975	0.975	0.975	97.36
	GRU	0.021	0.983	0.984	0.983	98.34
Hybrid	SVM	0.021	0.980	0.982	0.980	98.10
	RF	0.017	0.984	0.984	0.984	98.42
	KNN	0.019	0.982	0.983	0.982	98.26
	DT	0.032	0.971	0.972	0.971	97.17
	NB	0.053	0.951	0.952	0.951	95.18
	RNN	0.016	0.985	0.985	0.985	98.49
	LSTM	0.015	0.986	0.986	0.986	98.57
	GRU	0.014	0.987	0.987	0.987	98.63

Figure 4 shows the comparison of these five ML and three DL algorithms for static, dynamic, and hybrid features on the basis of accuracy.

It shows that all the classifiers perform better in case of a hybrid approach (when static and dynamic are combined). The minimum accuracy of 85.42% is obtained by NB in the case of static approach.

Table 6 describes the classification using optimized static features. PSO, GA, and BA optimization techniques are used to select the best features and used with SVM, RF, KNN, DT, NB, GRU, LSTM, and RNN to enhance the results of all the classifiers.

It is found that BA_GRU achieves the highest accuracy of 98.65%, followed by BA_LSTM and PSO_GRU, which give 98.29% and 97.98% accuracy, respectively. The precision, TPR, and F-measure obtained by BA_GRU are 0.988, 0.987, and 0.987, respectively.

Figure 5 shows the comparison of ML and DL classifiers (trained using the original set of static features) and the classifiers trained using optimized static features, i.e., the

features selected using optimization algorithms. These classifiers are named as Particle Swarm Optimization_classifiers (PSO_C), Genetic Algorithm_classifier (GA_C), and Bat Algorithm_classifiers (BA_C). It is evident from the figure that BA_C provides the best results as compared to GA_C and PSO_C.

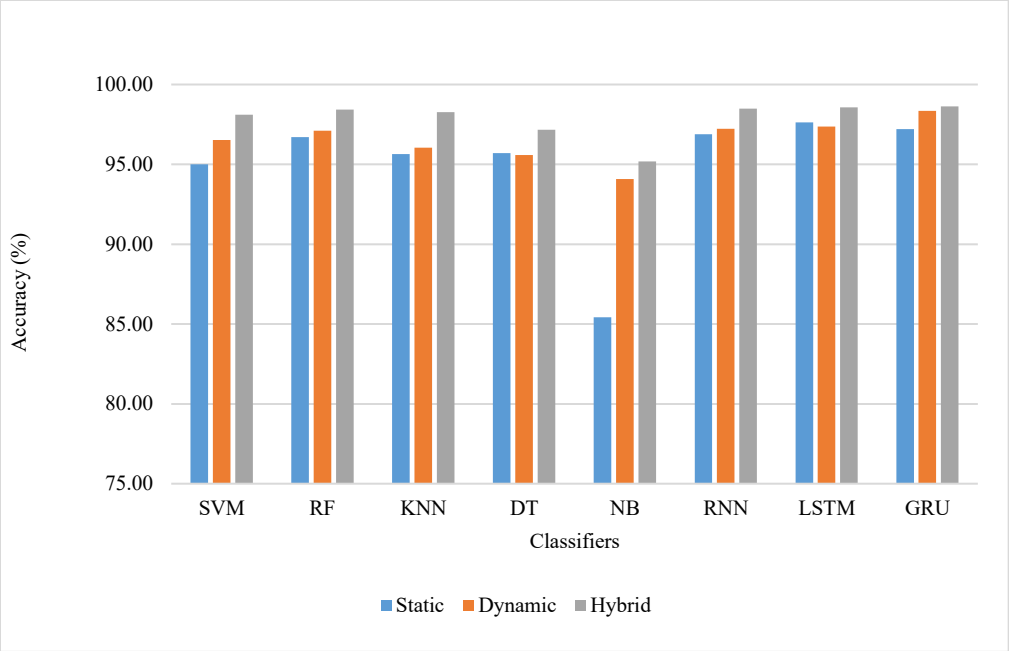

Fig. 4 Comparison of ML and DL classifiers using original set of static, dynamic, and hybrid features

Table 6. Classification results of ML and DL algorithms using an optimized set of static features

Classifiers	FPR	TPR	Precision	F-measure	Accuracy (%)
PSO_SVM	0.049	0.956	0.956	0.955	95.54
GA_SVM	0.056	0.953	0.953	0.953	95.31
BA_SVM	0.047	0.959	0.959	0.959	95.99
PSO_RF	0.035	0.968	0.969	0.968	96.83
GA_RF	0.036	0.967	0.968	0.967	96.74
BA_RF	0.030	0.972	0.973	0.972	97.22
PSO_KNN	0.051	0.959	0.959	0.959	95.91
GA_KNN	0.052	0.958	0.958	0.958	95.80
BA_KNN	0.048	0.959	0.96	0.959	95.98
PSO_DT	0.044	0.960	0.960	0.960	96.02
GA_DT	0.047	0.959	0.959	0.959	95.91
BA_DT	0.043	0.961	0.962	0.961	96.10
PSO_NB	0.123	0.876	0.878	0.876	87.67
GA_NB	0.125	0.872	0.873	0.871	87.25
BA_NB	0.100	0.881	0.882	0.881	88.13
PSO_GRU	0.026	0.98	0.981	0.98	97.98
GA_GRU	0.027	0.976	0.977	0.976	97.58
BA_GRU	0.02	0.987	0.988	0.987	**98.65**
PSO_LSTM	0.029	0.973	0.974	0.973	97.32
GA_LSTM	0.034	0.968	0.969	0.968	96.83
BA_LSTM	0.022	0.983	0.984	0.983	98.29
PSO_RNN	0.033	0.969	0.97	0.969	96.94
GA_RNN	0.038	0.961	0.963	0.962	96.20
BA_RNN	0.028	0.975	0.976	0.975	97.55

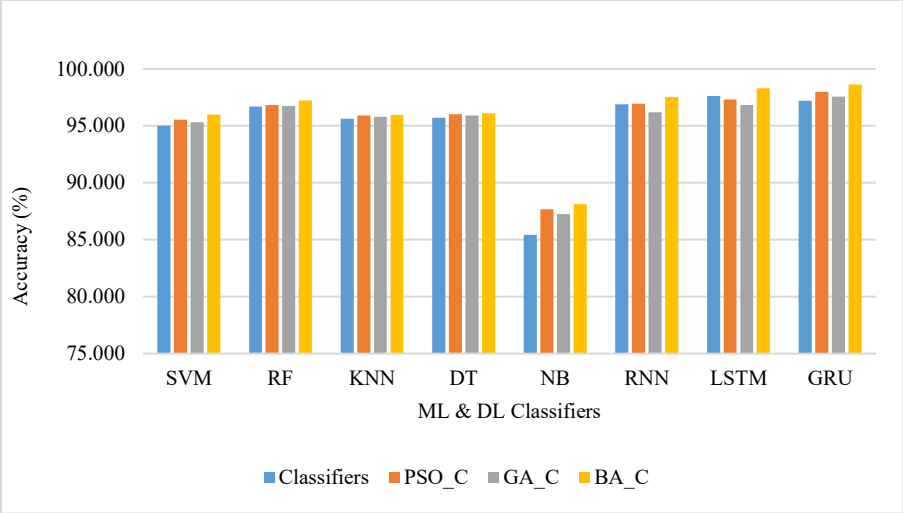

Fig. 5 Comparison of ML and DL algorithms using original set of static features and features selected using optimization algorithms

Table 7 shows the classification results of ML and DL algorithms using an optimized set of dynamic features. These features are then utilized to construct the model with ML and DL classifiers. During the evaluation, the FPR, TPR, accuracy, F-measure, and precision, and are used.

Among all the classifiers, GRU performs better than other classifiers. The accuracy obtained by BA_GRU is 99.12%, followed by PSO_GRU and BA_LSTM with 98.83% and 98.82%, respectively.

Table 7. Classification results of ML and DL algorithms using an optimized set of dynamic features.

Classifiers	FPR	TPR	Precision	F-measure	Accuracy (%)
PSO_SVM	0.030	0.969	0.969	0.969	96.92
GA_SVM	0.032	0.967	0.968	0.967	96.74
BA_SVM	0.028	0.971	0.972	0.97	97.13
PSO_RF	0.025	0.976	0.977	0.976	97.68
GA_RF	0.026	0.973	0.973	0.973	97.34
BA_RF	0.024	0.978	0.978	0.978	97.89
PSO_KNN	0.036	0.965	0.966	0.965	96.57
GA_KNN	0.038	0.961	0.964	0.961	96.28
BA_KNN	0.034	0.969	0.969	0.969	96.94
PSO_DT	0.042	0.960	0.961	0.960	96.12
GA_DT	0.045	0.958	0.958	0.958	95.81
BA_DT	0.04	0.963	0.963	0.962	96.30
PSO_NB	0.056	0.946	0.946	0.945	94.58
GA_NB	0.058	0.941	0.942	0.941	94.12
BA_NB	0.055	0.953	0.953	0.951	95.24
PSO_GRU	0.017	0.989	0.99	0.989	98.83
GA_GRU	0.026	0.976	0.976	0.976	97.48
BA_GRU	0.015	0.992	0.993	0.992	**99.12**
PSO_LSTM	0.020	0.985	0.986	0.985	98.52
GA_LSTM	0.024	0.979	0.98	0.979	97.98
BA_LSTM	0.018	0.988	0.989	0.988	98.82
PSO_RNN	0.023	0.981	0.982	0.981	98.05
GA_RNN	0.027	0.974	0.975	0.974	97.56
BA_RNN	0.020	0.984	0.985	0.984	98.41

Figure 6 shows the comparison of ML and DL classifiers (trained using the original set of dynamic features) and classifiers trained using optimized dynamic features, i.e., the features selected using optimization algorithms. It shows that BA_C provides the best results as compared to GA_C and PSO_C. BA_GRU outperforms the other classifiers.

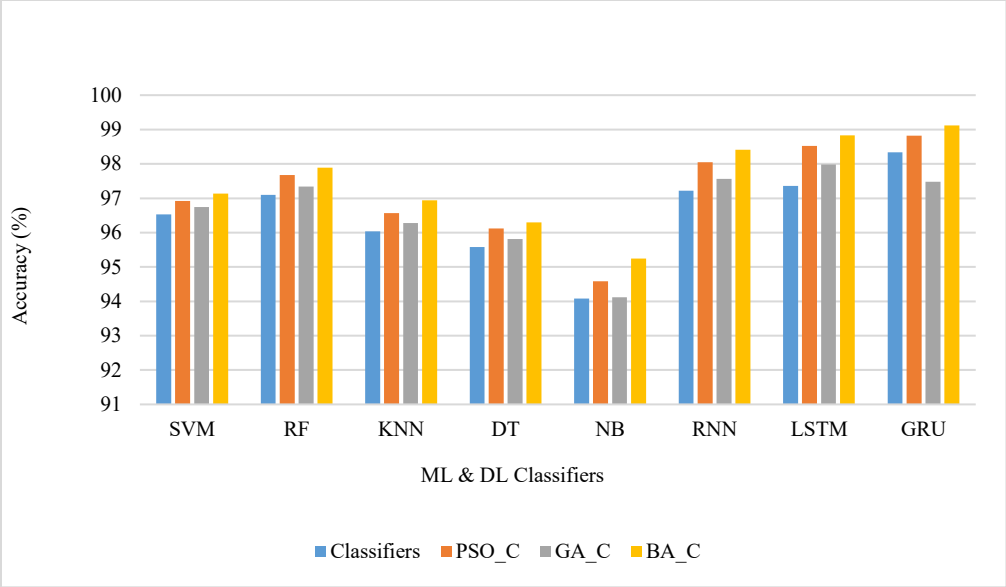

Fig. 6 Comparison of ML and DL algorithms using original set of dynamic features and features selected using the optimization algorithms

Table 8 shows the experimental results using an optimized set of hybrid features, in which BA_GRU achieves 99.45% accuracy, which is superior to static and dynamic features. The TPR, F-measure, and precision obtained by BA_GRU are 0.996, 0.996, and 0.996, respectively.

Table 8. Classification results of ML and DL algorithms using optimized set of hybrid features

Classifiers	FPR	TPR	Precision	F-measure	Accuracy (%)
PSO_SVM	0.017	0.984	0.984	0.984	98.45
GA_SVM	0.018	0.983	0.984	0.983	98.36
BA_SVM	0.014	0.988	0.989	0.988	98.89
PSO_RF	0.015	0.987	0.988	0.987	98.73
GA_RF	0.016	0.985	0.985	0.985	98.55
BA_RF	0.014	0.989	0.989	0.989	98.98
PSO_KNN	0.017	0.985	0.986	0.985	98.50
GA_KNN	0.019	0.983	0.983	0.983	98.31
BA_KNN	0.017	0.987	0.987	0.987	98.70
PSO_DT	0.028	0.976	0.976	0.976	97.65
GA_DT	0.032	0.973	0.973	0.973	97.32
BA_DT	0.027	0.978	0.978	0.978	97.83
PSO_NB	0.050	0.957	0.958	0.957	95.72
GA_NB	0.053	0.952	0.952	0.952	95.26
BA_NB	0.050	0.96	0.961	0.960	96.06
PSO_GRU	0.010	0.994	0.994	0.994	99.34
GA_GRU	0.013	0.991	0.991	0.991	99.02
BA_GRU	0.009	0.996	0.996	0.996	**99.45**
PSO_LSTM	0.011	0.992	0.992	0.992	99.21
GA_LSTM	0.014	0.989	0.989	0.989	98.93
BA_LSTM	0.01	0.994	0.994	0.994	99.32
PSO_RNN	0.013	0.99	0.99	0.99	99.05
GA_RNN	0.015	0.988	0.987	0.987	98.82
BA_RNN	0.012	0.992	0.992	0.992	99.18

Figure 7 shows the comparison of ML and DL classifiers (trained using the original set of hybrid features) and the classifiers trained using optimized hybrid features, i.e., the features selected using optimization algorithms. It shows that BA_C provides the best results as compared to GA_C and PSO_C.

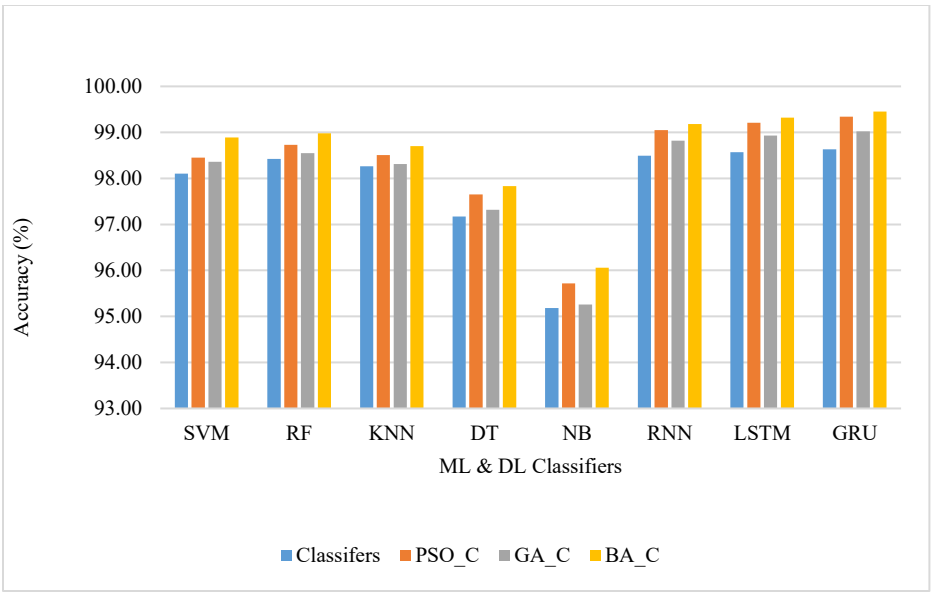

Fig. 7 Comparison of ML and DL algorithms using the original set of hybrid features and features selected using the optimization algorithms

Figure 8 shows the comparison of GRU models built using the optimized set of features obtained through PSO, GA, and BA optimization techniques for static, dynamic, and hybrid features.

It is clear from the figure that the hybrid approach performs best in all the optimization techniques. Further, it is found that BA optimization techniques give the best results for the hybrid approach.

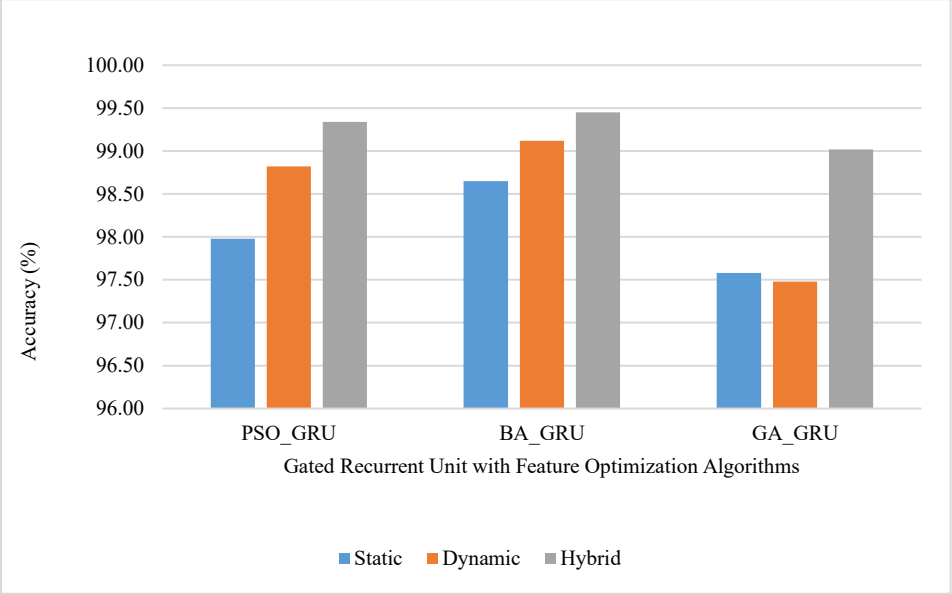

Fig. 8 Comparison of PSO, GA, and BA optimization techniques with GRU using static, dynamic, and hybrid features

5. Discussion

This section discusses the results obtained. Overall experimental findings indicate that hybrid features consistently outperform static and dynamic features across all ML and DL classifiers. Three metaheuristic optimization techniques, PSO, GA, and BA, are used for their ability to efficiently explore complicated and large datasets, making them ideal for high-dimensional feature selection problems. All three improve performance, with the BA achieving the most notable gains. This study suggests that selecting a compact and informative subset of features can significantly boost classifier efficiency while reducing unnecessary overhead. Five ML and three DL classifiers are used in conjunction with the optimization algorithms, with GRU using BA achieving the maximum accuracy of 99.45%. Since the dataset used in this study is small, the improvement in accuracy between the original and reduced feature sets is not

very good. However, it highlights the utility of the proposed approach for larger datasets, where it can dramatically cut the computational cost of model training, improve scalability, and help the classifier focus on the most discriminative features. Overall, the study indicates that hybrid features with metaheuristic optimization methods yield a reliable and efficient solution suitable for real-world Android malware detection.

The drawback of the proposed work is that integrating metaheuristic optimization approaches with DL techniques requires a lot of computing power during the feature selection process. Scaling the approach for real-time detection and deployment on resource-constrained Android devices might be challenging, particularly when dealing with complicated, varied, and constantly changing malware samples.

5.1. Comparison with Prior Work

The comparison of the proposed method with existing methods for the detection of Android malware is presented in Table 9. It is important to note that a direct quantitative comparison is not entirely possible, as the cited studies utilize different datasets and experimental conditions. Nevertheless, this comparison is included to provide a general perspective on how the proposed approach performs relative to prior work. The findings reveal that the proposed technique performs competitively in this situation.

To evaluate the efficacy of the suggested methodology, tests are conducted on two benchmark datasets developed and employed by Dhalaria et al. [6] and Daniel Arp et al. [52]. Table 10 shows a summary of the results of the comparison.

The optimized model that uses hybrid malware features, where a GRU is combined with the BA (BA_GRU) for feature optimization, gets 97.72% accuracy on Dhalaria's dataset and 99.21% accuracy on Drebin's dataset.

These findings indicate that the proposed approach effectively reduces features and enhances efficiency, making it well-suited for larger datasets, where optimization has a greater impact on accuracy and computational performance.

Table 9. Performance comparison with existing Android malware detection approaches

Author	Dataset Used	Type	Technique	Accuracy(%)
Dhalaria & Gandotra (2020) [6]	AndroMD	Hybrid	IG and ML algorithms	98.53%
Taher et al. (2023) [30]	CICAndMal2017 Drebin	Hybrid	Fuzzy-metaheuristics optimization and DL	98.10%
Proposed	Self-created	Hybrid	Metaheuristics optimization and DL	99.45%

Table 10. Comparative study of the proposed technique on the benchmark datasets

Dataset	Feature type	TPR	Precision	Accuracy (%)	F-measure
Self-curated	Hybrid features	0.996	0.996	99.45	0.996
Drebin [52]	Static Features	0.989	0.989	99.21	0.989
AndroMD [6]	Hybrid feature	0.986	0.987	98.71%	0.987

5.2. Discussion with Respect to Research Questions

The results obtained across static, dynamic, and hybrid features with a metaheuristic method used for feature optimization allow a systematic evaluation of the formulated research questions and corresponding hypotheses. The optimized hybrid features always perform better than the original set across classifiers (Table 8), thus supporting H1.

Classifiers with optimized hybrid features yield better accuracy and a higher F1-score over non-optimized features, thus proving H2. Among the three metaheuristic methods used for optimizing the features, BA provides the best performance, thus confirming H3.

Thus, the findings validate the stated hypotheses and emphasize that optimized hybrid features improve classification performance across all classifiers. Among the evaluated optimization techniques, the BA achieves the most effective and stable results.

6. Conclusion and Future Scope

This study provided a comprehensive technique for enhancing Android malware detection by synergistically combining hybrid malware analysis with feature optimization. We created our own collection of Android apps, both malicious and benign, after gathering both static and dynamic data. To assist security researchers in testing the recently created methods, it is made publicly available on Kaggle. By fusing feature optimization with hybrid malware analysis, the suggested approach enhanced Android malware detection. Hybrid malware analysis combines the features acquired from static and dynamic malware analysis.

Three optimization techniques, namely PSO, GA, and BA, were then applied to the acquired features. Original features are used to train five ML algorithms: SVM, RF, KNN, DT, and NB, and three DL algorithms: LSTM, RNN, and GRU. Optimization algorithms for static, dynamic, and hybrid analysis approaches are used to choose the features. Following

a number of experiments, the effectiveness of the suggested strategy was assessed. The findings demonstrate that the gated recurrent unit model with the highest accuracy of 99.45% is constructed using features derived from the bat optimization method for hybrid analysis.

In the future, we intend to enhance Android malware detection using advanced DL techniques. Explainable AI methods will be incorporated to improve model transparency and provide clearer justification for detection decisions.

We will include additional static, dynamic, and network-level features to capture a wider range of malicious behaviours. We also plan to use big data tools to deal with large and more diverse datasets for better scalability.

Data Availability

The dataset used in this research is publicly available on Kaggle. (https://www.kaggle.com/datasets/kshmtachauhan/android-malware-dataset).

Competing Interests

No conflict of interest.

Funding

No funding.

Acknowledgments

Not applicable.

References

[1] Statcounter Global Stats, Mobile Operating System Market Share Worldwide, Statcounter Global Stats, 2026. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Waleed Ali, "Hybrid Intelligent Android Malware Detection using Evolving Support Vector Machine based on Genetic Algorithm and Particle Swarm Optimization," *International Journal of Computer Science and Network Security*, vol. 19, no. 9, pp. 15-28, 2019. [Google Scholar]

[3] Zhongru Ren et al., "End-To-End Malware Detection for Android IoT Devices using Deep Learning," *Ad Hoc Networks*, vol. 101, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Mosaddek Hossain, Suzzana Rafi, and Shohrab Hossain, "An Optimized Decision Tree based Android Malware Detection Approach using Machine Learning," *Proceedings of the 7th International Conference on Networking, Systems and Security*, Association for Computing Machinery, New York, NY, United States, pp. 115-125, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[5] Yanjie Zhao et al., "On the Impact of Sample Duplication in Machine-Learning-based Android Malware Detection," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 3, pp. 1-38, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Meghna Dhalaria, and Ekta Gandotra, "A Hybrid Approach for Android Malware Detection and Family Classification," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, no. 6, pp. 174-188, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Bradley Barth, New Desert Scorpion Spyware Found in Malicious Chat App Aimed at Palestinians, SC Media, 2018. [Online]. Available: https://www.scmagazine.com/news/architecture/new-desert-scorpion-spyware-found-in-malicious-chat-app-aimed-at-palestinians

[8] Deepak Gupta, and Rinkle Rani, "Big Data Framework for Zero-Day Malware Detection," *Cybernetics and Systems*, vol. 49, no. 2, pp. 103-121, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[9] Deepak Gupta, and Rinkle Rani, "Improving Malware Detection using Big Data and Ensemble Learning," *Computers and Electrical Engineering*, vol. 86, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Hussein Al Bazar et al., "A Model for Android Platform Malware Detection Utilizing Multiple Machine Learning Algorithms," *Informatica*, vol. 48, no. 17, pp. 95-108, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] Mohd Faizal Ab Razak et al., "Bio-inspired for Features Optimization and Malware Detection," *Arabian Journal for Science and Engineering*, vol. 43, no. 12, pp. 6963-6979, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[12] Qi Li, and Xiaoyu Li, "Android Malware Detection based on Static Analysis of Characteristic Tree," *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Xi'an, China, pp. 84-91, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[13] B. Suribabu Naick et al., "Malware Detection in Android Mobile Devices by Applying Swarm Intelligence Optimization and Machine Learning for API Calls," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 3s, pp. 67-74, 2022. [Google Scholar] [Publisher Link]

[14] Rajesh Kumar et al., "A Multimodal Malware Detection Technique for Android IoT Devices using Various Features," *IEEE Access*, vol. 7, pp. 64411-64430, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[15] Iman Almomani, Aala Alkhayer, and Walid El-Shafai, "A Crypto-Steganography Approach for Hiding Ransomware within HEVC Streams in Android IoT Devices," *Sensors*, vol. 22, no. 6, pp. 1-20, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[16] P. Ravi Kiran Varma et al., "Bat Optimization Algorithm for Wrapper-based Feature Selection and Performance Improvement of Android Malware Detection," *IET Networks*, vol. 10, no. 3, pp. 131-140, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[17] Eslavath Ravi, and Mummadi Upendra Kumar, "A Comparative Study on Machine Learning and Deep Learning Methods for Malware Detection," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 20, pp. 6117- 6129, 2022. [Google Scholar] [Publisher Link]

[18] P.C. Senthil Mahesh, and S. Hemalatha, "An Efficient Android Malware Detection using Adaptive Red Fox Optimization based CNN," *Wireless Personal Communications*, vol. 126, no. 1, pp. 679-700, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Lingru Cai, Yao Li, and Zhi Xiong, "JOWMDroid: Android Malware Detection based on Feature Weighting with Joint Optimization of Weight-Mapping and Classifier Parameters," *Computers and Security*, vol. 100, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[20] Salamatu Aliyu Sulaiman et al., "Android Malware Classification using Whale Optimization Algorithm," *i-manager's Journal on Mobile Applications and Technologies*, vol. 5, no. 2, pp. 37-45, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[21] Omar A. Alzubi et al., "An Efficient Malware Detection Approach with Feature Weighting based on Harris Hawks Optimization," *Cluster Computing*, vol. 25, no. 4, pp. 2369-2387, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[22] Santosh K. Smmarwar et al., "An Optimized and Efficient Android Malware Detection Framework for Future Sustainable Computing," *Sustainable Energy Technologies and Assessments*, vol. 54, pp. 1-8, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[23] Altyeb Taha, and Omar Barukab, "Android Malware Classification using Optimized Ensemble Learning based on Genetic Algorithms," *Sustainability*, vol. 14, no. 21, pp. 1-11, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[24] Vitor Monte Afonso et al., "Identifying Android Malware using Dynamically Obtained Features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9-17, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[25] Vikas Sihag et al., "De-Lady: Deep Learning based Android Malware Detection using Dynamic Features," *Journal of Internet Services and Information Security*, vol. 11, no. 2, pp. 34-45, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[26] Fei Tong, and Zheng Yan, "A Hybrid Approach of Mobile Malware Detection in Android," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 22-31, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[27] Altyeb Altaher, and Omar Mohammed Barukab, "Intelligent Hybrid Approach for Android Malware Detection based on Permissions and API Calls," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, pp. 60-67, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[28] Abdullah Talha Kabakus, and Ibrahim Alper Dogru, "An In-Depth Analysis of Android Malware using Hybrid Techniques," *Digital Investigation*, vol. 24, pp. 25-33, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[29] Farhan Ullah, Gautam Srivastava, and Shamsher Ullah, "A Malware Detection System using a Hybrid Approach of Multi-Heads Attention-based Control Flow Traces and Image Visualization," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1-21, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[30] Fatma Taher et al., "DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection," *Applied Sciences*, vol. 13, no. 13, pp. 1-23, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[31] VirusShare.com, 2023. [Online]. Available: https://virusshare.com/

[32] APKPure APK for Android Download, 2023. [Online]. Available: https://apkpure.com/apkpure/com.apkpure.aegon

[33] APKMirror - Free APK Downloads - Free and Safe Android APK Downloads, 2023. [Online]. Available: https://www.apkmirror.com/

[34] Download Security Software for Windows, Mac, Android and iOS | Avira Antivirus, 2023. [Online]. Available: https://www.avira.com/?srsltid=AfmBOopZnoAzM0OmbWjT4SyY_Kk8iFWN-2etKe4G3EAPGMTYTtRI8QM6

[35] William Enck et al., "A Study of Android Application Security," *20th USENIX Security Symposium*, vol. 2, no. 2, 2011. [Google Scholar] [Publisher Link]

[36] Ronghua Tian et al., "An Automated Classification System based on the Strings of Trojan and Virus Families," *2009 4th International Conference on Malicious and Unwanted Software*, Montreal, QC, Canada, pp. 23-30, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[37] Nick Sayer, "Google Code Archive - Long-Term Storage for Google Code Project Hosting," *Retrieved from the Internet*, 2014. [Google Scholar]

[38] Cuckoo Droid, Installation-CuckooDroid v1.0 Book, Cuckoo Droid, 2023. [Online]. Available: https://cuckoo-droid.readthedocs.io/en/latest/installation/

[39] Gerhard Venter, and Jaroslaw Sobieszczanski-Sobieski, "Particle Swarm Optimization," *AIAA Journal,*, vol. 41, no. 8, pp. 1583-1589, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[40] G. Venter, AIAA 2002-1235 Particle Swarm Optimization 43rd AIAAIASMEIASCEIAHSIASC Structures, Structural Dynamics, Materials Conference April 22-25, 2002, Denver, Colorado, 2019. [Online]. Available: https://www.cs.odu.edu/~mln/ltrs-pdfs/NASA-aiaa-2002-1235.pdf

[41] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar, "A Review on Genetic Algorithm: Past, Present, and Future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091-8126, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[42] Xin-She Yang, and Amir Hossein Gandomi, "Bat Algorithm: A Novel Approach for Global Engineering Optimization," *Engineering Computations*, vol. 29, no. 5, pp. 464-483, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[43] S.S. Keerthi, and E.G. Gilbert, "Convergence of a Generalized SMO Algorithm for SVM Classifier Design," *Machine Learning*, vol. 46, no. 1, pp. 351-360, 2002. [CrossRef] [Google Scholar] [Publisher Link]

[44] Andy Liaw, and Matthew Wiener, "Classification and Regression by Randomforest," *R News*, vol. 2, no. 3, pp. 18-22, 2002. [Google Scholar]

[45] Laura S. Wood, "Introduction," *Seminars in Oncology Nursing*, vol. 28, no. 3, pp. 141-142, 2012. [CrossRef] [Publisher Link]

[46] J.R. Quinlan, "Learning Decision Tree Classifiers," *ACM Computing Surveys*, vol. 28, no. 1, pp. 71-72, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[47] Pedro Domingos, and Michael Pazzani, "On the Optimality of the Simple Bayesian Classifier Underzero-One Loss," *Machine Learning*, vol. 29, no. 2, pp. 103-130, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[48] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *Journal of Machine Learning Research*, vol. 3, pp. 115-143, 2002. [Google Scholar] [Publisher Link]

[49] Alex Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network," *Physica D: Nonlinear Phenomena*, vol. 404, pp. 1-28, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[50] Farhad Mortezapour Shiri et al., "A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU," *arXiv preprint*, pp. 1-62, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[51] Scikit-Learn, Scikit-Learn: Machine Learning in Python - scikit-learn 1.2.2 Documentation, 2023. [Online]. Available: https://scikit-learn.org/stable/whats_new/v1.2.html

[52] Daniel Arp et al., "Drebin: Effective and Explainable Detection of Android Malware in your Pocket," *Network and Distributed System Security Symposium*, vol. 14, no. 1, pp. 1-15, 2014. [Google Scholar] [Publisher Link]