© 2023 IJNRD | Volume8, issue 11 November 2023 | ISSN: 2456-4184 | IJNRD.ORG

Ensembling Machine Learning-Based Hybrid Feature Vector and Adaptive Genetic Algorithms for Robust Android Malware Detection

[1]Mohammad Huzaif Anwar, [2]Hanumanthakari Sai Sravya, [3]Preet Sureshbhai Sojitra, [4]Shiza Ahmad Khan, [5]Hanfi Aziz
[1,2,3,4,5]UG Student, [1,2,3]Information Science and Engineering, [4,5]Information Technology
[1,2,3]Nitte Meenakshi Institute of Technology, [4,5]Pune Institute of Computer Technology, India

1. ABSTRACT

The proliferation of Android smartphones among consumers has fueled a rapid increase in malware attacks on the Android platform. Cybercriminals leverage technological advancements to actively create and distribute malicious content, posing a significant threat to users. While researchers have made considerable strides in generic Android malware detection, a deeper understanding of specific malware groups is essential for a more nuanced defense strategy. This paper addresses the critical challenge of Android malware detection by presenting a comprehensive solution that amalgamates static analysis of Android APKs with advanced machine learning techniques. Leveraging the recently introduced CICMalDroid 2020 dataset, our approach focuses on robust feature extraction, encompassing API Calls, Intents, Permissions, and Command signatures. To enhance classification accuracy, we employ a two-step methodology. Initially, we perform feature selection using Particle Swarm Optimization (PSO) on the dataset. Subsequently, we optimize the performance of XGBoost and CatBoost machine learning classifiers through an Adaptive Genetic Algorithm (AGA). Our experiments yield exceptional results, achieving a remarkable 99.175% accuracy and a corresponding F-score of 99.054 % with the ensembled XGBoost and CatBoost classifiers. This performance is attributed to PSO-based feature selection and AGA-based hyperparameter optimization. This research underscores the effectiveness of our hybrid approach, which seamlessly integrates static analysis with ensembled machine learning models and adaptive genetic algorithms. The proposed solution significantly enhances classification performance, demonstrating its potential for improved Android malware detection in real-world scenarios. The findings contribute to advancing the field of Android security and serve as a foundation for developing more targeted defenses against evolving malware threats.

***IndexTerms* - Android Malware, Malware Detection, Machine Learning, Static Analysis, Feature Selection, CICMalDroid 2020 Dataset, Feature Extraction. API Calls, Particle Swarm Optimization (PSO), XGBoost, CatBoost.**

2. INTRODUCTION:

The proliferation of Android smartphones, owing to the Android operating system's open-source nature and widespread accessibility, has undeniably transformed the mobile landscape. According to Stat-Counter, Android mobile phones commanded a remarkable 76.67% market share in 2019, marking a substantial growth of 1.1 billion users from 2015 to 2019. However, this widespread adoption has made Android a lucrative target for malicious actors aiming to exploit vulnerabilities and compromise user data.

The advent of Android malware represents a formidable challenge in the realm of cybersecurity. Android, with its ecosystem of millions of applications available through various app stores like Google Play Store and Samsung Store, has become a breeding ground for a diverse array of malware. These malicious programs, ranging from viruses and spyware to keyloggers, have found ingenious ways to infiltrate Android devices, often bypassing traditional signature-based detection methods.

The dynamic nature of Android malware is fueled by its ability to adapt and evolve over time. Through sophisticated techniques like coding obfuscation and encryption, malware creators continuously elevate the complexity of their malicious strains, rendering conventional detection approaches insufficient. Traditional signature-based methods, relying on known patterns and characteristics, struggle to keep pace with the ever-expanding repertoire of Android malware.

Recognizing the inadequacy of traditional detection methods, researchers have turned to machine learning-based strategies. Android malware detection now extends beyond simple signature matching, with a growing emphasis on static and dynamic analysis approaches. Static analysis methods examine factors such as permission usage, function calls, and the presence of specific strings within the Android Application Package (APK). On the other hand, dynamic analysis involves scrutinizing system calls and API calls during the execution of an application.

Despite the advancements in static and dynamic analysis, the Android malware landscape continues to evolve. This prompts researchers to explore innovative approaches, such as feature extraction based on permission combinations. By treating each

permission as a distinct entity and investigating their combinations, researchers aim to enhance the accuracy and robustness of Android malware detection.
This paper delves into this dynamic landscape, proposing a groundbreaking solution that combines ensembling machine learning models with adaptive genetic algorithms for a more resilient Android malware detection system. Through this approach, the research aims to contribute to the ongoing efforts in fortifying the security of Android devices against the ever-growing sophistication of malware attacks.

3. RESEARCH MOTIVATION:

- **Contributions to Android Security:** The overall motivation of the research is to contribute to the field of Android security. The proposed hybrid approach, integrating static analysis, machine learning models, and adaptive genetic algorithms, is positioned as a significant advancement in addressing the challenges posed by Android malware**.**
- **Potential for Real-World Application:** The research suggests that the proposed solution has the potential for real-world application in improving Android malware detection in practical scenarios. The robustness of the approach is demonstrated through experimentation and high-performance metrics.
- **Ineffectiveness of Signature-Based Approaches:** Traditional signature-based detection methods struggle to keep up with the dynamic and evolving nature of Android malware. The limitations of these approaches necessitate the exploration of more advanced techniques.
- **Need for Specific Malware Group Understanding:** While generic Android malware detection methods exist, a deeper understanding of specific malware groups is crucial for a more nuanced defense strategy. Different malware types may require tailored detection approaches.
- **Hybrid Approach with Ensembled Models:** The proposed solution combines machine learning models (XGBoost and CatBoost) through ensemble learning, achieving exceptional accuracy through feature selection (Particle Swarm Optimization) and hyperparameter optimization (Adaptive Genetic Algorithm).
- **CICMalDroid 2020 Dataset:** The research leverages the CICMalDroid 2020 dataset, indicating a commitment to utilizing up-to-date and relevant data for the development and evaluation of the proposed Android malware detection solution.

4. PROJECT OBJECTIVE:

- **Enhancing Detection Accuracy:** The primary goal is to improve the accuracy of Android malware detection beyond traditional methods by incorporating advanced techniques such as machine learning, ensemble learning, and adaptive algorithms.
- **Understanding Specific Malware Groups:** The research aims to deepen the understanding of specific Android malware groups, recognizing that a nuanced defense strategy requires insights into the characteristics and behaviors of distinct types of malware.
- **Utilizing Comprehensive Feature Extraction:** The study focuses on comprehensive feature extraction, encompassing API Calls, Intents, Permissions, and Command signatures, to create a detailed feature vector that captures diverse aspects of Android malware.
- **Optimizing Machine Learning Models:** The objective is to optimize the performance of machine learning classifiers (XGBoost and CatBoost) through feature selection using Particle Swarm Optimization and hyperparameter optimization using an Adaptive Genetic Algorithm.
- **Contributing to Android Security:** The overarching aim is to contribute to the field of Android security by providing a hybrid solution that seamlessly integrates static analysis, machine learning models, and adaptive genetic algorithms. This contribution is expected to enhance the capability of Android devices to defend against evolving and sophisticated malware threats in real-world scenarios.

5. LITERATURE SURVEY:

[1] The paper addresses the escalating threat of malware on the Android Operating System, proposing an innovative malware detection method using machine learning and static analysis of Android APKs. Leveraging the Drebin and Malgenome datasets, the model achieves impressive classification accuracies of 98.19% and 98.84%, respectively, with all four features. Notably, using a twosome permission combination, the model attains 96.27% accuracy with Random Forest for Drebin and 97.63% with SVM and PCA for Malgenome. These findings, presented at the 2022 CSCI conference by P. Raghuvanshi and J. P. Singh, underscore the paper's significant contribution to Android malware detection.

[2] To enhance virus detection on Android IoT devices, a novel framework was introduced by Rajesh Kumar et al., integrating machine learning and blockchain technology. The proposed technique incorporates sequential clustering, classification, and blockchain to bolster its effectiveness. Specifically, machine learning is employed to autonomously identify and eliminate malware data, which is then securely stored in the blockchain. This innovative approach allows the network to disseminate historical malware data from the blockchain, thereby facilitating robust and efficient malware detection.

[3] Machine learning techniques play a crucial role in numerous articles focused on the identification and categorization of malware. Various studies leverage dynamic or static analysis to extract features for effective malware detection. For instance, Nikola Milosevic et al. proposed two static analysis approaches for Android malware [5]. The initial method centers on permissions, while the second method involves parsing the source code with the assistance of machine learning, utilizing a model that encapsulates a

bag of words to characterize the data. The source code-based classification models achieved an impressive F-score of 95.1%, while permission-based classification models demonstrated a commendable F-measure of 89%.

[4] Dynamic analysis methodologies are employed in Android malware detection due to the challenges posed by obfuscated malware and the loading of malicious dynamic content, which may go undetected through static analysis. Numerous research articles, including Mahindru and Sangal (2021a, 2021b), Martín et al. (2018), Abderrahmane et al. (2019), and D'Angelo et al. (2021), have delved into the realm of dynamic analysis-based Android malware detection. Martin A et al. (2018) introduced CANDYMAN, a malware classification tool that utilizes the power of Markov chains to categorize Android malware families. Their approach incorporates deep learning techniques, relying on Markov chains for robust detection. Abderrahamane et al. (2019) leveraged system calls as features in convolutional neural networks to identify fraudulent Android applications. This method relies on pair-level system call dependencies for effective detection. In another dynamic analysis-based Android malware classification system, D'Angelo et al. (2021) utilized the CuckooDroid sandbox (2020), a mobile security framework facilitating static and dynamic feature extraction. Their approach demonstrates the versatility and effectiveness of dynamic analysis in addressing the challenges posed by evolving Android malware threats.

[5] In the referenced study Bayes' algorithm was applied for the analysis of cyber-terrorism on the Internet. The researchers gathered 10,000 data samples from the AndroZoo and Drebin databases. Impressively, the maximum accuracy achieved through this approach reached 91%. However, it is noteworthy that the classification of Android malware using the Bayesian model in this study lacked a comprehensive systematic analysis of classifier performance. Specifically, there was a gap in assessing the significance of characteristics and hyperparameter optimization across various machine learning classifiers to ensure the attainment of the most reliable detection rates. This underscores the need for further exploration and refinement in the methodology to enhance the overall effectiveness of Android malware detection.

[6] In the cited paper the author introduced a lightweight framework for Android malware detection. The methodology involved the utilization of the non-negative matrix factorization method to select vulnerable features, resulting in the extraction of 50 lower-dimensional features from an initial set of 209 features. To facilitate static analysis, mapping techniques were employed to associate each API call with specific features. Subsequently, all these features were aggregated, enabling the counting of the frequency of each feature. This approach showcases a streamlined yet effective strategy for Android malware detection by focusing on a reduced set of crucial features and their frequency in static analysis.

[7] In the referenced paper the author introduced a novel approach termed "artificial malware-based detection" to address the challenges posed by obfuscation techniques employed by attackers. The proposed method involved the extraction of features and the generation of artificial malicious patterns using an evolutionary genetic algorithm. By incorporating both the original features and artificially generated malicious features, the proposed model demonstrated the capability to detect new variants of malware. This innovative method represents a robust strategy for enhancing the detection capabilities of malware, especially in the face of evolving obfuscation techniques utilized by attackers.

[8] In the paper denoted as [12], the author focused on feature weight mapping and joint optimization of selected features, aiming to enhance the performance of the AMD (Android Malware Detection) model. The framework presented in the study involves a calculation of the weight assigned to each feature within an application. Subsequently, a joint parameter optimization method, specifically the differential evolution algorithm, was employed to elevate the overall accuracy of the model. This approach signifies a concerted effort to fine-tune the AMD model by assigning appropriate weights to features and optimizing parameters jointly for improved detection.

[9] The Android Malware Buster (AMB) is a cutting-edge malware detection application designed for Android devices. Developed by Areen Eltaher, Dania Abu-juma'a, Dania Hashem, and Heba Alawneh, AMB employs a machine learning classifier trained on a diverse set of Adware, Scareware, and Ransomware apps. With an impressive 93% accuracy rate, the AMB classifier excels in identifying ongoing malicious behavior through network traffic analysis. Real-time testing further validates AMB's effectiveness, making it a crucial tool to safeguard user privacy amidst the escalating threat of malware on smartphones.

[10] In the realm of cybersecurity, efficient malware identification is crucial for safeguarding system resources and data privacy, especially with the proliferation of Android smartphones facing an increasing array of malware attacks. This paper introduces a novel approach, XAI-AMD-DL (Explainable Artificial Intelligence Android Malware Detection System using Deep Learning), which leverages a hybrid Convolutional Neural Network (CNN) and Bi-Gated Recurrent Unit (Bi-GRU) model. Unlike existing methods that lack interpretability, the proposed model emphasizes Explainable Artificial Intelligence (XAI), allowing a clear understanding of each feature's contribution to the detection system. Evaluation on the CICAndMal2019 android malware dataset demonstrates impressive performance, with the XAI-AMD-DL model achieving 97.98% accuracy, surpassing existing deep learning models in precision, recall, and f1score with values of 97.75%, 97.76%, and 97.75%, respectively. This innovative approach addresses the critical challenge of designing an interpretable and effective Android malware detection system amidst the evolving landscape of AI-enabled malware attacks.

6. PROPOSED METHODOLOGY:

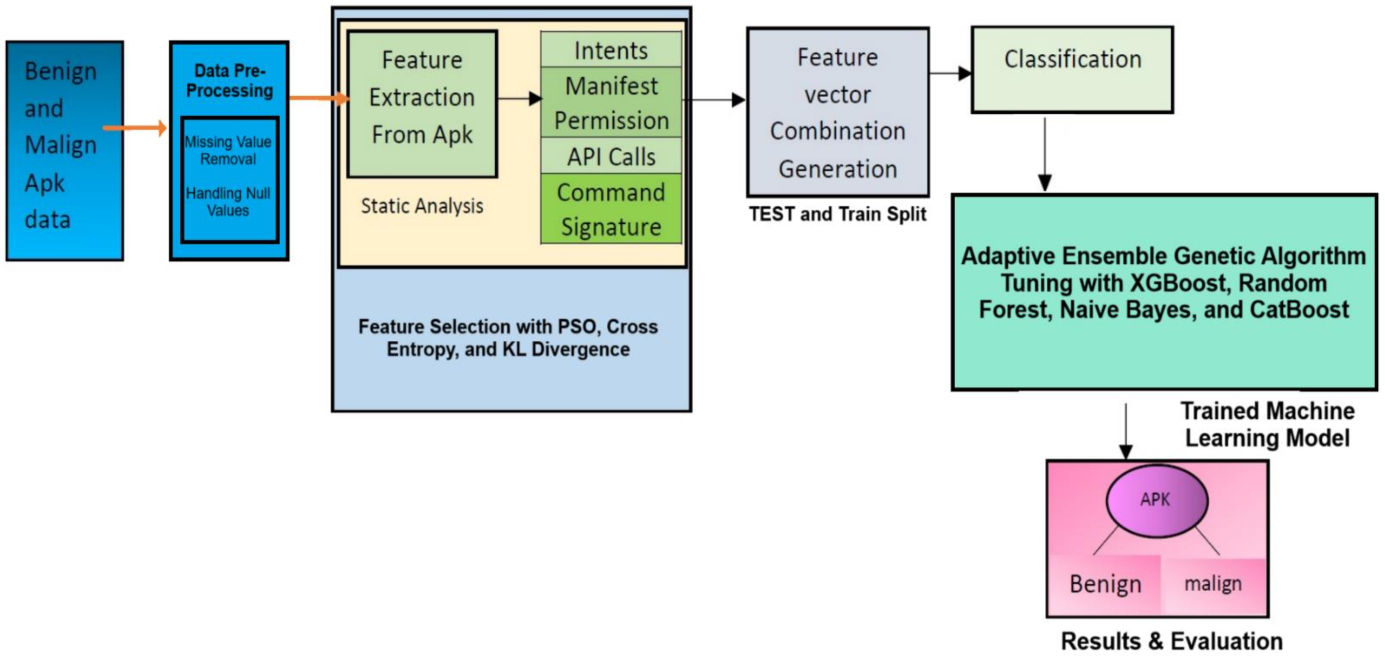
Benign and Malign Apk data
Data Pre-Processing
Missing Value Removal
Handling Null Values
Feature Extraction From Apk
Static Analysis
Intents
Manifest Permission
API Calls
Command Signature
Feature Selection with PSO, Cross Entropy, and KL Divergence
Feature vector Combination Generation
TEST and Train Split
Classification
Adaptive Ensemble Genetic Algorithm Tuning with XGBoost, Random Forest, Naive Bayes, and CatBoost
Trained Machine Learning Model
APK
Benign
malign
Results & Evaluation

Figure 1. Block Diagram of proposed methodology

The proposed methodology for Android malware detection encompasses the collection of a dataset with both benign and malicious applications, followed by meticulous data preprocessing to handle missing and null values. Employing static analysis techniques, relevant features are extracted from APK files, including API Calls, Intents, Permissions, and Command signatures. Feature selection is performed using Particle Swarm Optimization (PSO), Cross Entropy, and KL Divergence to optimize the feature set. The selected features are combined into a feature vector, and a train-test split is executed for model evaluation. Machine learning classifiers, namely XGBoost, Random Forest, CatBoost, and Naive Bayes, undergo hyperparameter tuning via an Adaptive Genetic Algorithm (AGA). The trained models are then ensembled, leveraging their collective strengths. Evaluation on the test dataset yields results in terms of accuracy, precision, recall, and F-score, showcasing the effectiveness of the holistic approach in robust Android malware detection.

6.1 DATA COLLECTION & PREPROCESSING:

In the "Data Collection" phase, our research effort has successfully accumulated an extensive Android dataset, surpassing 17,341 samples sourced from reputable outlets. We take pride in introducing a novel Android Malware dataset, named CICMalDroid 2020, distinguished by four key properties.

In the subsequent "Data Preprocessing" stage, meticulous attention has been dedicated to ensuring the quality and reliability of our dataset. This involves a thorough examination and adept handling of missing values and null values inherent in the collected data. The objective is to establish a pristine dataset that serves as a robust foundation for subsequent analyses, model training, and evaluations. By addressing missing and null values, we aim to enhance the integrity of the dataset, mitigating potential biases and discrepancies. This commitment to data preprocessing underscores our dedication to producing credible and high-quality results in the realm of Android malware detection.

6.2 FEATURE EXTRACTION

In the "Feature Extraction from APK using Static Analysis" phase, our investigation extends to two crucial types of files, namely small files and AndroidManifest.xml, to comprehensively capture the nuances of Android applications. Small files are generated through the decomplication of classes.dex, providing insights into the low-level implementation details of the app. Simultaneously, the AndroidManifest.xml file, in XML format, contains essential properties and metadata defining the application's characteristics. Our approach involves extracting features from both the source code and program levels to ensure a thorough analysis. At the source code level, we employ reverse engineering techniques to obtain the actual source code of Android applications from APK files. This process allows us to delve into the intricacies of the codebase, revealing details such as API calls, function invocations, and structural elements that contribute to the app's behavior. By combining these source code and program-level analyses, our feature extraction process provides a comprehensive representation of the Android app's behavior, encompassing not only high-level functionalities and permissions but also delving into the intricacies of the code implementation. This multifaceted approach enhances the depth and accuracy of our feature set, contributing to a robust foundation for subsequent stages in our Android malware detection methodology.

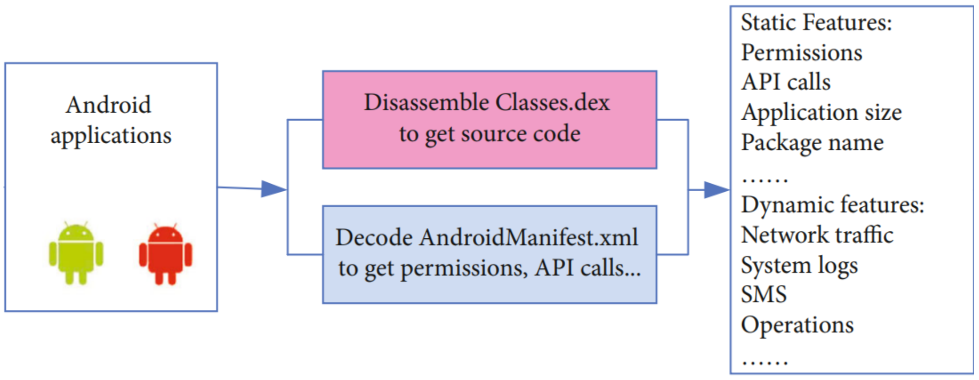

Figure 2- feature extraction block diagram.

6.2 FEATURE SELECTION

In the feature selection phase, our objective is to enhance the efficiency and accuracy of Android malware detection by reducing the dimensionality of the dataset, originally comprising 9504 features. To achieve this, we employ a sophisticated approach involving the Particle Swarm Optimization (PSO) algorithm, Cross Entropy (CE), and Kullback–Leibler (KL) Divergence techniques.

The PSO algorithm iteratively navigates the vast solution space, identifying a subset of features that demonstrates a high correlation with the types of Android applications targeted for classification. This iterative process ensures that the selected features are not only relevant but also contribute significantly to the accurate prediction of app types. The strategic inclusion of CE and KL Divergence further refines the feature selection criteria, considering the dissimilarity between probability distributions and enriching the selection process with additional evaluation metrics.

The combined utilization of these techniques facilitates the removal of redundant and unnecessary features, refining the dataset to include only those attributes critical for robust classification. The PSO algorithm, with its adaptive balance between exploration and exploitation, ensures the identification of an optimal feature subset that is both representative and impactful. This meticulous feature selection methodology is pivotal for improving the accuracy of subsequent machine learning models while expediting the analysis of results in Android malware detection. By focusing on the most informative features and excluding irrelevant ones, the model is equipped to make more efficient and targeted predictions, aligning with the overarching goal of enhancing the efficacy of Android security measures. Below figure shows the extracting and selecting best feature flow diagram.

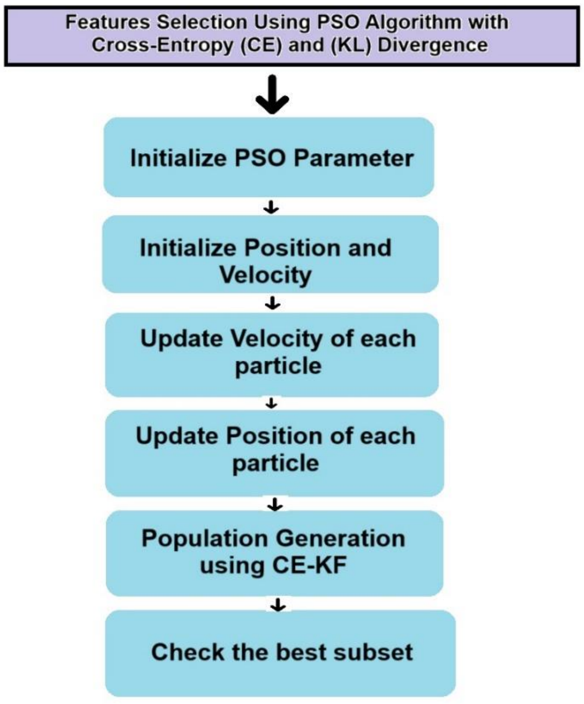
Features Selection Using PSO Algorithm with Cross-Entropy (CE) and (KL) Divergence
Initialize PSO Parameter
Initialize Position and Velocity
Update Velocity of each particle
Update Position of each particle
Population Generation using CE-KF
Check the best subset

Figure 3- Flow diagram of PSO & KL

6.2.1 PARTICLE SWARM OPTIMIZATION (PSO) ALGORITHM

Particle Swarm Optimization (PSO) is an optimization algorithm developed by Kennedy and Eberhart in 1995. It draws inspiration from the collective behavior of flocks of birds and schools of fish. The fundamental concept of PSO involves simulating the collaborative movement of particles in a multidimensional search space with the goal of finding optimal solutions to a given problem. In PSO, a population of potential solutions, represented as particles, traverses the solution space. Each particle adjusts its position and velocity iteratively based on its own experience and the collective knowledge of the swarm. The algorithm is particularly effective for solving optimization problems and has found applications in various fields, including machine learning, data mining, and engineering.

The PSO algorithm begins with a randomly generated population of particles in the solution space. These particles explore the space, and in each iteration, their positions and velocities are updated according to specific formulas. The update is influenced by the particle's historical best position (individual experience) and the best position of any particle in the entire swarm (collective knowledge).
The three key behaviors in PSO, as defined by [13] Del-Valle and co-authors, are separation, alignment, and cohesion. Separation encourages particles to avoid crowded areas in the solution space, alignment guides them to move in the general direction of local mates, and cohesion steers them toward the median position of nearby particles.
The PSO algorithm's mathematical representation includes formulas that govern the update of particle positions and velocities. These formulas incorporate constants and random values, influencing the balance between individual exploration and collective exploitation of the search space.
The formulas for updating the position and velocity vectors in the PSO algorithm are given as follows:

- $v_{id}^{t+1} = v_{id}^{t} + c_1 \cdot \text{rand} \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \text{rand} \cdot (p_{gd}^{t} - x_{id}^{t})$
- $x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}$

where:

- v_{id}^{t} is the velocity vector of particle i at time t,
- x_{id}^{t} is the position vector of particle i at time t,
- p_{id}^{t} is the best position of particle i so far at time t,
- p_{gd}^{t} is the best position in the entire swarm at time t,
- c_1 and c_2 are constants, and
- rand is a random number between 0 and 1.

These formulas govern how the particles' positions and velocities are updated in each iteration of the PSO algorithm.

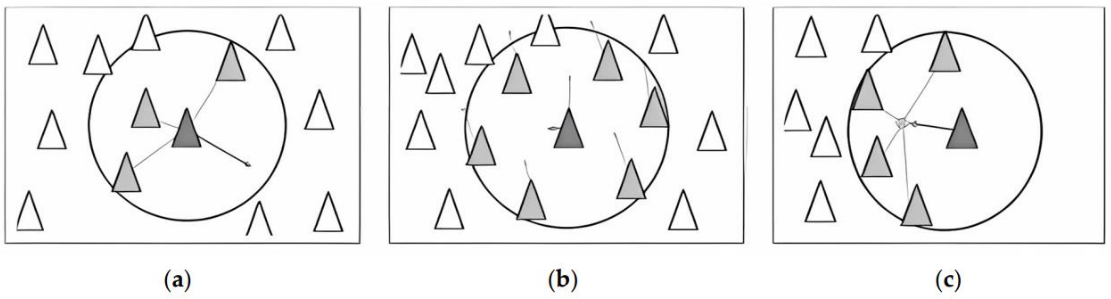

Figure 4- (a) A separation behavior in which particles try to avoid meeting one another. (b) Alignment. behavior, where particles move themselves with the leader of their local flock while keeping their relative velocities constant. (c) Cohesion. behavior, where particles tend to move to where their neighbors in the flock are.

Moreover, the initial phase of the process involves employing the PSO algorithm to establish the population. Subsequently, in the second step, each particle's fitness values are determined, leading to updates in both global and individual bests. Following this, the positions and velocities of the particles are adjusted. This iterative process persists until the termination condition is met, at which point steps two through four are reiterated. The evolution of the PSO algorithm over multiple iterations is depicted in Figure 5. During the first iteration, all particles disperse, emphasizing exploration, and each particle undergoes assessment. Optimal solutions are identified based on the configuration of the swarm members' neighborhood, prompting updates to their individual and collective best particles. The convergence is achieved by orchestrating movements of all particles toward the one with the most favorable solution [14].

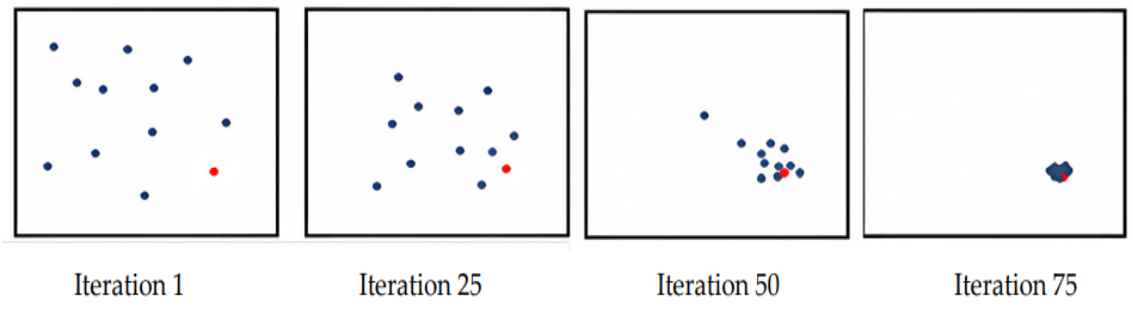

Figure 6- Over iterations, particle swarm optimization tends toward global optimums.

In summary, PSO is a heuristic optimization technique that mimics the social behavior of natural systems. Its ability to efficiently explore complex solution spaces has led to its widespread use in various fields, making it a valuable tool for finding optimal solutions in diverse problem domains.

6.3 FEATURE VECTOR COMBINATION GENERATION & TEST-TRAIN SPLIT

In the context of our research, we employed a feature vector representation to depict mobile applications within the CICMalDroid2020 dataset. This feature vector encapsulates crucial attributes such as API call names and permissions, and each application is defined by a binary feature vector of identical length. A binary value of 1 indicates the presence of a specific feature, while 0 denotes its absence.
Two distinctive feature sets were explored during our experiments:

- Combined Features (225 features): This comprehensive set integrates 113 permissions, 75 API calls, 27 intents, and 10 commands. The binary feature vector assigned to each application captures the presence or absence of these features.
- Permission Feature: In this set, binary combinations of the 113 permissions were generated, resulting in nC2 combinations. The binary feature vector representing each application is derived from these combinations, signaling the presence or absence of specific pairs of permissions.

To create these feature sets, we implemented two algorithms:

Algorithm 1: Feature Vector Generation Algorithm This algorithm delineates the process of constructing the combined feature set. It involves iterating through distinct components such as permissions, API calls, intents, and commands, marking their presence or absence in the feature vector.

Algorithm 1 Feature Vector generation algorithm

```
1: feature_vector = [0,0,0,1,1,..,0]
2: index = 0
3: FOR features in Fs_db # For feature in database
4:         IF features in F_apk THEN
5:                 feature_vector [index]= 1
6:         END IF
7: End FOR
8: return feature_vector
```

Algorithm 2: Feature Vector Permission Combination Generation Algorithm This algorithm elucidates the steps for generating the permission feature set. It likely includes creating binary combinations of the 113 permissions to form pairs (nC2), and each application is represented by a feature vector indicating the presence or absence of these permission pairs.

Algorithm 2 Feature Vector permission combination generation algorithm

```
1: feature_vector = [0,0,0,1,1,..,0]
2: col_check = []
3: index = 0
4: FOR features in Fs_db # For feature in database
5:         features in F_apk THEN
6:          ((col1 != col2) & ((col1 +"_" +col2) not in
    col_check) & ((col2 +"_" +col1) not in col_check)):
7:         Append col1 and col2
8:         output df[col1] * df[col2]
9: End FOR
10: return output
```

Following feature generation, the dataset underwent a crucial feature selection process, reducing the initial 9883 features to a refined set of 423 features. This selection aimed to spotlight characteristics of malware instances by family and category. In conclusion, these methodologies and feature sets form a foundational aspect of our research, enabling the representation of mobile applications in a binary feature space. The subsequent feature selection process on the CICMalDroid2020 dataset distilled these features to 423, elucidating distinctive traits of malware by family and category. Lastly, the data is split into training (60%) and testing (40%) sets for evaluation purposes.

6.3 CLASSIFICATION METHODS

In the pursuit of robust Android malware detection, an ensemble of diverse machine learning algorithms, including Random Forest (RF), Naive Bayes (NB), CatBoost, and XGBoost, is strategically employed. Each algorithm plays a unique role in enhancing the ensemble's ability to discern nuanced patterns indicative of Android malware.

- **Random Forest (RF):** Random Forest, a versatile ensemble learning method, contributes significantly to the ensemble's strength in Android malware detection. By constructing multiple decision trees, each trained on a random subset of the feature vector, RF introduces diversity to capture various facets of malware behavior. Its ensemble voting mechanism consolidates these individual predictions, collectively enhancing the ensemble's understanding. RF's robustness to overfitting and suitability for large datasets make it a cornerstone in the collaborative effort to discern complex patterns indicative of Android malware.

- **Naive Bayes (NB):** In the ensemble, Naive Bayes plays a pivotal role with its probabilistic modeling approach. Modeling the probability distribution of features given the class, NB offers a different lens through which to analyze Android malware characteristics. Its outputs, grounded in probability, are seamlessly integrated into the ensemble through weighted voting.

NB's computational efficiency and robustness in the presence of irrelevant features make it an effective component, providing a probabilistic perspective on uncertain relationships within Android malware detection.

- **CatBoost:** CatBoost's specialized strength lies in efficiently handling categorical features, making it a valuable asset in Android malware detection. Its adeptness in decoding categorical attributes adds a layer of understanding to the ensemble's feature representation. As an integral part of the collaborative effort, CatBoost enhances the ensemble's ability to discern malware patterns by addressing the challenges posed by categorical features. Its built-in regularization further ensures the ensemble's resilience to overfitting, contributing to a more robust detection system.

- **XGBoost:** XGBoost, an optimized gradient boosting algorithm, is a cornerstone in the ensemble's pursuit of high predictive power and efficiency. With boosting techniques and regularization, XGBoost sequentially refines the model, making it a valuable contributor to the ensemble. Outputs from XGBoost are seamlessly integrated through model stacking, enhancing the overall decision-making process. Its ability to work efficiently and quickly, especially with large datasets, positions XGBoost as a key player in Android malware detection, optimizing the ensemble for structured tabular data scenarios.

6.4 HYPERPARAMETERS TUNING USING ADAPTIVE GENETIC ALGORITHM (AGA)

The adaptive genetic algorithm (AGA) is a powerful optimization tool inspired by natural selection principles to address complex problems [15]. Introduced by Holland in 1975, the GA mimics biological evolution through crossover and mutation processes, creating optimized solutions. In nature, genes partially crossover and mutate, leading to the development of new genes that differ from existing ones, adapting to the environment. GAs excel in selecting optimal combinations and exploring vast search spaces, making them suitable for solving NP-hard problems like feature selection [16].

A standard GA commences with randomly generated gene populations, evaluated for fitness according to D. Whitley's approach. Genes meeting specific criteria are given more chances for reproduction, and a selection process identifies those meeting requirements. These selected genes are then recombined in various ways to generate new genes. Crossovers and mutations, guided by specific probabilities, can create novel genes. This iterative process continues until the termination criteria are met, as illustrated in Figure 7 [17].

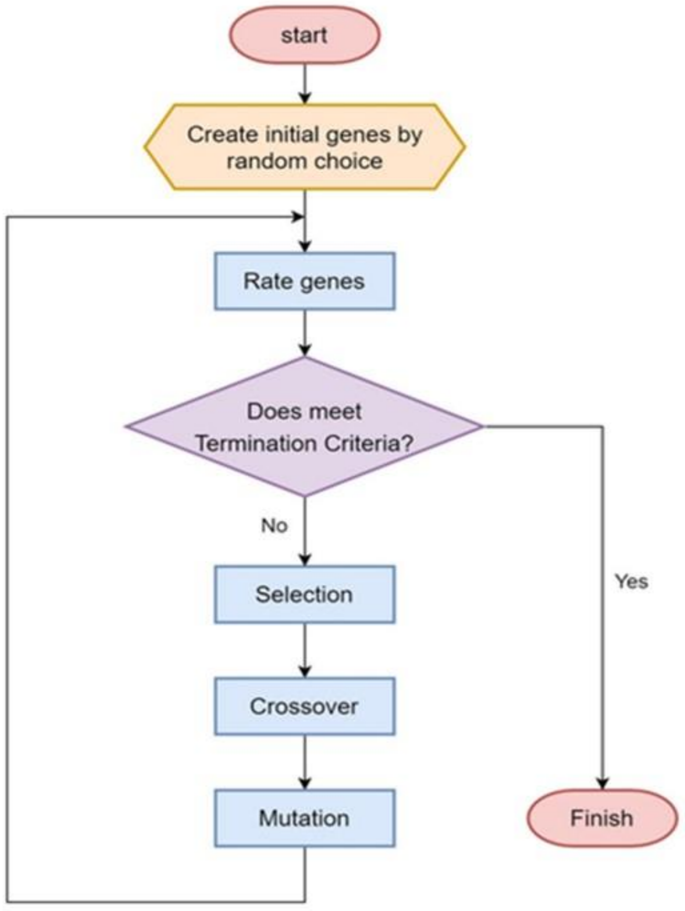
start
Create initial genes by random choice
Rate genes
Does meet Termination Criteria?
No
Yes
Selection
Crossover
Mutation
Finish

Figure 7- A schematic diagram to present the pattern of how the “genetic algorithm” progresses

The Adaptive Genetic Algorithm (AGA) enhances the effectiveness by dynamically adjusting the crossover and mutation rates based on fitness. The AGA employs coding, fitness computation, selection, reproduction, crossover, mutation, and decoding processes. The adaptive crossover and mutation probability in AGA dynamically adjusts between the population's average fitness and the highest fitness level at each individual's fitness level [18]. This adjustment is expressed in the below equations.

$$P_c = \begin{cases} k_1 = \frac{(f_{\max} - f)}{f_{\max} - f_{\text{avg}}} & \text{if } f \geq f_{\text{avg}} \\ k_3 & \text{if } f < f_{\text{avg}} \end{cases}$$

Here, P_c represents the crossover probability, which determines the likelihood of crossover occurring in the genetic algorithm. The calculation of P_c is conditionally defined based on the fitness (f) of an individual in the population compared to the average fitness (f_{avg}) and the maximum fitness (f_{max}).

- If the fitness (f) of an individual is greater than or equal to the average fitness (f_{avg}), then the crossover probability (P_c) is dynamically adjusted using the formula $k_1 = (f_{max}-f)/ (f_{max}-f_{avg})$.
- If the fitness (f) is less than the average fitness (f_{avg}), the crossover probability P_c) is set to a predefined value k_3.

$$P_m = \begin{cases} k_2 = \frac{(f_{\max} - f)}{f_{\max} - f_{\text{avg}}} & \text{if } f \geq f_{\text{avg}} \\ k_4 & \text{if } f < f_{\text{avg}} \end{cases}$$

Here, P_m represents the mutation probability, determining the likelihood of mutation occurring in the genetic algorithm. The calculation of P_m is conditionally defined based on the fitness (f) of an individual in the population compared to the average fitness (f_{avg}) and the maximum fitness (f_{max}).

- If the fitness (f) of an individual is greater than or equal to the average fitness (f_{avg}), then the mutation (P_m) is dynamically adjusted using the formula $k_1 = (f_{max}-f)/ (f_{max}-f_{avg})$.
- If the fitness (f) is less than the average fitness (f_{avg}), the mutation probability P_m) is set to a predefined value k_3.

These equations showcase the adaptive nature of the AGA, where the probabilities of crossover and mutation dynamically change based on the individual's fitness, promoting more optimal results in the evolutionary process. The constants $K_{1,}$ $K_{2,}$ $K_{3,}$ K_4 parameters that determine the adaptation speed and influence the adjustment of crossover and mutation probabilities.

In our proposed method for improving algorithm results, the Adaptive Genetic Algorithm fine-tunes hyperparameters using the AdaptiveGeneticSelectionCV method for machine learning classifiers (RF, NB, CatBoost and XGBoost). The method's parameters are carefully selected, ensuring optimal results, as demonstrated in the implementation and performance measurement of the classifiers. While various versions of AGAs aim to enhance techniques, the AGA's adaptive nature, weighted by fitness, enhances its ability to discover optimal solutions. By dynamically adjusting crossover and mutation probabilities, AGAs seek broader applicability, offering a more adaptive approach to problem-solving. The iterative AGA-based procedures, driven by fitness values, contribute to obtaining refined solutions.

6.4 RESULTS AND EVALUATION

A. Experiment Environment

All the models in the experiment were trained using TensorFlow, a powerful Python-based deep learning toolkit. TensorFlow is known for its capability to perform efficient mathematical operations on GPUs, which significantly enhances computational performance. In this particular project, the training simulations were executed on a system equipped with a Core i7 CPU, an NVIDIA RTX3060 graphics card, and 128 GB of RAM.The utilization of a GPU in the training process played a crucial role in accelerating the computations. This hardware acceleration, especially with the NVIDIA RTX3060 graphics card, resulted in a tenfold reduction in training time compared to using only the CPU. While the focus of the project may not have been primarily on speed considerations, the use of a GPU undoubtedly contributed to the efficiency and speed of the model training process.

B. Evaluation Metrics

Performance metrics are indispensable in assessing the success and efficiency of systems. In the realm of Android malware detection, these metrics play a crucial role in monitoring progress, identifying areas for improvement, and aiding decision-making in the ongoing pursuit of effective intrusion detection and prevention. The study emphasizes the significance of applying performance metrics to gauge the fulfillment and effectiveness of the machine learning-based approach.
Several standard performance metrics are employed to evaluate the classification models utilized in the study:

1. **Accuracy:** Measures the proportion of correctly classified models among all predictions, providing an overall assessment of model correctness. The accuracy (ACC) is calculated as the total number of two correct predictions divided by the total number of a dataset.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision (Positive Predictive Value):** Evaluates the accuracy of positive predictions, answering the question of how many instances predicted as positive were correct. Particularly relevant when the cost of false positives is high. The precision (PRE) is the ability of a model to find all the predicted positive cases are correctly classified.

$$PRE = \frac{TP}{TP + FP}$$

3. **Recall (Sensitivity or True Positive Rate):** Gauges the model's ability to capture all relevant instances of the positive class by assessing how many actual positive instances were correctly predicted. The recall or true positive rate (TPR) is the proportion of the correctly identified positive cases.

$$TPR = \frac{TP}{TP + FN}$$

4. **F1 Score:** A comprehensive metric that combines precision and recall, balancing the trade-off between the two. Particularly useful when dealing with an uneven class distribution. The F1-score is a measure of positive class, which combines the precision and recall of a classifier into a single metric by taking their harmonic mean.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

In summary, accuracy offers a comprehensive perspective on a model's correctness, whereas precision and recall concentrate on distinct aspects related to positive predictions. The F1 Score, which integrates these metrics, provides a well-rounded evaluation, particularly beneficial in scenarios marked by an imbalance between positive and negative classes. Each metric serves a distinctive role in evaluating various facets of a classification model's performance.

C. Experimental Results on CICMalDroid2020 Dataset

Introducing CICMalDroid 2020, a cutting-edge Android Malware dataset designed with four distinctive attributes. Boasting over 17,341 samples, this dataset is both extensive and recent, encapsulating sophisticated Android malware scenarios up to 2018. Noteworthy for its diversity, CICMalDroid 2020 spans five categories: Adware, Banking malware, SMS malware, Riskware, and Benign, providing a comprehensive representation of Android threats. The dataset's strength lies in its comprehensive feature set, capturing the most extensive static and dynamic features compared to publicly available datasets. Data collection sourced samples from reputable entities like VirusTotal, Contagio, AMD, and MalDozer, ensuring credibility. Employing CopperDroid for dynamic analysis, we successfully ran 13,077 samples, revealing valuable insights into runtime behaviors. The resultant dataset, meticulously categorized with 11,598 samples after balancing, sets the stage for robust AI-based analysis, offering a nuanced understanding of Android malware for cybersecurity research.

Figure 8 illustrates the distribution of benign and malware classes, offering a visual representation of the dataset's composition. The chart provides insights into the relative proportions of benign and malicious samples within the dataset.

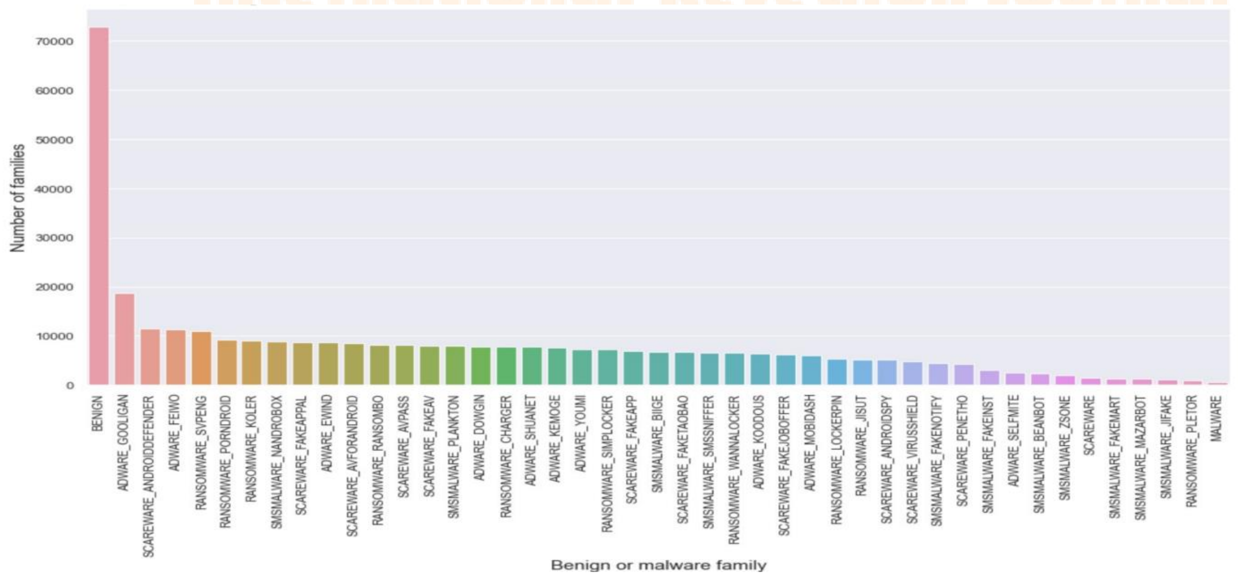

Figure 8-Distribution of benign and malware classes

TABLE I: Attribution for CICMalDroid2020 dataset

No.	Category of Samples	Of Samples
1	**Adware**	**1514**
2	**Banking**	**2777**
3	**Benign**	**4033**

4	**Riskware**	**3890**
5	**SMS**	**4802**

Figure 9 shows Gini Index for CICMalDroid2020 serves as a quantitative measure, assessing the dataset's diversity and distribution of malware classes. This index offers insights into the balance and variability within the dataset, aiding in the evaluation of its representativeness for robust analysis.

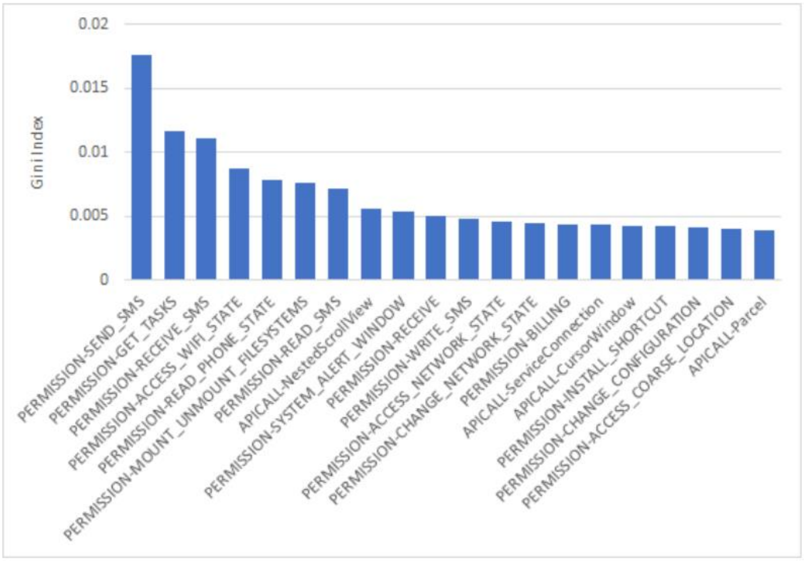

Fig. 9: CICMalDroid2020 Gini Index

In the comprehensive training and validation of our model encompassing all classes within the dataset, we conducted an extensive analysis over 150 epochs. The rigorous evaluation resulted in an impressive accuracy rate of 97.62%. This outcome signifies the model's robust learning and discriminative capabilities across diverse classes, highlighting its efficacy in accurately classifying instances within the dataset. The extended training duration allowed for a thorough exploration of the model's capacity to generalize and make accurate predictions across a wide array of scenarios, contributing to its overall reliability in handling various classes present in the dataset.

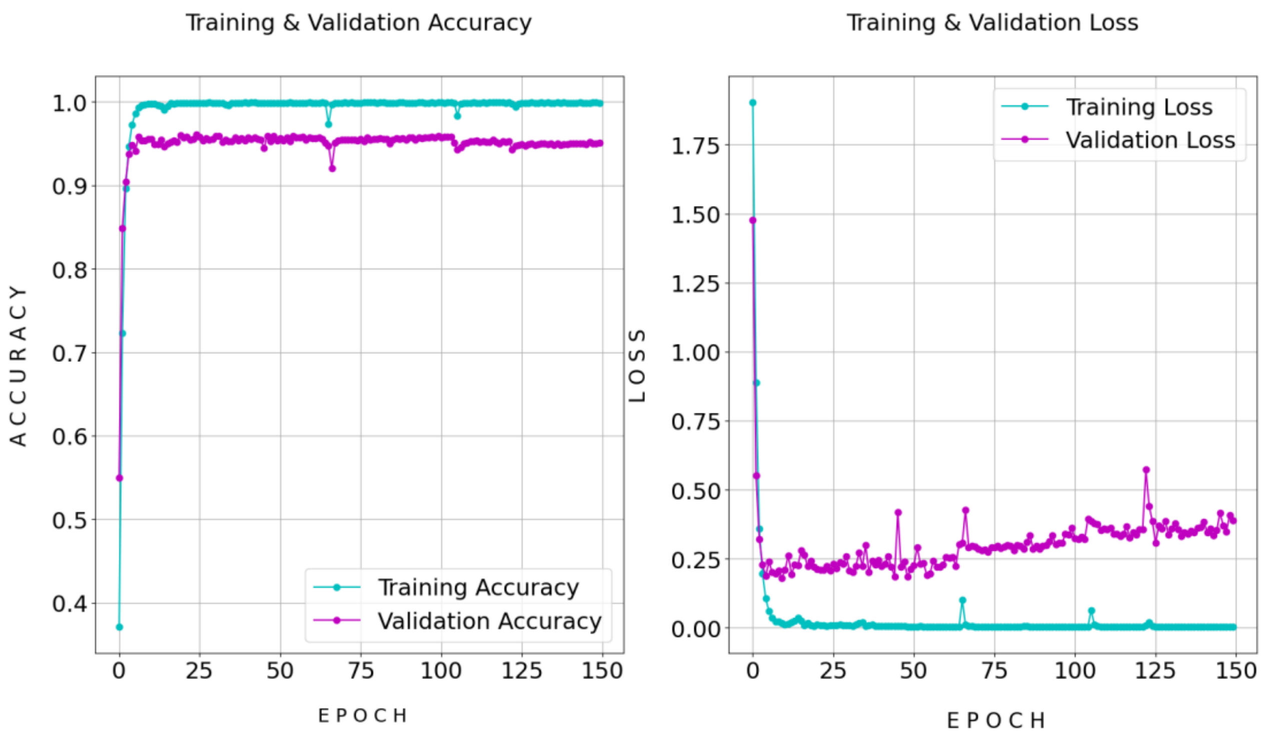

Figure 10-CICMalDroid2020 training and validation result.

In the training and validation process utilizing Adaptive Genetic Algorithm (ADA) on the dataset, our model demonstrated exceptional performance over 100 epochs, achieving an impressive accuracy rate of 98.97%. This outcome underscores the efficacy of incorporating adaptive genetic algorithms in the model's optimization, leading to heightened precision and robustness in handling the dataset. The utilization of ADA for feature selection and hyperparameter tuning has evidently contributed to the model's enhanced accuracy, showcasing its effectiveness in addressing the complexities present in the dataset and improving overall predictive performance.

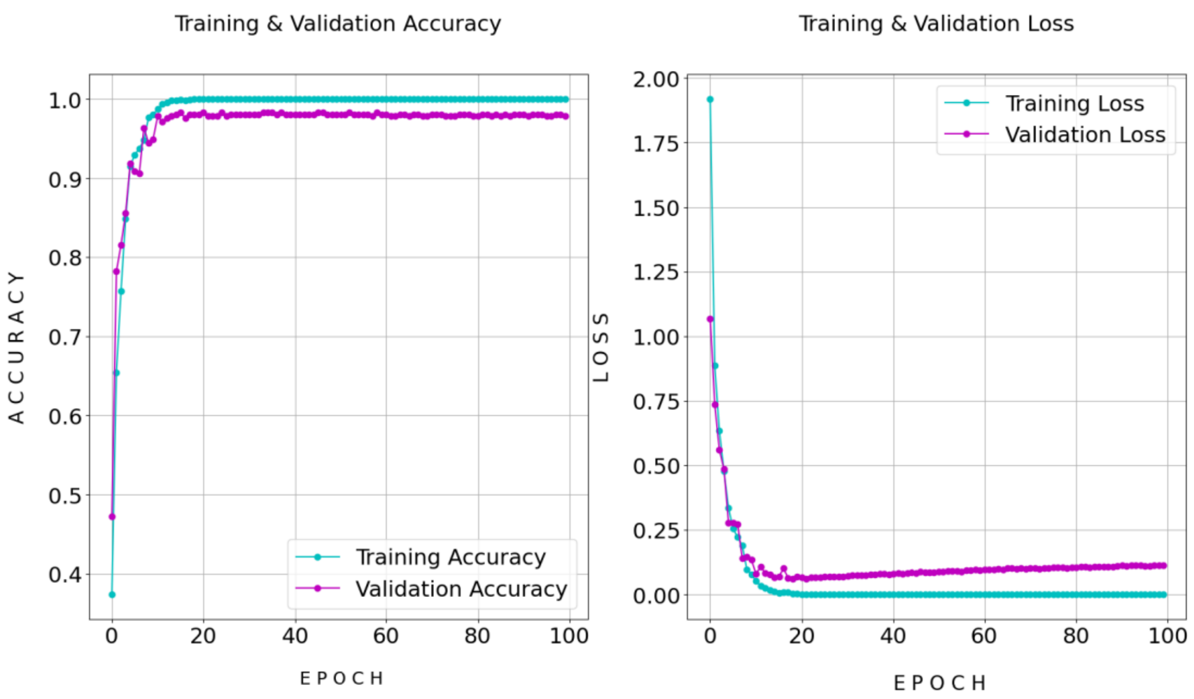

Figure 11-CICMalDroid2020 training and validation result using ADA.

D. Final Results

In our study, we achieved a significant reduction in the number of features, downsizing from 9883 to 423. The experimental outcomes demonstrate that all classifiers exhibit robust performance in accurately classifying malware. Particularly noteworthy is the hybrid feature set, incorporating PSO and KL divergence, coupled with hyperparameter tuning using ADA, which not only attains the highest accuracy but also yields an impressive F1-Score of 99.054. The detailed results showcase exceptional precision at 99.14, recall at 98.979, and an overall accuracy of 99.175. These findings underscore the efficacy of our proposed model in achieving both high accuracy and precision in the classification of malware.

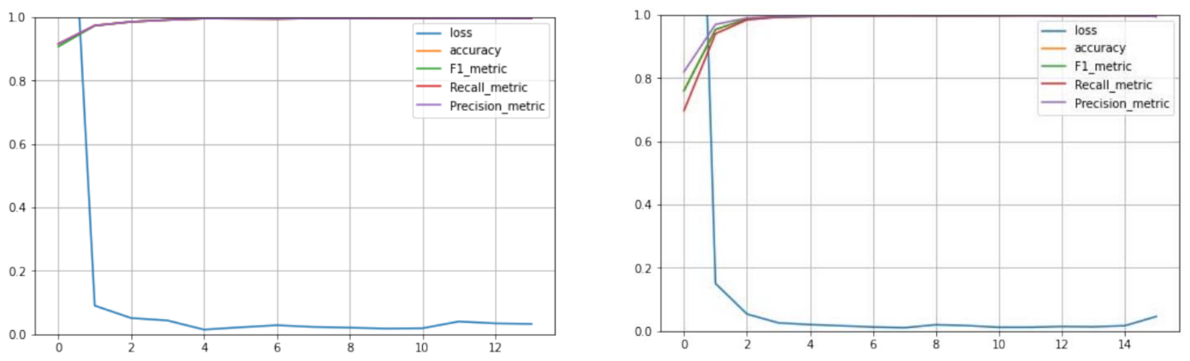

Figure 12 - Learning curves: the mean training loss and evaluation metrics measured over each epoch with (a)two and (b)five target classes.

The confusion matrix is a powerful tool for evaluating the performance of our model, particularly when dealing with two target classes and an expanded scenario with five target classes. In the binary classification setting, the matrix provides insights into true positives, true negatives, false positives, and false negatives, enabling a comprehensive assessment of the model's precision, recall, and accuracy. As we extend our analysis to five target classes, the confusion matrix becomes instrumental in understanding the model's ability to discern between diverse categories, shedding light on potential areas for improvement. By scrutinizing these matrices, we gain a nuanced understanding of our model's performance, which is crucial for refining and optimizing its classification capabilities.

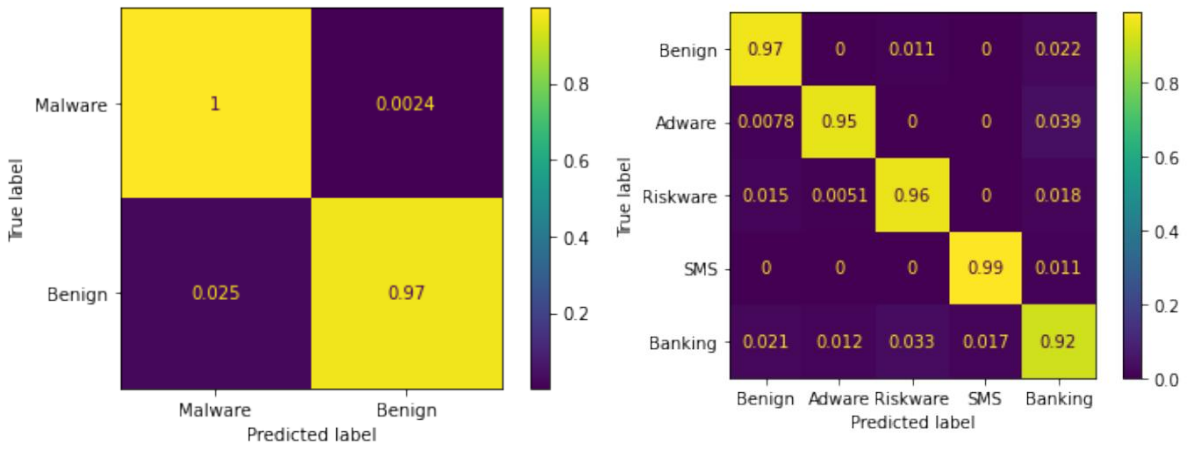

Figure 13- Confusion matrix of our model with (a)two and (b)five target classes

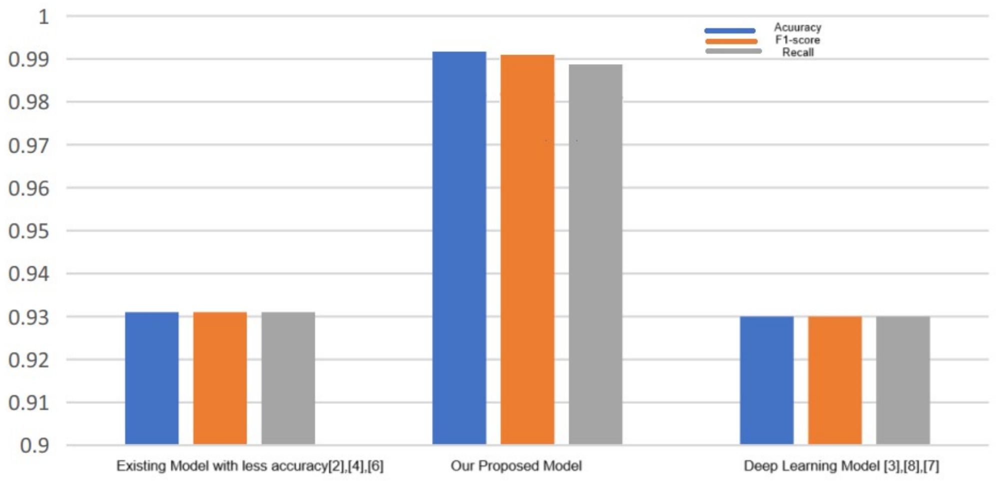

Figure 14 Accuracy, F1-score, Recall compared with other models.

7. CONCLUSION

In this study, we addressed the critical challenge of Android malware detection by presenting a comprehensive solution that integrates static analysis of Android APKs with advanced machine learning techniques. Leveraging the CICMalDroid 2020 dataset, our approach focuses on robust feature extraction, including API Calls, Intents, Permissions, and Command signatures. Through a two-step methodology involving feature selection using Particle Swarm Optimization (PSO) and hyperparameter optimization with an Adaptive Genetic Algorithm (AGA), we achieved exceptional results. Our experiments demonstrated a significant reduction in feature dimensionality, from 9883 to 423, showcasing the effectiveness of the proposed feature selection methods. The ensemble of machine learning classifiers, particularly XGBoost and CatBoost, exhibited robust performance, with an outstanding accuracy of 99.175% and an impressive F1-Score of 99.054. The hybrid feature set, incorporating PSO, KL divergence, and ADA-based hyperparameter tuning, emerged as a key contributor to this success. Noteworthy precision at 99.14 and recall at 98.979 underscored a harmonious balance in the model's ability to accurately identify positive instances while capturing the majority of actual positive cases. The achieved F1-Score of 99.054 emphasized a balanced trade-off between precision and recall, crucial for real-world scenarios where both false positives and false negatives carry significant consequences. The consistent accuracy across classifiers and hybrid feature sets, combined with the model's general applicability, positions our solution as a promising candidate for real-world deployment. In conclusion, our hybrid approach, seamlessly integrating static analysis with ensembled machine learning models and adaptive genetic algorithms, significantly enhances the capability of Android devices to defend against evolving and sophisticated malware threats. This research contributes to the advancement of Android security, providing a foundation for targeted defenses and proactive measures in the dynamic landscape of Android malware.

8. REFERENCES

[1] P. Raghuvanshi and J. P. Singh, "Android Malware Detection Using Machine Learning Techniques," 2022 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2022, pp. 1117-1121, doi: 10.1109/CSCI58124.2022.00200.

[2] Kumar, R.; Zhang, X.; Wang, W.Y.; Khan, R.U.; Kumar, J.; Sharif, A. A Multimodal Malware Detection Technique for Android IoT Devices Using Various Features. IEEE Access 2019, 7, 64411–64430.

[3] Riggs, C.; Rigaud, C.-E.; Beard, R.; Douglas, T.; Elish, K. A Survey on Connected Vehicles Vulnerabilities and Countermeasures. J. Traffic Logist. Eng. 2018, 6, 11–16.

[4] Ambarwari A, Adrian QJ, Herdiyeni Y (2020) Analysis of the efect of data scaling on the performance of the machine learning algorithm for plant identifcation. J Resti Rekayasa Sist Dan Teknol Inf 4:117–122 Atzeni A, Diaz F, Marcelli A, Sánchez A, Squillero G, Tonda A (2018) Countering android malware: a scalable semi-supervised approach for familysignature generation. IEEE Access. https://doi.org/10.1109/ACCESS.2018. 2874502

[5] Tchakount F., Dayang P. System calls analysis of malwares on android International Journal of Science and Tecnology (IJST), 2 (9) (2013).

[6] A. Roy, D. Singh, A. Roy, D. Singh, G. Jaggi, and K. Sharma, "ScienceDirect Android Malware Detection based on Vulnerable Feature Aggregation Android Malware Detection based on Vulnerable Feature," Procedia Comput. Sci., vol. 173, no. 2019, pp. 345–353, 2020, doi: 10.1016/j.procs.2020.06.040.

[7] M. Jerbi, Z. C. Dagdia, S. Bechikh, and M. Makhlouf, "On the Use of Artificial Malicious Patterns for Android Malware Detection," pp. 1–43, 2020

[8] L. Cai, Y. Li, and Z. Xiong, "JOWMDroid : Android Malware Detection Based on Feature Weighting with Joint Optimization of Weight-Mapping and Classifier Parameters," Comput. Secur., p. 102086, 2020, doi: 10.1016/j.cose.2020.102086

[9] A. Eltaher, D. Abu-juma'a, D. Hashem and H. Alawneh, "Design and Implementation of a Malware Detection Tool Using Network Traffic Analysis in Android-based Devices," 2023 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 2023, pp. 276-280, doi: 10.1109/JEEIT58638.2023.10185826.

[10] S. K. Smmarwar, G. P. Gupta and S. Kumar, "XAI-AMD-DL: An Explainable AI Approach for Android Malware Detection System Using Deep Learning," 2023 IEEE World Conference on Applied Intelligence and Computing (AIC), Sonbhadra, India, 2023, pp. 423-428, doi: 10.1109/AIC57670.2023.10263974.

[11] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, Ali A. Ghorbani; **Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning**, The 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC), Aug. 17-24, 2020.

[12] Samaneh Mahdavifar, Dima Alhadidi, and Ali A. Ghorbani (2022). **Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder**, Journal of Network and Systems Management 30 (1), 1-34

[13] Singh, V. Sunflower leaf diseases detection using image segmentation based on particle swarm optimization. Artif. Intell. Agric. 2019, 3, 62–68.

[14] Ab Wahab, M.N.; Nefti-Meziani, S.; Atyabi, A. A Comprehensive Review of Swarm Optimization Algorithms. PLoS ONE 2015, 10, e0122827

[15] Do ˘gru, ˙I.A. Mobile Security Laboratory; Department of Computer Engineering, Gazi University—Faculty of Technology: Ankara, Turkey, 2017. Available online: https://mobseclab.gazi.edu.tr

[16] Muniyappan, S.; Rajendran, P. Contrast Enhancement of Medical Images through Adaptive Genetic Algorithm (AGA) over Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). Multimedia Tools Appl. 2019, 78, 6487–6511.

[17] Chen, T.; Guestrin, C. XGBoost: A scalable tree boosting system. In Proceedings of the KDD'16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.

[18] Muniyappan, S.; Rajendran, P. Contrast Enhancement of Medical Images through Adaptive Genetic Algorithm (AGA) over Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). Multimedia Tools Appl. 2019, 78, 6487–6511.