International Journal of Analysis and Applications

Enhanced Multiclass Android Malware Detection Using a Modified Dwarf Mongoose Algorithm

Rawan D. Alabdallat[1,*], Mosleh M. Abualhaj[1], Ahmad Abu-Shareha[2]

[1]*Department of Networks and Cybersecurity, Al-Ahliyya Amman University, Amman, Jordan*
[2]*Department of Data Science and Artificial Intelligence, Al-Ahliyya Amman University, Amman, Jordan*
**Corresponding author:* alabdallatrawan80@gmail.com

ABSTRACT. The Android operating system has the most market share due to its easy handling and numerous advantages to Android users, which have attracted malicious actors. Android malware detection (AMD) systems based on machine learning (ML) are progressively being developed. However, these systems frequently struggle with high-dimensional datasets, increasing computation time, and lower accuracy. This study proposes a novel method for identifying malware in Android applications that employs a modified Dwarf Mongoose Optimization Algorithm (DMOA) for feature selection. The modified DMOA uses adaptive strategies, including crossover and mutation, to explore the search space more effectively, avoiding local optima and revealing higher-quality feature subsets that increase detection performance. The proposed modified DMOA model is trained and evaluated using the CICAndMal2017 dataset. The results show that it significantly outperforms existing techniques, achieving an accuracy of 100%.

1. Introduction

Malware attacks pose a risk to the security of Android systems for all users and organizations; hence, effective detection methods should be developed [1]. AMD is an essential challenge for smartphone platforms because of the exponential increase in users for Android applications [2]. Recent research has focused on the use of ML methods for application behavior prediction and malware detection [3]. However, ML methods function poorly in malware detection for large-dimensional datasets, which can increase the computation time and lead to imprecise results [4]. Feature selection techniques are used to extract important features and eliminate unwanted data[5]. Nevertheless, these techniques often yield false positive or negative results [6]. Moreover, given the changing nature of malware and the emergence of new types,

Received Aug. 13, 2025

2020 Mathematics Subject Classification. 68T05.

Key words and phrases. android malware detection; dwarf mongoose optimization algorithm; feature selection.

https://doi.org/10.28924/2291-8639-23-2025-248

ISSN: 2291-8639

some features have become irrelevant, necessitating continuous adaptation and the integration of new techniques. Another challenge is that traditional optimization algorithms can be trapped in local optima, especially when dealing with the complex search space typical of malware detection datasets. In such cases, irrelevant feature subsets are selected, which fail to improve detection accuracy [7]. An advanced optimization technique that can help search for the best solution and improve the performance of AMD systems without falling into local optima must be developed. This study deploys a modified version of the Dwarf Mongoose Optimization Algorithm (DMOA) to optimize feature selection for AMD. The goal is to improve classification accuracy and reduce computation time when working with high-dimensional datasets. Based on the CICAndMal2017 dataset, this study focuses on generating, applying, and evaluating the proposed methodology.

Margins, column widths, line spacing, and type styles are The remaining sections of this paper is organized as follows: Section 2 covers the background of the study. Section 3 presents the proposed model. Section 4 provides the implementation process and analyzes the results. Section 5 concludes the findings and presents recommendations for future research.

2. Background

2.1 Malware

The term malware stands for "malicious software". Malware consists unauthorized programs designed by hackers or cyber criminals to perform dangerous activities. These malicious tools are designed for information theft, bypassing the devices' protection, and unauthorized access to a personal computer. The target device can be harmed, corrupting its data or even affecting the functionalities of the applications installed on the computer [8]. In addition, one of the main goals of adversaries is to turn the device into a zombie or slave as part of a botnet, which can be used in future attacks.

The first generation of malware is known as static malware and consists of several types identified by means of infection. Viruses hook on executable code and replicate the same as other programs do, and worms reproduce to extend the attack to more computers. Trojans present other valuable facilities that are unconventional features of trojan horses. Other types of first-generation malware include spyware, rootkits, crimeware, and adware. Even though these types of malware exhibit various types of malicious behavior, their architecture does not change and remains similar [9]. Second-generation malware or dynamic malware act differently, as they change after every infection to generate more versions of themselves but possess an exact destructive nature. This generation can be further divided on the basis of how it conceals itself to prevent identification of its signature: encrypted, oligomorphic, polymorphic, or metamorphic[10].Encrypted malware is malware that uses encryption to hide the actual code for malicious activity inside the general design of a program, which then takes malware out of

detection via host system platform techniques[11]. Oligomorphic malware uses two keys for each set of encryption and decryption on its payload. Encryption keys may frequently be replaced; thus, they are typically applied to obfuscate detection more than is the case with other individual kinds of encryption [12].Polymorphic malware employs individual keys for encryption and decryption, as depicted in oligomorphic keys. However, the decoder is even more difficult to detect, as several copies of it can be included inside an encrypted payload. Furthermore, multilevel encryption can be applied, which makes polymorphic malware much more challenging to identify than oligomorphic malware [9].Metamorphic malware differs from encrypted malware in that it does not use encryption. Instead, dynamic code modification, in which the code is replaced with a new one each time the program runs through its cycle, is used. This situation results in a new type of signature for every version of the malware, something that is almost impossible to identify easily [11].

2.2 Android Structure

Android, an operating system created by Google and based on the Linux kernel, is used in various devices, from smart phones to automobiles. Android apparatuses have touchscreens and are operated via swipe and tap, which makes them user friendly. Android is an open-source operating system, which means that developers can handle it by implementing their modifications via the Android Open-Source project. This flexibility has enabled Android to be the leading competitor to Apple's iOS [13]. The centroid is composed of four distinct layers: the application layer, framework layer, middle tier, and kernel layer [14] (Fig.1).

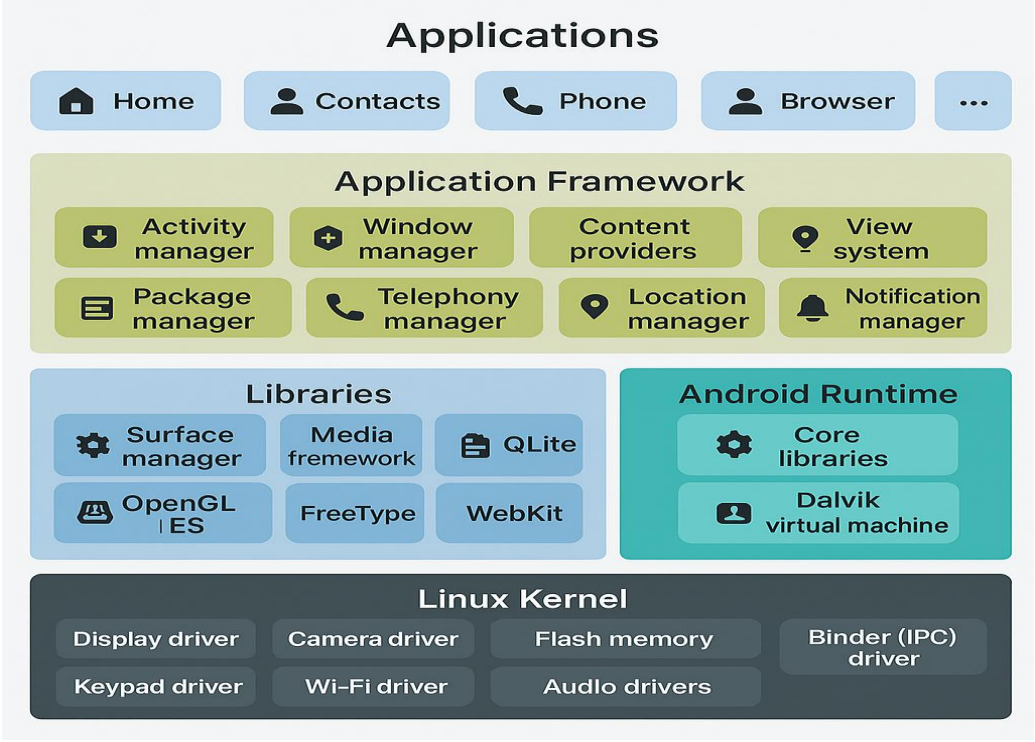

Fig.1: Android architecture [15].

2.2.1 Application Layer: The main components at this layer are the API framework, which works as Android's development platform, and the fundamental mechanism of the whole system. The application layer comprises daily use applications such as call logs, calendars, and many others, which are built and programmed in Java. This layer also involves collecting basic application packages such as e-mail clients and maps, browsers, and contacts [14].

2.2.2 Framework Layer: The framework layer offers the basic functionality that can be used to run applications. This layer facilitates important tasks such as application-to-application communication via content providers, voice call handling via the telephony manager, and identifying the device's location via the location manager. Without this layer, the applications cannot address each other, invoke calls, or access many standards, such as locational standards [12].

2.2.3 Middle Tier: The middle tier comprises libraries and the runtime environment found in the Android system. Most of these applications are in C++ and serve as the basis for several system elements. The middle tier mainly serves the application framework in delivering services to developers. Some of the components are SQLite for managing the database system, the open graphics library for managing the graphical user interface system, and the WebKit for controlling the web browsing system. The core library consists of the core Android API, which is mandatory for application development, and the basic Java API [14].

2.2.4 Linux Kernel Layer: The Linux kernel (version 2.6) is at the base of the architecture and is enhanced by approximately 115 patches. This kernel offers basic system functions such as control of processes, memory, and devices and the support of peripherals, including cameras and keyboards. This layer performs media operations and handles various device drivers to handle external devices [16].

The most susceptible layer is the application layer, which hosts a multitude of applications that are added by users. This layer is vulnerable to malware attacks because it is open source, and users can download the applications from a third-party provider. Some of these applications can be attacked by poorly built security features that enable malware to exploit loopholes or adjust user privileges. Therefore, the large number of applications that can be installed on the Android operating system and the extent to which their security differs make the platform highly susceptible to malware.

2.3 Malware Detection Methods

Malware detection methods are typically divided into three main parts: signature-based, behavior-based, and heuristic-based detection. These are classified into static, dynamic, and hybrid methods. Signature-based detection is pattern-oriented and uses a malware signature library containing unique features for each known Android malware. These signatures include file names, content strings, or bytes known or generated by individuals or machines. Evaluations

check whether the Android sample matches any signatures in the repository. This approach is popular due to its speed and efficiency. Most documented Android malware can be correctly identified. However, updating the signature library is labor intensive, and it only works against already observed malware [17].Heuristic-based detection applies ML and data mining to learn executable file behavior. Features such as API calls, CFGs, N-grams, and OpCodes are used [12]. It helps detect new and unknown threats but may lead to more false positives[17]. Behavior-based detection is widely used as it can detect unknown malware by analyzing behavioral patterns. These methods use ML and DL techniques to adapt and detect emerging Android threats. Since large data is analyzed, the false positive rate can be high, and detection may take longer[18]. Static analysis examines code without execution, helping prevent future harm, but cannot detect unknown malware. It is common in AMD due to its simplicity and speed. Dynamic analysis detects malware by monitoring activity post-execution, usually in a virtual environment for safety. Hybrid analysis combines both methods—first evaluating code signatures, then observing behavior during execution.

2.4 Machine Learning

ML is the process through which a machine is trained to process data efficiently. In some cases, meaningful insights may be difficult to obtain after reviewing data; this is where machine learning becomes useful. As a result of the increase in large datasets, the need for machine learning has also increased. ML has proven to be extremely useful in extracting valuable information. Therefore, many industries use ML to extract knowledge from data without human intervention[19]. Researchers have used supervised learning, unsupervised learning, and DL to detect Android malware. Supervised learning models are widely used in AMD; they learn from labeled training data and models to separate malicious Android applications from legitimate ones [20]. ML classifiers are an essential part of malware detection techniques, particularly in securing Android devices[21]. The following ML classifiers can be used for AMD:

2.4.1 Decision tree (DT): is a nonparametric supervised learning technique used for classification and regression. The purpose is to provide a framework for predicting the value of a target variable via basic decision rules derived from data features.

2.4.2 Random Forest (RF): This classifier combines decision trees generated from subsets of a dataset and makes the final decision on the basis of the most popular choice among these trees. One of the advantages of RF is its ability to work efficiently with datasets and accurately estimate information while addressing imbalances in data errors and pinpointing crucial features for analysis.

2.4.3 XGBoost: This optimized distributed gradient boosting library is highly efficient, adaptable, and portable. This tool employs ML methods within the gradient boosting framework and offers parallel tree boosting, thereby solving numerous data science issues rapidly and accurately.

2.4.4 K-Nearest Neighbors(KNN): KNN is an ML algorithm that can handle both classification and regression. This technique is performed by classifying a data point according to the prevailing class among its K neighbors in the feature space by using metrics such as the Euclidean distance for measurement purposes. Owing to the simplicity and interpretability of this technique, it has been applied across a wide range of areas.

2.5 Dwarf Mongoose Optimization Algorithm

The DMOA is a type of stochastic, population-based metaheuristic algorithm introduced by [22]. The DMOA works by adopting a sequence of structured phases that mimic the feeding and social behaviors of these mongooses. Below is a step-by-step breakdown of the DMOA process:

Step 1: Initialization. As outlined in Equation (1.1), the DMO initiates its process by generating a candidate population of mongooses. The candidates are randomly created within the lower and upper bounds (LB and UB) specific to the problem considered.

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d-1} & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & x_{i,j} & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d-1} & x_{n,d} \end{bmatrix}, \tag{1.1}$$

where X represents the current set of candidate solutions generated randomly, as described in Equation (1.2). $x_{\mathrm{i,j}}$ refers to the position of the jth dimension of the ith candidate; n denotes the population size; and d indicates the problem's dimensionality.The generation of candidate positions is expressed as

$$x_{i,j} = \text{unifrnd(VarMin, VarMax, VarSize)}, \tag{1.2}$$

where unifrnd produces a random number that follows a uniform distribution. $VarMin$ and $VarMax$ define the lower and upper bounds, respectively, and $VarSize$ corresponds to the dimensionality of the problem. Throughout each iteration, the best solution found is taken as the best solution obtained thus far.

Step 2: Alpha Group. The alpha female (α) oversees the family unit and is selected according to Equation (1.3).

$$\alpha = \frac{fit_i}{\sum_{i=1}^{n} fit_i}, \tag{1.3}$$

where fit represents the fitness value of an individual mongoose (solution) in the population that measures how well a solution satisfies the optimization problem's objective. iter (i) denotes the current iteration or step in the optimization process. n−bsn denotes the number of mongooses in the alpha group, where bs represents the number of babysitters. The term "peep" refers to the sound made by the alpha female, which helps guide the family in the right direction.

The position of the sleeping mound is influenced by the abundant food source and is given by

$$X_{i+1} = X_i + phi * \text{ peep }, \tag{1.4}$$

where phi is a random number uniformly distributed within the range [−1,1]. After each iteration, the sleeping mound is assessed as follows:

$$sm_i = \frac{\text{fit }_{i+1} - \text{ fit }_i}{\max\{|\ fit\ _{i+1},\ fit\ _i|\}}. \tag{1.5}$$

The average value is calculated when a sleeping mound is located as follows:

$$\varphi = \frac{\sum_{i=1}^{n} Sm_i}{n}. \tag{1.6}$$

Once the criteria for exchanging babysitters are met, the algorithm transitions to the scouting phase. The sleeping mound is subsequently evaluated on the basis of the availability of food sources.

Step 3: Scout Group. The scout group actively searches for the next sleeping mound to facilitate exploration, as mongooses typically do not return to previous sleeping mounds. In the DMOA, foraging and scouting occur simultaneously. The farther the family forages are, the greater the chance of discovering the next sleeping mound. , as simulated by the following:

$$X_{i+1} = \begin{cases} X_i - CF * \varphi\ *\ \text{rand}\ * \left[X_i - \vec{M}\right] & \text{if } \varphi_{i+1} > \varphi_i \\ X_i + \text{ CF*}\ \varphi\ *\ \text{rand}\ * \left[X_i - \vec{M}\right] & \text{else} \end{cases},$$

where rand is a random number in the range [0,1], and CF is defined as follows:

$$CF = \left(1 - \frac{iter}{Maxiter}\right)^{\left(2 \cdot \frac{iter}{Maxiter}\right)}, \tag{1.7}$$

where CF regulates the collective-volatile movement of the mongoose group, which decreases linearly as the number of iterations (iter) progresses. M represents the vector that guides the mongooses toward a new sleeping mound and is calculated as follows:

$$M = \sum_{i=1}^{n} \frac{(X_i - sm_i)}{X_i}. \tag{1.8}$$

Step 4: Babysitters Group.The babysitter group manages and cares for young dwarf mongooses. The remaining mongooses handle daily foraging tasks until the criteria for exchanging babysitters are met.

Step 5: Termination. The termination of the DMO algorithm is governed by specific criteria, which typically include limiting the number of generations (iterations). Algorithm 1 provides the pseudocode for the standard DMOA.

The DMOA offers several key advantages relevant to AMD. The DMO efficiently selects features and, therefore, is highly fit for the dimensionality reduction of large datasets, such as those employed in ML and malware detection systems [23]. Additionally, it assists in avoiding local optima, which is typical in most optimization problems. This task is accomplished using an effective search space [23]. Moreover, the DMOA generally converges faster than other

metaheuristic algorithms do. Fast convergence is advantageous when working on high-dimensional datasets, as the computation time becomes an issue [6]. Furthermore, the DMOA is flexible and dynamic, which enables it to address changing data and differing objective functions. Therefore, this scheme is suitable for complex multidimensional problems[24]. The efficiency, adaptability, and robust search make the DMOA an ideal choice for improving existing AMD systems. Compared with traditional Android malware systems, the DMOA optimizes feature relevance and model performance, enabling accurate and timely threat detection.

Algorithm 1: Dwarf Mongoose Optimization

Initialize DMO parameters: *(peep)*
Initialize the mongoose population of size N
Initialize bs (number of babysitters)
Set $N \leftarrow N - bs$
Set L
    For (iter = 1 to max_iter) do
        Compute fitness for current population
        Set C (based on current fitness values)
        Find α using Eq. (3)
        Find X_{i+1} using Eq.(4)
        Assess the new fitness of solution
        Assess sm using Eq. (5)
        Compute φ using Eq. (6)
        Compute $\vec{M}$ using Eq.(7)
        Exchange babysitters if $C \geq L$
        Initialize bs position
        Compute fitness $fit_i \leq \alpha$
        Simulate the scout mongoose's next position
    Update best solution found so far
    End for
Return best solution

2.6 Current Metaheuristic Methods in AMD

Recent advancements in dimensionality reduction techniques have significantly improved the detection and classification of AMD using ML algorithms. Several studies have explored the combination of nature-inspired metaheuristics and ML methods for Android malware detection. Taher et al. [25] introduced a hybrid model, DroidDetectMW, for Android malware detection and classification, which uses an enhanced artificial neural network (ANN) optimized by an Enhanced Hybrid Honeybee Optimization (EHHO) algorithm. The EHHO integrates surprise-pounce strategies and quasireflection-based learning to improve both exploration and exploitation phases. Feature selection was performed using metaheuristic optimization algorithms, such as Moth Flame Optimization (MFO) and enhanced Moth Flame

Optimization (EMFO). The detection phase utilized ML algorithms like RF and Support Vector Machines (SVM) to classify Android app behaviors. Experimentation on the CICAndMal2017 dataset showed significant improvements in precision, recall, and classification accuracy, achieving 98.1% accuracy in binary malware classification and 96.9% in malware category classification. Aldehim et al.[26] proposed the GBWODL-AMC method, which combines Gauss-mapping black widow optimization (GBWO) for feature selection and deep learning for Android malware classification. The method used the Deep Extreme Learning Machine (DELM) model for classification and Ant Lion Optimization (ALO) to optimize model parameters. Testing on the CICAndMal2017 dataset revealed that the GBWODL-AMC approach outperformed other malware detection methods, achieving 98.59% accuracy in binary classification and 98.50% in multiclass classification. Smmarwar et al. [27] introduced OEL-AMD, using the Black Widow Optimization (BGWO) algorithm for feature selection. The optimal features were then used for training various benchmark classification algorithms. The method was evaluated on the CICInvesAndMal2019 dataset, yielding a best classification accuracy of 96.95% for binary classification using static features and 83.49% for multiclass classification using dynamic features. Despite the success of these methods, challenges remain, particularly in reducing redundant information while maintaining high accuracy. For instance, while the GBWODL-AMC method shows high accuracy, it increases system complexity due to the combination of GBWO for feature selection, DELM for classification, and ALO for optimization, resulting in high computational costs and longer processing times. This makes the approach less suitable for real-time applications. Furthermore, although metaheuristic optimization techniques are widely used, there has been limited research on feature selection for AMD using the DMOA.

3. Methodology

This section covers the dataset, preprocessing steps, and a comparison between the original and modified DMOA algorithms.

3.1 Dataset

This study utilizes the CICAndMal2017 (Android malware dataset) developed by the Canadian Center for Cybersecurity at the University of New Brunswick [28]. The dataset is divided into two main categories: benign applications and malware. It consists of 10,854 samples, with 6,500 benign (59.9%) and 4,354 malware (40.1%) samples. The malware samples are further classified into four primary families. The four subtypes of malware (Ransomware, Adware, Scareware, and SMSmalware) form 42 different attack types[29].

- Adware is software that is designed to display unwanted advertisements to increase clicks and views. It comprises 104 applications, including the Ewind, Selfmite, and Gooligan families.

- Ransomware is a malicious application that seeks to prevent access to computer resources. It comprises 101 applications for Android devices, such as Charger, Jisut, Koler, and WannaLocker.
- Scareware is malware that compels users to purchase unnecessary and potentially malware applications. It comprises 102 applications, such as AndroidDefender, FakeAV, and FakeApp.
- SMSmalware is an unauthorized call or text message sent to others without the mobile owner's permission [30]. It consists of 99 applications, including Ji Fake, Bean Bot, Nandrobox, Fake Mart, Fake Notify, and SMS sniffer families.

The dataset's features include fundamental information about the behavior of malicious and benign applications, as well as precise network traffic flow characteristics in mobile devices. These features are critical for understanding how different malware kinds interact with network protocols and how to differentiate them based on network activity.

3.2 Proposed Methodology

The DMOA incorporates multiple classification algorithms for Android malware detection on the basis of the CICAndMal2017 dataset. The CICAndMal2017 dataset includes over 80 features, some of which may be irrelevant; thus, it might negatively impact model performance. This issue is addressed by integrating feature selection to identify the most relevant features for use as inputs for various classification algorithms. Figure 2 shows the main phases of the proposed model.

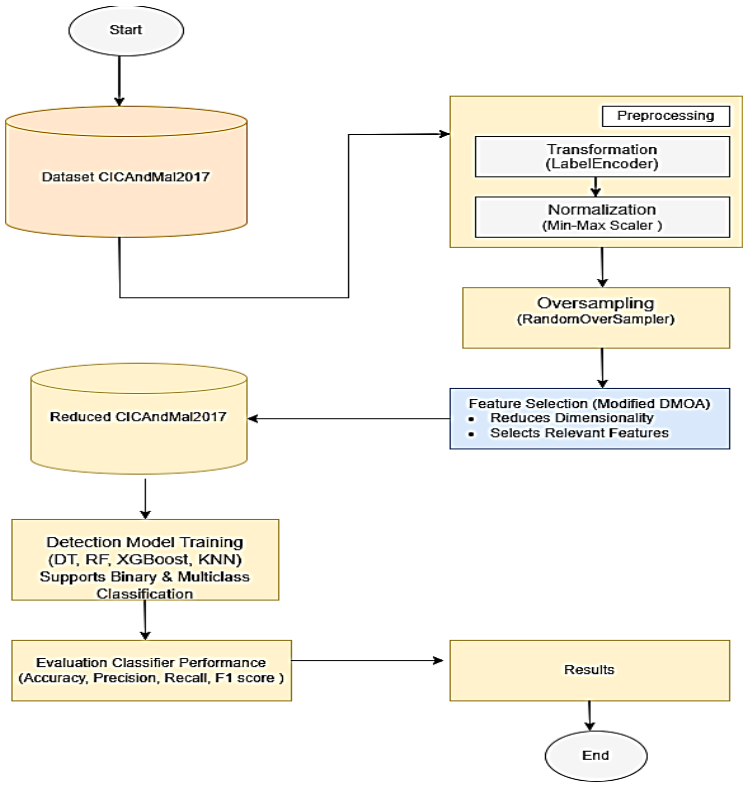

Fig 2: Proposed model.

Data preprocessing is an essential step in the proposed model. The process includes two main steps: data transformation and data normalization.

3.2.1 Data Transformation

Data transformation is an essential stage in the preprocessing of ML models; it acts as an encoder, converting the given labels into categorical values. In multiclass classification, label encoding assigns a numerical integer to every class. In this manner, the model can differentiate between various forms of malware. Table 1 shows the label encoding mapping for multiclass classification in the CICAndMal2017 dataset.

Table 1: Label Encoding Mapping

Original Label	Encoded Label
Benign	0
Adware	1
Ransomware	2
SMS malware	3
Scareware	4

3.2.2 Normalization

In this study, the goal of normalization is to ensure that each feature has an equal impact on the model's performance, especially when the features have different units or ranges. For the CICAndMal2017 dataset, normalization was achieved through min-max scaling. This process centers the data around 0 and scales it to have a unit variance. The min-max scaling formula used is:

$$\mathrm{X}_{norm} = \frac{\mathrm{X} - \mathrm{X}_{\mathrm{Min}}}{\mathrm{X}_{\mathrm{Max}} - \mathrm{X}_{\mathrm{Min}}}, \tag{3.1}$$

Where *X_norm* is the normalized value, *X* is the original value, $\mathrm{X}_{\mathrm{Min}}$ is the minimum value, and $\mathrm{X}_{\mathrm{Max}}$ is the maximum value within the feature column. This scaling method was applied to 83 features in the dataset.

3.2.3 Oversampling

The total number of samples for each class of malware in the CICAndMal2017 varies significantly, causing an imbalance in the classification process. ML models trained on imbalanced data tend to favor classes with larger sample sizes, which may result in erroneous predictions for dominating classes. The resulting bias also results in unrealistically high accuracy and misclassification of underrepresented classes, rendering the model worthless on the test set, particularly for rare malware classes. Biases can be avoided through oversampling, which involves creating an equal number of samples for each type. This work replicates samples from underrepresented classes to provide a balanced dataset and improve model effectiveness.

3.2.4 *Feature Selection*

With many potential features from the dataset used in this work, the appropriate feature must be selected. Feature selection is the operation of choosing the most relevant features from a high-dimensional dataset to utilize in a ML model. Choosing the most relevant features improves model performance [31]. Figure 3 shows a simplified illustration of the feature selection process. The selection process selects only relevant features and discards irrelevant features, thus increasing the effectiveness of our assessment process.

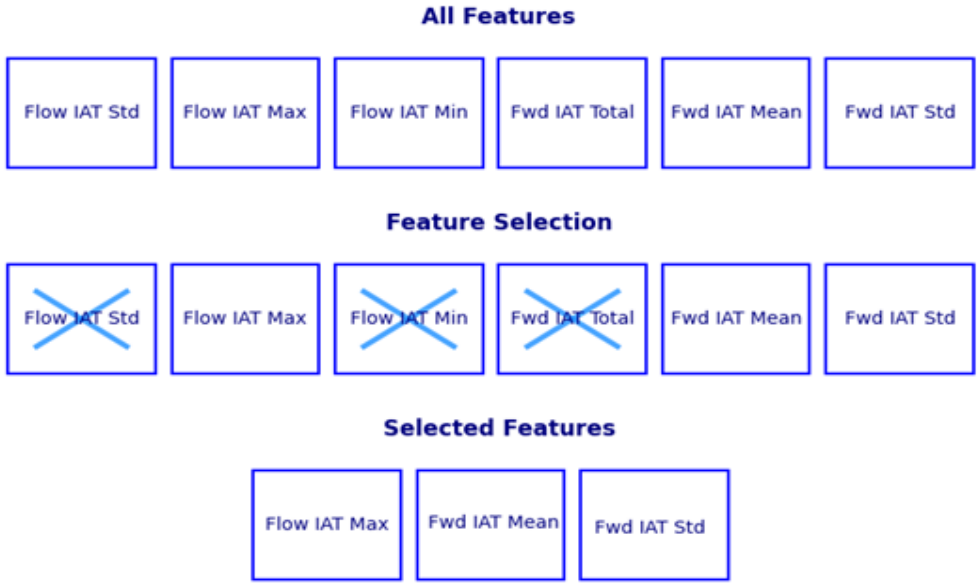

Fig 3: Simple representation for feature selection.

The DMOA is effective for feature selection, particularly when used with high-dimensional datasets [32], and it is suitable for AMD. Because of its adaptive search capabilities, the DMOA is effective at feature selection and other optimization tasks. However, the DMOA, like many other optimization approaches, presents limitations in terms of convergence to local optima. This study improves the DMOA's performance by using adaptive mechanisms such as crossover operations and mutation processes. These enhancements allow the system to adjust and optimize itself, examining alternatives more thoroughly for better results. These improvements also assist the system diversify its search process, allowing the algorithm to avoid local optima and reveal superior feature subsets. Algorithm 2 illustrates the pseudocode for the proposed modified DMOA.

Adding crossover and mutation functions to the DMOA increases the exploration of the feature space, thereby improving the exploration capabilities of the DMOA and helping the algorithm avoid local optima. The addition of these operators increases the diversity within the solution population. In malware detection, this step is crucial because the feature space is often complex and high-dimensional. Crossover enables a set of successful feature combinations to be

explored, and mutation helps to avoid keeping the algorithm fixed by allowing the exploration of new relevant features. This dynamic approach may enhance the detection capabilities.

Algorithm 2: Proposed modified DMOA.

Initialize a population P of n_pop individuals (binary vectors)
Evaluate the fitness of each individual in P using fitness_function
Set Best_solution as the individual with the highest fitness in P
For iter = 1 to n_iter do
Step 1: Alpha Selection:
- Identify Alpha = best individual in current P

Step2: Crossover Phase:
For each individual iii in P:
- Randomly select two parents p_1 and p_2 from P.
- Perform a single-point crossover operation on p_1 and p_2 to generate a child ccc.
  - With probability mutation_rate, mutate the bits in ccc by flipping randomly selected bits.
- Add ccc to the new population P_new.

End For
Step 3:Scout Phase:
With probability scout_rate, add a new random individual to P_new
Step 4: Evaluate fitness of all individuals in P_new
Step 5: Update Best_solution if any individual in P_new has higher fitness
Step 6: Replace P with P_new
End For
Return Best_solution

3.2.5 Classification

The classification stage is vital in the AMD model, distinguishing between benign and malicious network activity. After selecting the most relevant features, machine learning classifiers are trained on labeled data to detect malware patterns and tested on unseen data to assess performance. Multiple classifiers—DT, RF, KNN, and XGBoost—were used to enhance detection accuracy. Each offers unique strengths: DTs provide interpretability, RF improves accuracy and prevents overfitting, KNN detects subtle differences in malware, and XGBoost handles complex datasets through gradient boosting.

4. Implementation and results

This section provides the implementation process and analyzes the results of implementing the model using different ML algorithms, comparing them to other metaheuristic algorithm models.

4.1 Implementation Environment

The study was carried out on a Windows platform using the VSC IDE, which includes a comprehensive set of tools for building, verifying, and debugging code. Anaconda is a Python package that contains a variety of prominent data science and machine learning libraries that are easily integrated with VSC. Python's most recent version (V3.12) was selected for model

implementation and evaluation due to its broad ML support and the availability of numerous data analysis packages. The proposed model was implemented using a high-performance computer.

4.2 Implementation Operations

The development of the proposed modified DMOA model includes several significant stages, each of which contributed to the development of an effective and reliable system. The implementation started with the collection of the CICAndMal2017 dataset from the Canadian Institute for Cybersecurity website. The first step in data preprocessing involves transforming categorical values (e.g., "BENIGN", "ADWARE", and "SMS MALWARE") into numerical representations via the LabelEncoder from the sklearn.preprocessing module. For multiclass classification, the LabelEncoder fit_transform() method analyzes the unique class labels and assigns each one a unique integer (e.g., "BENIGN" becomes 0, and "ADWARE" becomes 1). This transformation ensures that the target variable is in a numerical format that ML algorithms can process. The encoded labels are used for model training and evaluation. The second stage of preprocessing starts with data normalization via the min–max scaler from the (sklearn.preprocessing) ML library to scale the features with large numerical values into the range of [0, 1]. With this scheme, the ML algorithms and DMOA perform better because they assume that all the features are on the same scale. After the transformation and normalization phases are completed, oversampling (the third stage) begins. This step involves the use of RandomOverSampler to balance the dataset by increasing the number of samples in the minority class. This scheme ensures that the ML model does not bias toward the majority class, thereby improving the model's ability to generalize across all classes.

After that, the modified DMOA was applied to select the most relevant features based on a fitness function that considers accuracy and the number of selected features. The fitness function was defined to calculate the error rate , considering both the prediction accuracy and the number of selected features .A lower fitness value (error rate) indicates a better feature subset. As a result, forty-eight features were selected from 83 features.

4.3 Evaluation Metrics

The proposed model was assessed using various evaluation metrics commonly applied in research to evaluate the performance of AMD. The following metrics provide insights into how well the model can classify data, its correctness, and its efficiency in classifying malware:

- *Accuracy* is expressed as the ratio of correct detections (TP and TN) to the total number of detections (N), where N = (TP+TN+FP+FN). It determines the model's overall correctness in detecting benign and malware instances.
- *Precision* is the ratio of the correct TP to the total number of predicted positives (TP+FP). It represents the model's ability to identify positive instances accurately.

- *Recall* is the ratio of the accurate TP for a given class divided by the class's total sample size. Recall refers to the model's ability to determine exactly benign classes that the model can correctly identify.
- *F1 score* is a balancing measure of a model's precision and recall. This indicator is especially important when dealing with inconsistent datasets in which one class may be underrepresented.
- *Convergence Time:* This metric refers to the speed with which the model is trained. Evaluating the model via these various metrics provides an extensive overview of its effectiveness and correctness in predicting malware and benign instances.
- *Confusion matrix* (CM): This metric is a table that is frequently used to gain insight into the performance of a model (TP, FP, FN, or TN) across all classes of the dataset.

4.4 Results

The proposed model was evaluated using ML algorithms. The algorithms used are DT, RF, KNN, and XGBoost. The RF classifier demonstrates exceptional performance in classifying malware families within the AMD dataset, achieving nearly perfect metrics across the board. With an accuracy of 1.00 and precision and an F1 score of 0.999882, the RF model effectively distinguishes between various malware classes with minimal errors. This precise performance is indicative of the ability of the RF classifier to perform complex multiclass classification in a rather robust manner, especially for various malware types. Figure 4 shows the CM for the RF classifier in multiclass classification of malware families.

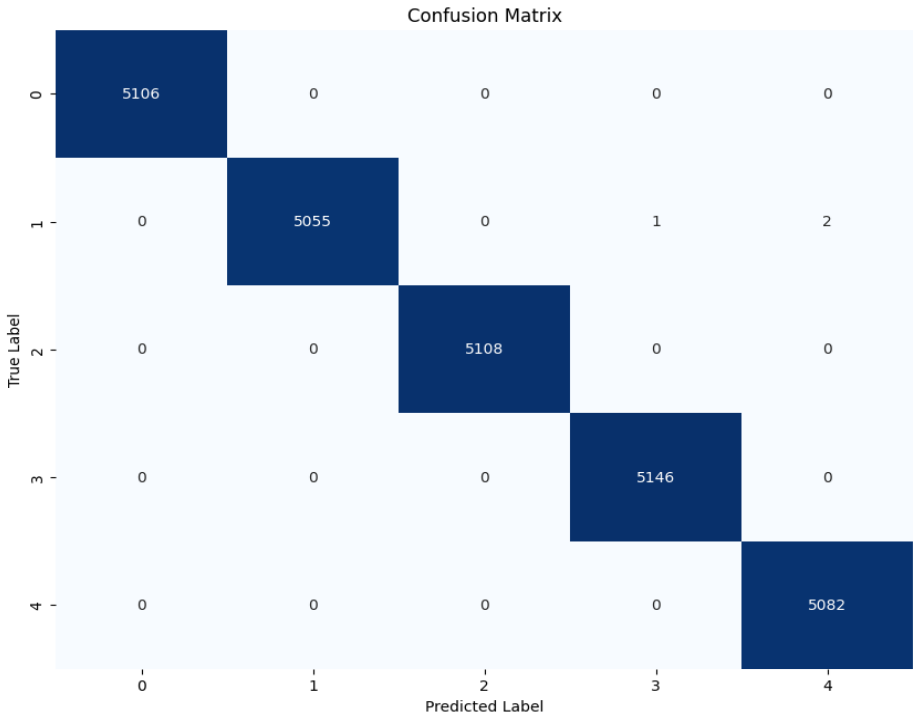

Fig 4: RF classifier CM multiclass classification

The DT classifier results indicate exceptional performance across all the metrics. The DT achieved an accuracy of 1.000, correctly classifying all instances without any errors. The classifier also measured precision and an F1 score of 0.999961, indicating a near-perfect balance between precision and recall. Figure 5 shows the confusion matrix results for the DT classifier in the multiclass classification of malware families. The DT classifier successfully classified nearly all malware classes with a single misclassification, showing a remarkable alignment between the actual and predicted labels. This finding proves the ability of the DT classifier to distinguish between various malware families with minimal error.

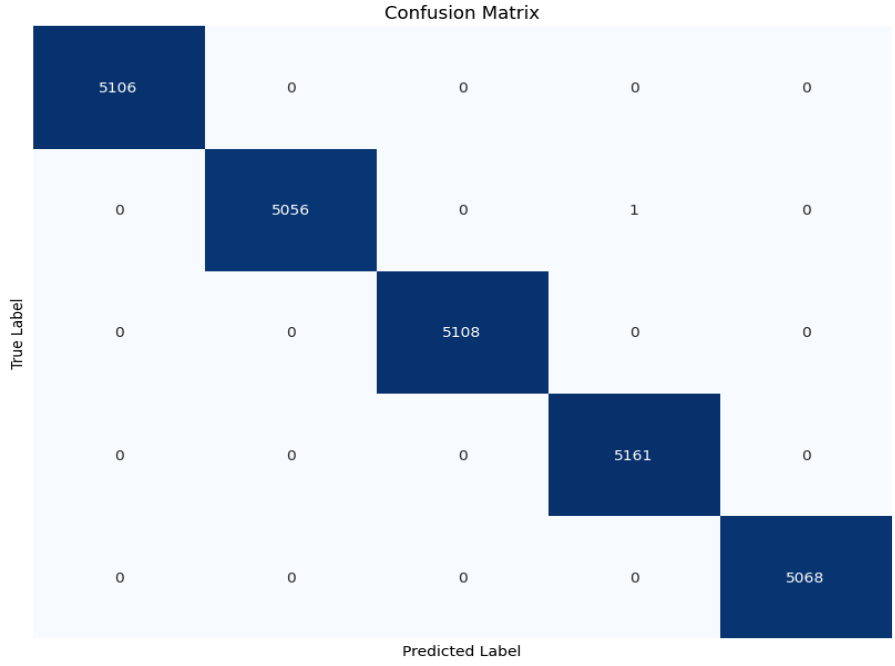

Fig 5: DT classifier CM multiclass classification

XGBoost performed exceptionally well, nearing perfect accuracy, precision, and F1 scores, although its convergence time was greater than those of DT and RF. KNN had the lowest accuracy and took the longest to converge, suggesting that it may be less effective for this task than the other classifiers are. Overall, DT and RF were the most efficient and accurate classifiers in this evaluation. Table 2 provide a comprehensive summary of all the classifiers employed in the modified DMOA model for multiclass classification in terms of average accuracy, precision, F1 score and convergence time.

Table 2: Multiclass Classification Results

Classifier	Accuracy	Precision	F1-score	Time(s)
RF	1.000000	0.999882	0.999882	1330.01
DT	1.000000	0.999961	0.999961	1109.48
KNN	0.995216	0.984838	0.984219	3223.30
XGBoost	0.999882	0.999765	0.999764	1928.08

The results for the original DMOA and the proposed modified DMOA are compared. The proposed modified DMOA improves accuracy across some ML algorithms in malware family multiclass classification. The RF and DT classifiers have the highest accuracy of 1.00, increasing from 0.9467 and 0.9025, respectively, when the original DMOA is used. XGBoost improves to 0.9999 from 0.9500. KNN increases to 0.9952 from 0.9696. Figure 6 shows the accuracy comparison between the proposed modified DMOA and the original DMOA in multiclass classification.

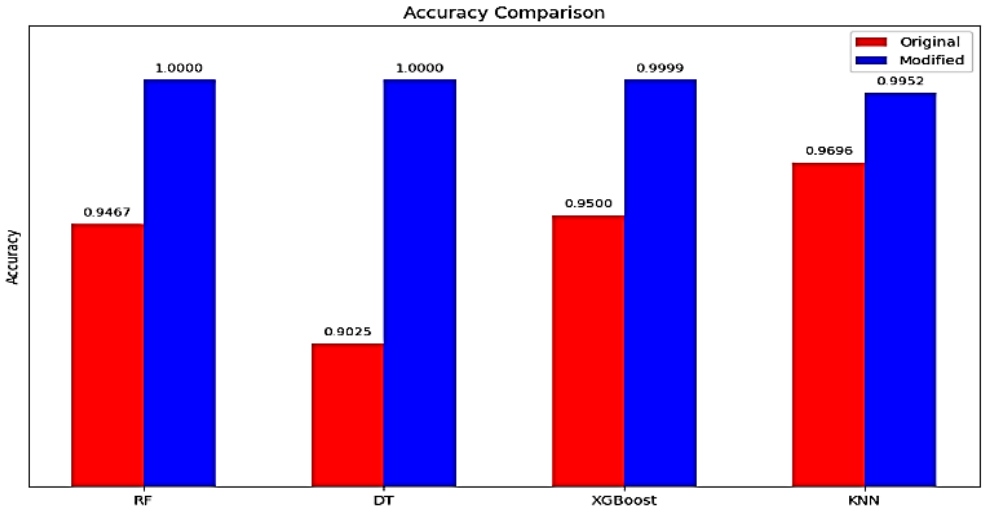

Fig 6: Accuracy comparison

The proposed modified DMOA demonstrates constant improvements in precision for the ML algorithms in multiclass classification. The RF improves to 0.9999 from 0.9063 in the original DMOA. DT increases to 1.00 from 0.9290, XGBoost increases to 0.9998 from 0.9455, and KNN increases to 0.9848 from 0.9561. Figure 7 shows the precision comparison between the proposed modified DMOA and the original DMOA in multiclass classification.

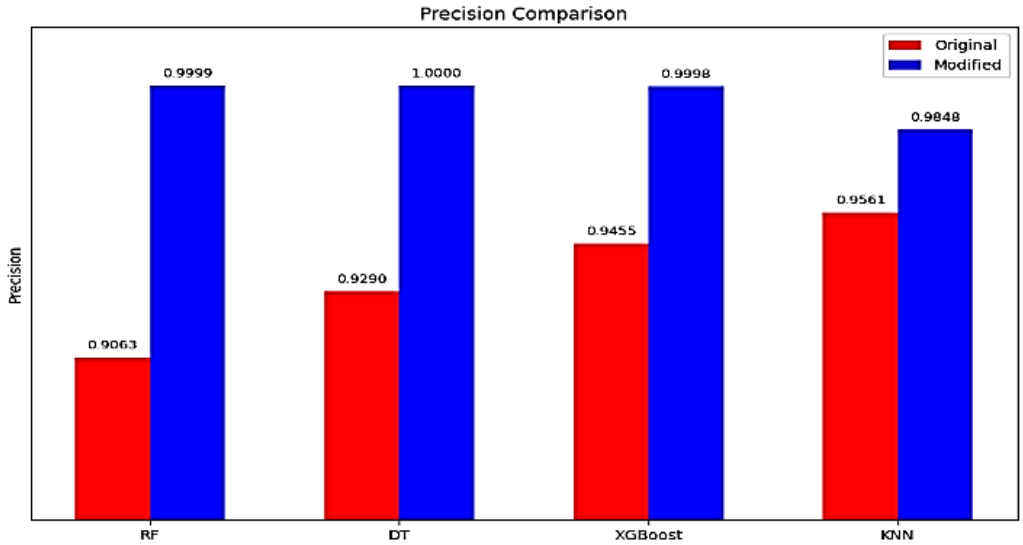

Fig 7: Presision comparison

The proposed modified DMOA yields improvements in the F1 score for the RF, DT, XGBoost, and KNN classifiers in multiclass classification. The RF achieves an F1 score of 0.9999, improving from 0. 8832. Furthermore, the DT improves to 1.00 from 0.8978, XGBoost increases to 0.9998 from 0.9438, and the KNN improves to 0.9842 from 0.9512. Figure 8 shows the F1 score comparison between the proposed modified DMOA and the original DMOA in multiclass classification

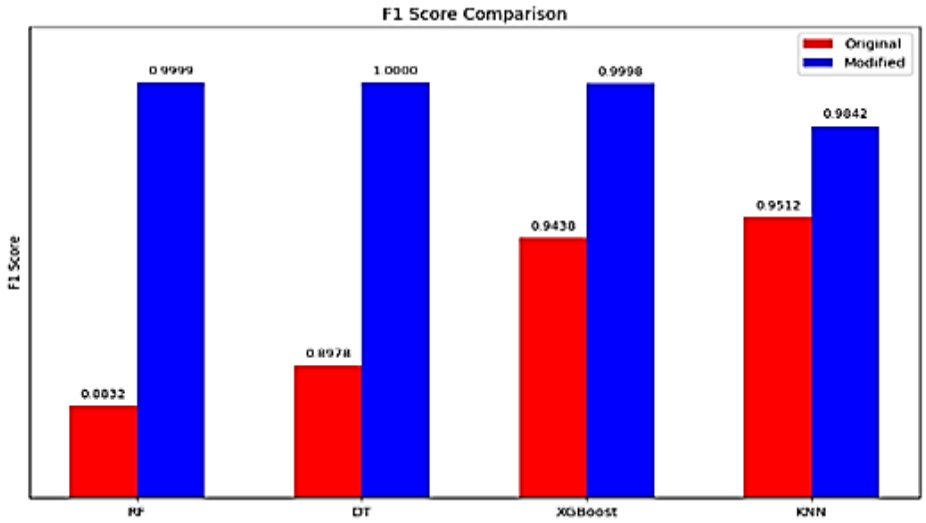

Fig 8: F1 score comparison

The convergence time comparison highlights the enhanced efficiency of the proposed modified DMOA across all classifiers (Figure 9.). The DT experienced a substantial reduction in convergence time, decreasing by 52.83 s from 1162.31 s in the original DMOA to 1109.48 s in the modified DMOA. The KNN classifier showed a significant improvement, with the convergence time reduced from 5071.01 to 3223.30 s, reflecting a reduction of 1847.71 s. For XGBoost, the convergence time decreased from 2292.97 to 1928.08 s, a reduction of 364.89 s. These results underscore the efficiency gains achieved by the modified DMOA and its superior performance in multiclass classification tasks.

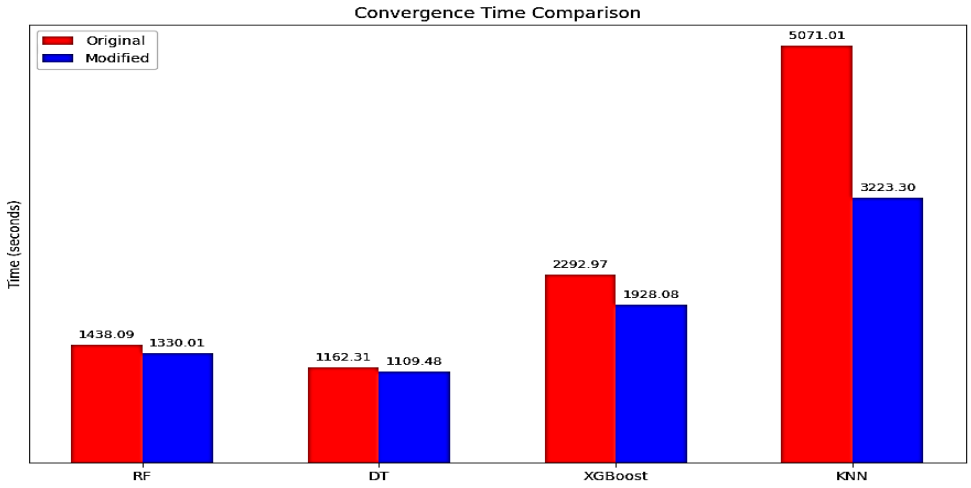

Fig 9: Convergence time comparison

The modified DMOA model was evaluated against other studies that used the CICAndMal2017 dataset. It outperforms all the compared models in multiclass classification. RF has an overall accuracy of 100%, surpassing the GBWO model of Aldehim et al. [26], with a score of 98.50%, and the MVO and EMFO models of Taher et al.[25], with a score of 96.9%. Figure 10 shows an accuracy comparison between the proposed modified DMOA and other models in the multiclass classification of malware. The results further confirm the superior performance of the proposed model, particularly when the DT and RF classifiers are used. This finding reinforces the effectiveness of the modified DMOA in handling multiclass classification tasks, further demonstrating its ability to achieve higher accuracy than existing approaches.

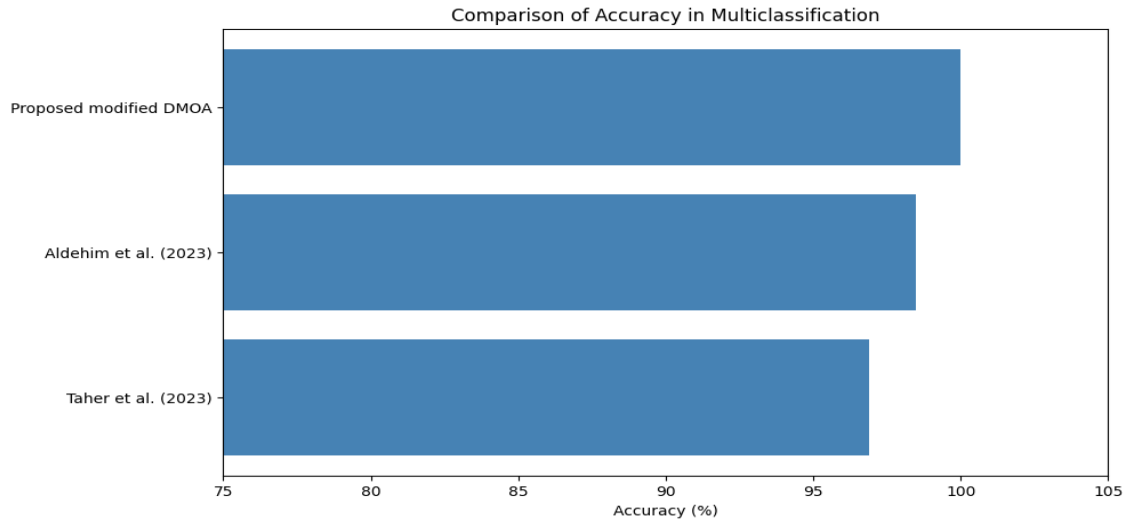

Fig 10: Accuracy comparison between the proposed modified DMOA and other models

5. Conclusion

This study introduced a modified DMOA to enhance AMD through improved feature selection. By addressing the limitations of traditional ML methods and existing optimization algorithms—such as poor performance on high-dimensional data and vulnerability to local optima—the proposed model integrates adaptive strategies like crossover and mutation to expand search capabilities and boost classification accuracy. Using the CICAndMal2017 dataset, the model was tested across multiple classifiers, including DT, RF, KNN, and XGBoost. The results show the effectiveness of the proposed model. The modified DMOA significantly outperforms the original version and several state-of-the-art models in terms of accuracy, precision, F1 score, and convergence time. DT and RF achieved 100% accuracy, highlighting the model's capability to distinguish between different malware types accurately. Overall, the modified DMOA model proves to be a robust and reliable approach for AMD. Its ability to maintain high performance across diverse evaluation metrics and classifiers underscores its adaptability and efficiency. Future

work may enhance the model's scalability and test its performance on other complex and evolving malware datasets to ensure continued relevance in real-world applications.

Conflicts of Interest: The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] P. Agrawal, H.F. Abutarboush, T. Ganesh, A.W. Mohamed, Metaheuristic Algorithms on Feature Selection: A Survey of One Decade of Research (2009-2019), IEEE Access 9 (2021), 26766-26791. https://doi.org/10.1109/access.2021.3056407.

[2] A. Sabbah, A. Taweel, S. Zein, Android Malware Detection: A Literature Review, in: Communications in Computer and Information Science, Springer, Singapore, 2023: pp. 263-278. https://doi.org/10.1007/978-981-99-0272-9_18.

[3] F. Nawshin, R. Gad, D. Unal, A.K. Al-Ali, P.N. Suganthan, Malware Detection for Mobile Computing Using Secure and Privacy-Preserving Machine Learning Approaches: A Comprehensive Survey, Comput. Electr. Eng. 117 (2024), 109233. https://doi.org/10.1016/j.compeleceng.2024.109233.

[4] R.K. Varma P, S.K.R. Mallidi, S. Jhansi K, P. Latha D, Bat Optimization Algorithm for Wrapper-based Feature Selection and Performance Improvement of Android Malware Detection, IET Netw. 10 (2021), 131-140. https://doi.org/10.1049/ntw2.12022.

[5] N. Honest, A Survey on Feature Selection Techniques, GIS Sci. J. 7 (2020), 353–358.

[6] O.A. Akinola, J.O. Agushaka, A.E. Ezugwu, Binary Dwarf Mongoose Optimizer for Solving High-Dimensional Feature Selection Problems, PLOS ONE 17 (2022), e0274850. https://doi.org/10.1371/journal.pone.0274850.

[7] P.S. Game, V. Vaze, E. M, Bio-inspired Optimization: Metaheuristic Algorithms for Optimization, arXiv:2003.11637 (2020). https://doi.org/10.48550/arXiv.2003.11637.

[8] J. El Abdelkhalki, M.B. Ahmed, B.A. Abdelhakim, Image Malware Detection Using Deep Learning, Int. J. Commun. Netw. Inf. Secur. 12 (2022), 180-189. https://doi.org/10.17762/ijcnis.v12i2.4600.

[9] S.K. Sahay, A. Sharma, H. Rathore, Evolution of Malware and Its Detection Techniques, in: Advances in Intelligent Systems and Computing, Springer Singapore, Singapore, 2019: pp. 139-150. https://doi.org/10.1007/978-981-13-7166-0_14.

[10] M.N. Alenezi, H.K. Alabdulrazzaq, A.A. Alshaher, M.M. Alkharang, Evolution of Malware Threats and Techniques: A Review, Int. J. Commun. Netw. Inf. Secur. 12 (2022), 326-337. https://doi.org/10.17762/ijcnis.v12i3.4723.

[11] O. Aslan, R. Samet, A Comprehensive Review on Malware Detection Approaches, IEEE Access 8 (2020), 6249-6271. https://doi.org/10.1109/access.2019.2963724.

[12] V.N. Uzel, Detecting Android Malware by Using Fuzzy Set-Based Weighting Method and Firefly Optimization Algorithm, Master's Thesis, Hacettepe University, 2022.

[13] N. Ekanayake, Android Operating System, 2018. https://www.researchgate.net/publication/325257105.

[14] Y. Chen, Research on Android Architecture and Application Development, J. Phys.: Conf. Ser. 1992 (2021), 022168. https://doi.org/10.1088/1742-6596/1992/2/022168.

[15] A. Subramanian, Exploring the Layers: A Deep Dive into Android OS Architecture, 2023. https://medium.com/@ayyappansubramanian77/exploring-the-layers-a-deep-dive-into-android-os-architecture-31a2cd7a4036.

[16] M. Jaiswal, Android the Mobile Operating System and Architecture, Int. J. Creative Res. Thoughts 6 (2018), 514-525.

[17] L. Meijin, F. Zhiyang, W. Junfeng, C. Luyu, Z. Qi, et al., A Systematic Overview of Android Malware Detection, Appl. Artif. Intell. 36 (2021), 2007327. https://doi.org/10.1080/08839514.2021.2007327.

[18] F.A. Almarshad, M. Zakariah, G.A. Gashgari, E.A. Aldakheel, A.I.A. Alzahrani, Detection of Android Malware Using Machine Learning and Siamese Shot Learning Technique for Security, IEEE Access 11 (2023), 127697-127714. https://doi.org/10.1109/access.2023.3331739.

[19] B. Mahesh, Machine Learning Algorithms - A Review, Int. J. Sci. Res. 9 (2020), 381-386. https://doi.org/10.21275/art20203995.

[20] N. Chowdhury, A. Haque, H. Soliman, M.S. Hossen, T. Fatima, et al., Android Malware Detection Using Machine Learning: A Review, in: Lecture Notes in Networks and Systems, Springer, Cham, 2024, pp. 507-522. https://doi.org/10.1007/978-3-031-47715-7_35.

[21] P. Agrawal, B. Trivedi, Machine Learning Classifiers for Android Malware Detection, in: Advances in Intelligent Systems and Computing, Springer, Singapore, 2020, pp. 311-322. https://doi.org/10.1007/978-981-15-5616-6_22.

[22] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Dwarf Mongoose Optimization Algorithm, Comput. Methods Appl. Mech. Eng. 391 (2022), 114570. https://doi.org/10.1016/j.cma.2022.114570.

[23] G. Moustafa, A.M. El-Rifaie, I.H. Smaili, A. Ginidi, A.M. Shaheen, et al., An Enhanced Dwarf Mongoose Optimization Algorithm for Solving Engineering Problems, Mathematics 11 (2023), 3297. https://doi.org/10.3390/math11153297.

[24] S. Fu, H. Huang, C. Ma, J. Wei, Y. Li, et al., Improved Dwarf Mongoose Optimization Algorithm Using Novel Nonlinear Control and Exploration Strategies, Expert Syst. Appl. 233 (2023), 120904. https://doi.org/10.1016/j.eswa.2023.120904.

[25] F. Taher, O. AlFandi, M. Al-kfairy, H. Al Hamadi, S. Alrabaee, DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection, Appl. Sci. 13 (2023), 7720. https://doi.org/10.3390/app13137720.

[26] G. Aldehim, M.A. Arasi, M. Khalid, S.S. Aljameel, R. Marzouk, et al., Gauss-mapping Black Widow Optimization with Deep Extreme Learning Machine for Android Malware Classification Model, IEEE Access 11 (2023), 87062-87070. https://doi.org/10.1109/access.2023.3285289.

[27] S.K. Smmarwar, G.P. Gupta, S. Kumar, P. Kumar, An Optimized and Efficient Android Malware Detection Framework for Future Sustainable Computing, Sustain. Energy Technol. Assessments 54 (2022), 102852. https://doi.org/10.1016/j.seta.2022.102852.

[28] University of New Brunswick (UNB), Android malware dataset (CIC-AndMal2017), https://www.unb.ca/cic/datasets/andmal2017.html, Accessed: Oct. 26, 2024.

[29] A.H. Lashkari, A.F.A. Kadir, L. Taheri, A.A. Ghorbani, Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification, in: 2018 International Carnahan Conference on Security Technology (ICCST), IEEE, 2018, pp. 1-7. https://doi.org/10.1109/CCST.2018.8585560.

[30] M. Abuthawabeh, K. Mahmoud, Enhanced Android Malware Detection and Family Classification, Using Conversation-Level Network Traffic Features, Int. Arab. J. Inf. Technol. 17 (2020), 607-614. https://doi.org/10.34028/iajit/17/4a/4.

[31] J. Barrera-García, F. Cisternas-Caneo, B. Crawford, M. Gómez Sánchez, R. Soto, Feature Selection Problem and Metaheuristics: A Systematic Literature Review About Its Formulation, Evaluation and Applications, Biomimetics 9 (2023), 9. https://doi.org/10.3390/biomimetics9010009.

[32] M. Elaziz, A. Ewees, M. Al-qaness, S. Alshathri, R. Ibrahim, Feature Selection for High Dimensional Datasets Based on Quantum-Based Dwarf Mongoose Optimization, Mathematics 10 (2022), 4565. https://doi.org/10.3390/math10234565.

[33] M.M. Abualhaj, S. Al-Khatib, M.O. Hiari, Q.Y. Shambour, Enhancing Spam Detection Using Hybrid of Harris Hawks and Firefly Optimization Algorithms, J. Soft Comput. Data Min. 5 (2024), 161–174. https://doi.org/10.30880/jscdm.2024.05.02.012.

[34] M.M. Abualhaj, Spam Feature Selection Using Firefly Metaheuristic Algorithm, J. Appl. Data Sci. 5 (2024), 1692-1700. https://doi.org/10.47738/jads.v5i4.336.

[35] M.M. Abualhaj, A.A. Abu-Shareha, S. Nabil Alkhatib, Q.Y. Shambour, A.M. Alsaaidah, Detecting Spam Using Harris Hawks Optimizer as a Feature Selection Algorithm, Bull. Electr. Eng. Inform. 14 (2025), 2361-2369. https://doi.org/10.11591/eei.v14i3.9198.

[36] Y. Sanjalawe, S. Fraihat, S. Al-E'Mari, M. Abualhaj, S. Makhadmeh, et al., A Review of 6g and Ai Convergence: Enhancing Communication Networks with Artificial Intelligence, IEEE Open J. Commun. Soc. 6 (2025), 2308-2355. https://doi.org/10.1109/ojcoms.2025.3553302.

[37] Y. Sanjalawe, S. Fraihat, M. Abualhaj, S.R. Al-E'Mari, E. Alzubi, Hybrid Deep Learning for Human Fall Detection: A Synergistic Approach Using Yolov8 and Time-Space Transformers, IEEE Access 13 (2025), 41336-41366. https://doi.org/10.1109/access.2025.3547914.

[38] Y. Sanjalawe, S. Al-E'mari, S. Fraihat, M. Abualhaj, E. Alzubi, A Deep Learning-Driven Multi-Layered Steganographic Approach for Enhanced Data Security, Sci. Rep. 15 (2025), 4761. https://doi.org/10.1038/s41598-025-89189-5.

[39] M.M. Abualhaj, Q.Y. Shambour, A.A. Abu-Shareha, S.N. Al-Khatib, A. Amer, Enhancing Malware Detection Through Self-Union Feature Selection Using Gray Wolf Optimizer, Indones. J. Electr. Eng. Comput. Sci. 37 (2025), 197-205. https://doi.org/10.11591/ijeecs.v37.i1.pp197-205.

[40] S. Fraihat, Q. Shambour, M.A. Al-Betar, S.N. Makhadmeh, Variational Autoencoders-Based Algorithm for Multi-Criteria Recommendation Systems, Algorithms 17 (2024), 561. https://doi.org/10.3390/a17120561.

[41] Q. Shambour, Artificial Intelligence Techniques for Early Autism Detection in Toddlers: A Comparative Analysis, J. Appl. Data Sci. 5 (2024), 1754-1764. https://doi.org/10.47738/jads.v5i4.353.

[42] M. MADI, F. JARGHON, Y. FAZEA, O. ALMOMANI, A. SAAIDAH, Comparative Analysis of Classification Techniques for Network Fault Management, Turk. J. Electr. Eng. Comput. Sci. 28 (2020), 1442-1457. https://doi.org/10.3906/elk-1907-84.

[43] A. Hamdan Mohammad, T. Alwada'n, O. Almomani, S. Smadi, N. ElOmari, Bio-inspired Hybrid Feature Selection Model for Intrusion Detection, Comput. Mater. Contin. 73 (2022), 133-150. https://doi.org/10.32604/cmc.2022.027475.

[44] A. Almomani, I. Akour, A. M. Manasrah, O. Almomani, M. Alauthman, et al., Ensemble-based Approach for Efficient Intrusion Detection in Network Traffic, Intell. Autom. Soft Comput. 37 (2023), 2499-2517. https://doi.org/10.32604/iasc.2023.039687.