Integrating Built-in Feature Importance and Mutual Information for Efficient Android Malware Classification Model

Lukmanul Hakim[1], Wildanil Ghozi[2], Fauzi Adi Rafrastara[3]

[1,2,3]*Informatics Engineering Department, Universitas Dian Nuswantoro, Semarang, Indonesia*

[1]111202214197@mhs.dinus.ac.id

[2] wildanil.ghozi@dsn.dinus.ac.id (*)

[3] fauziadi@dsn.dinus.ac.id

Received: 2025-11-19; Accepted: 2026-01-15; Published: 2026-01-29

Abstract— The rapid growth of the Android operating system in the global market demands efficient and accurate malware detection solutions. This study proposes an Android malware classification approach based on machine learning with a focus on feature selection optimization to achieve an optimal balance between performance and computational efficiency. Using the CICMalDroid2020 dataset, consisting of 11598 samples and 470 dynamic features, this study evaluates four machine learning algorithms (LightGBM, XGBoost, Random Forest, and K-Nearest Neighbours) combined with two feature selection methods: Mutual Information (MI) and Embedded Feature Importance (FI). Experiments were conducted with automatic feature selection over 50-475 features to identify the optimal configuration for each model. The research results show that LightGBM with Feature Importance achieves the best performance, with an accuracy of 96.49%, an F1-score of 95.74%, using only 270 features (a 42.6% reduction), and the fastest test time of 0.036 seconds. XGBoost FI achieves 96.31% accuracy with 225 features (52.1% reduction), Random Forest MI achieves 95.62% with 240 features, while KNN MI achieves 91.37% with 135 features. Feature overlap analysis reveals that the 135 core features selected by KNN MI are a subset of features from other models, with dominant categories including system calls (40%), Android API (25%), network operations (15%), file system patterns (10%), and behavioural patterns (10%). This research shows that the Feature Importance method from tree-based algorithms outperforms Mutual Information by 5-6% in capturing non-linear dependencies and complex interactions in malware behaviour. Feature Importance can detect contextual patterns, such as the combination of getDeviceId and NETWORK_ACCESS, that are only dangerous when occurring simultaneously, which are more easily detected by tree-based methods. The optimal range of 225-270 features provides a sweet spot between comprehensiveness and efficiency; XGBoost with 225 features is only 0.18% below LightGBM but 16.7% more computationally efficient, making it ideal for real-time scanning. The main contribution of this research is the development of a light-weight yet reliable model without destructive sampling techniques, providing a practical solution for real-time malware detection on Android devices with limited resources. This approach successfully reduces dimensions by up to 52% while maintaining, or even improving, performance, making significant contributions to the development of efficient, accurate, and applicable Android malware detection techniques for real-time security systems. For further development, exploring LightGBM-XGBoost ensembles could increase accuracy beyond 97%, along with advanced feature engineering and periodic evaluation of the latest malware variants.

Keywords— Malware Classification Model; Machine Learning; Feature Selection; Feature Importance; Data Security; Android.

I. INTRODUCTION

Malicious software (malware) refers to harmful programs designed to damage computers or networks and steal sensitive information, encrypt or delete critical data, and disrupt important services without user consent, thereby causing significant losses [1-4], making it crucial to protect digital assets. The global malware landscape includes various types, such as viruses, worms, Trojans, ransomware, and spyware, as well as increasingly sophisticated variants, such as polymorphic and metamorphic malware, which can change their code to avoid detection [5-8]. Modern malware targets individuals and governments alike, making cybersecurity a global challenge [9]. Traditional malware detection approaches, primarily based on signature-based techniques and heuristic analysis, have proven inadequate for dealing with modern malware threats, especially in mobile environments [4][7]. The use of machine learning offers an innovative approach to detecting malware, with more reliable capabilities in identifying types of malware that are difficult to handle with traditional methods [10].

Alongside the rise of malware, Android-based devices dominate the market and have become the backbone of the smartphone world, controlling 75.5% of the global market and reaching approximately 4.5 billion users by 2024 [11]. This makes it an easy target for cyber attacks [12-14]. Android anti-malware technology capable of quickly detecting and classifying malware to plan emergency responses has become popular in recent years. Several studies have demonstrated the potential of deep learning, but also highlighted significant obstacles. Although deep learning can reduce the need for manual feature engineering, designing an optimal neural network architecture still requires deep expertise in the field [15]. The computational challenges are also no less important, as deep learning will require significant resources.

Meanwhile, Bensaoud et al. [14] emphasise that malware detection is a complex task and that there is no truly perfect approach; even evaluation standards are difficult to determine. Research [16] highlights that deep learning requires large datasets to achieve high performance, which are often unavailable in cybersecurity contexts. Thus, data scarcity and high computation are the main obstacles to deep learning.

DOI : https://doi.org/10.25139/inform.v11i1.11220

Detection and classification of Android malware have become active areas of research as mobile security threats increase, with various machine learning and deep learning approaches proposed to address this challenge [17]. The CICMalDroid2020 dataset, developed by Mahdavifar et al. [18] and serving as the primary reference, contains 17341 Android application samples across five categories: Adware, Banking Malware, SMS Malware, Riskware, and Benign. Research [18] proposes a semi-supervised deep learning framework based on pseudo-labelling, using a seven-layer Deep Neural Network (DNN) architecture trained simultaneously on a combination of labelled and unlabeled data to overcome the limitations of labelled data. However, this research has limitations, including parameter tuning complexity, long training time, dependence on the quality of initial pseudo-labels, limited exploration of feature selection, and the use of all 470 features without reduction, which can lead to the curse of dimensionality and slow down the model.

Study [18] proposes a hybrid method based on the Pseudo-Label Stacked Auto-Encoder (PLSAE), which achieved an accuracy of 98.28%. This semi-supervised approach is effective when labelled data is limited. Still, it is less efficient than models such as XGBoost and LightGBM, which are faster to train, more transparent in terms of feature importance, and capable of achieving similar performance with fewer resources. With the support of modern optimizations such as GPU acceleration, XGBoost and LightGBM are superior for real-time and large-scale applications. At the same time, PLSAE still faces challenges in computational efficiency and interpretability.

Research conducted by Villarroel and Gutiérrez-Cárdenas [19] used frequency-based dynamic analysis of system calls to detect and classify malware by comparing XGBoost, LightGBM, and Random Forest on the CICMalDroid2020 dataset. They applied extensive pre-processing, including outlier removal using the PyOD library with the K-Nearest Neighbours algorithm, undersampling to address class imbalance, and feature selection using Random Forest with a threshold of 0.004, reducing 470 features to 89. The results of the experiment showed that LightGBM achieved the best performance, followed by XGBoost and Random Forest. Sampling reduced the data from 11598 to 4085 records, potentially eliminating important information and leading to underfitting, thereby reducing the model's ability to generalise well [20. The feature selection method relies solely on a single algorithm, without exploring alternatives or systematically comparing methods.

The research [21] also conducted a comparative analysis of 13 machine learning algorithms on the CICMalDroid2020 dataset, with LightGBM achieving the highest F1-score of 94.72% (baseline) and 94.81% after SMOTE. PCA lowered the performance to 92.76%, but fine-tuning the hyperparameters achieved an optimal accuracy of 95.49%. Limitations include the lack of exploration of alternative sampling techniques, the use of feature selection methods other than PCA (e.g., feature importance or mutual information), and the use of all 470 features without accounting for redundancy.

This study addresses the persistent trade-off between accuracy and computational efficiency in Android malware detection. While deep learning models can achieve high accuracy (e.g., >98%), their reliance on large datasets, high-end hardware, and long training times limits their deployment on resource-constrained mobile devices. Moreover, most existing machine learning approaches use a uniform feature selection strategy across classifiers, thereby ignoring the intrinsic characteristics of each algorithm. This often leads to suboptimal performance or unnecessary computational overhead. In response, we propose a hybrid feature selection framework that combines Mutual Information (MI) and embedded Feature Importance (FI) to adaptively select feature subsets tailored to specific models, namely LightGBM, XGBoost, Random Forest, and KNN, without resorting to destructive data sampling that may eliminate critical information.

II. Research Methodology

This chapter describes the methods used in the research presented in Fig.1, including the research design, data collection techniques, and analysis procedures used to achieve the research objectives.

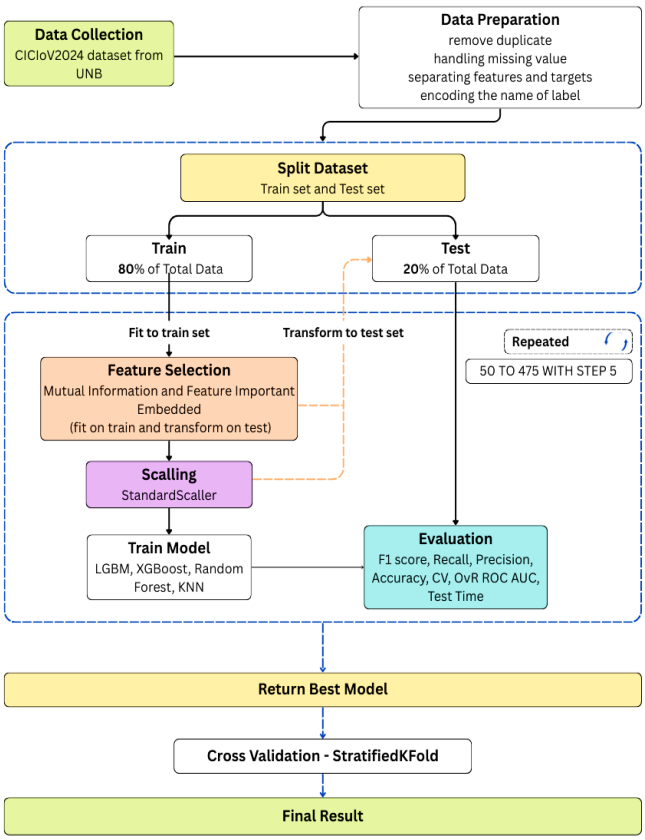

Fig.1. Research Process

A. *Preparation*

Model training is a very important stage and becomes complex when applied to large datasets. In the context of machine learning model training, especially on large datasets,

DOI : https://doi.org/10.25139/inform.v11i1.11220

hardware and software specifications play a significant role. The right combination of software and hardware can improve efficiency, speed up training, and optimise model performance. Table I shows the specifications of the devices used in this study.

TABLE I
SOFTWARE AND HARDWARE SPECIFICATIONS

Component	Specification
RAM	32 GB
Graphic Card	NVIDIA RTX 3050
Processor	Intel® Core™ i7-12700
CUDA	v.12.6
Anaconda	Conda v24.9.2
Python	V3.13.2

B. Dataset

The CICMalDroid2020 dataset was selected as a benchmark because it represents one of the largest and most recent public datasets that provides comprehensive dynamic features of Android applications, including system calls, binders, and composite behaviours. This dataset contains 4 CSV files.

(1) 'feature_vectors_static'
(2) 'syscall_unique'
(3) 'feature_vectors_syscallsbinders_frequency_5_Cat'
(4) 'feature_vectors_syscalls_frequency_5_Cat'.

File (4) feature_vectors_syscallsbinders_frequency_5_Cat is the main file used in this study, comprising 17,341 APK samples collected; only 13,077 were successfully executed. At the same time, the rest failed due to timeouts, invalid APK files, or memory allocation failures [18]. The file contains 11,598 data points with 470 features and one target column covering five classes: Adware, Banking, SMS Malware, Riskware, and Benign.

C. Data Preparation

After the data required for this study are collected, the next step is to process them. This step is important for maintaining data quality because data is the main fuel for machine learning models. The better the data preparation, the better the model.

1) Deduplication of Data: Removing duplicate data is the first step in maintaining data validity and preventing the model from memorising the training data to the point of bias or overfitting [22]. Table II below shows the amount of data before and after removing duplicates.

TABLE II
DEDUPLICATION OF DATA

Class	Before	After
SMS Malware	3904	3903
Riskware	2546	2534
Banking	2100	2044
Benign	1795	1792
Adware	1253	1253

2) Handling Missing Values: Missing values are crucial for optimal machine learning model performance, as non-random missingness can introduce bias, reduce accuracy, and affect precision and recall. If ignored, it can undermine model generalisation and raise ethical concerns, such as discrimination, in sensitive applications [23]. In this study, the dataset is free of missing values.

3) Feature and Target Separation: After data cleaning, the input variables (X) are separated from the target (Y), where X includes all application attributes, and Y represents the class (e.g., malware or benign). This separation is essential for the supervised model to learn the mapping from X to Y. Next, the target categories are encoded numerically to support algorithms such as XGBoost.

4) Data Splitting: The dataset is divided into two subsets, train and test, with an 80:20 split, where the training subset contains 9920 data points, and the test subset contains 2306 data points.

D. Feature Selection:

This study employs two feature selection methods simultaneously to achieve optimal results and align with the configured model. The methods are mutual information and feature importance embedding.

Mutual Information (MI) is a machine learning technique that measures statistical dependence between features and target variables. It helps identify the most informative features for classification tasks. MI quantifies the reduction in uncertainty about the target variable based on knowledge of the feature values. The process involves calculating an MI score for each feature and sorting the features in descending order by score. The final feature set used for modelling includes the top features that show a strong relationship with the target variable. The MI between two random variables, X and Y, is given by Equation (1). Where, the $I(X;Y)$ is the mutual information between two random variables, namely X and Y. $p(x,y)$ is the joint probability function of X and Y, whereas $p(x)$ and $p(y)$ are the probability functions of each of X dan YIn the context of feature selection, X usually represents a feature (characteristic of the data), and Y represents the target variable (what will be predicted). Mutual information (MI) measures how much knowledge about the features X can help reduce uncertainty in predicting Y [24].

$$I(X;Y) = \left(\sum_{x\in X}\sum_{y\in Y} p(x,y)\cdot\frac{p(x,y)}{p(x)\cdot p(y)}\right) \quad (1)$$

There are many ways to select features. Some of these ways include wrapper-based Recursive Feature Elimination (RFE), embedded Lasso regularization, and dimensionality reduction via Principal Component Analysis (PCA). Another way is to use metaheuristic genetic algorithms. Mutual Information (MI) was chosen for its ability to capture both linear and non-linear dependencies between features and the multi-class target. It does this without assuming data distribution or linearity. MI is efficient and works with any model. It is an effective way to filter high-dimensional data (470 features).

On the other hand, RFE is expensive because it requires repeated model training and risks overfitting. Lasso favours linear relationships, PCA yields uninterpretable features by ignoring target relevance, and genetic algorithms require excessive evolutionary iterations, making them unsuitable for detecting mobile malware when resources are limited. MI is

DOI : https://doi.org/10.25139/inform.v11i1.11220

light-weight, it can be used easily with other programs. It works with tree-based models, where it quickly finds important features and FI handles how they interact with the context. This makes MI useful for real-time Android applications.

The next feature selection method is feature importance, which is built into tree-based models such as XGBoost (XGB), Random Forest (RF), and LightGBM (LGBM). Feature importance is an internal mechanism that calculates each feature's contribution to the model prediction based on the ensemble's decision tree structure. In RF, feature importance is typically calculated using mean decrease in impurity. Each feature is evaluated based on how much it reduces the average impurity (e.g., Gini or entropy) across the tree. The feature that reduces impurity the most is considered the most important. In XGB and LGBM, which are gradient-boosting variants, feature importance is measured using metrics such as gain, coverage, and frequency. Gain measures the total reduction in the loss function (e.g., mean squared error (MSE) or log loss) achieved by the split on that feature across the tree. Cover counts the number of samples affected by the split. Frequency counts how many times the feature was used for splitting. The process begins with tree construction. At each node, the algorithm searches for the optimal feature and split point to maximize gain. Importance scores are then accumulated across the ensemble to determine feature rankings. These rankings are useful for model interpretability, feature selection, and reducing data dimensions without sacrificing performance [25] [26].

The complementary integration of MI and FI is therefore model-adaptive rather than uniform. MI is prioritized for classifiers that rely on global feature relevance and distance metrics. At the same time, FI is emphasized for ensemble tree-based models that benefit from contextual and interaction-aware feature evaluation. This model-specific justification ensures that feature selection aligns with the inductive bias of each learning algorithm, leading to improved generalisation, preserved interpretability, and enhanced computational efficiency in high-dimensional malware-detection scenarios.

E. Machine Learning Model

The CICMalDroid2020 dataset is evaluated in this study using four models: LGBM, XGBoost, Random Forest, and KNN.

1) LightGBM (LGBM): The Light Gradient Boosting Machine (LightGBM or LGBM) is a machine learning algorithm based on Gradient Boosted Decision Tree (GBDT) developed by Microsoft. The objective of this algorithm is to enhance efficiency, speed, and scalability in model training, particularly for large, high-dimensional data. In contrast to conventional GBDT, which necessitates a thorough examination of all data samples to ascertain the optimal separation point at each tree node, this approach employs a more streamlined methodology. LGBM introduces two primary techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS prioritises the use of samples with substantial gradients, which are recognised as the primary drivers of model updates. This approach accelerates computation without compromising accuracy. Concurrently, EFB integrates a set of exclusive features, thereby reducing the number of features that require processing and enhancing memory efficiency. This approach enables LGBM to generate models with high performance, computational efficiency, and applicability across a range of supervised learning problems, including classification, regression, and ranking [27], [28]. The mathematical expression describing the model update process is given by Equation (2). Where, the $F_m(x)$ is the model in the iteration to$m$, $h_m(x)$is a new decision tree (weak learner), and $\rho_m$is the weighting coefficient. LightGBM optimizes the loss function by adding regularization to control model complexity, as shown in Equation (3). Where, the regularization function $\Omega(h)$Equation (4), with T is the number of leaves on a tree, ω the weight of each leaf, and γ, λ is a parameter that controls complexity.

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \tag{2}$$

$$L(F) = \sum_{i=1}^{n} L(y_i, F(x_i)) + \sum_{m=1}^{M} \Omega(h_m) \tag{3}$$

$$\Omega(h) = \gamma T + \frac{1}{2}\lambda \parallel \omega \parallel^2 \tag{4}$$

2) XGBoost (XGB): A gradient boosting-based machine learning algorithm created by Chen and Guestrin [29], is an improvement on the Gradient Boosted Regression Trees (GBRT) framework. This algorithm using Equation (5) utilizes parallel and iterative decision tree building techniques to reduce prediction errors, resulting in increased computational efficiency, accuracy, and reliability, particularly for solving complex scientific data problems. [30]. Where, the l variable is a loss function, y_i is the actual target label, $\hat{y}_i^{(t-1)}$ is the result of predictions in previous iterations, $f_t(x_i)$ is the decision tree added in t-th iteration, $\Omega(f_t)$ is a regularization function to control model complexity.

$$\mathcal{L}(t) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{5}$$

3) Random Forest (RF): Machine learning approaches for regression and classification. RF works by building multiple decision trees from random subsets of data and features (bagging), which aims to reduce correlation between trees and reduce overfitting. In classification, the final result is determined by majority voting. The main strengths of RF include high accuracy, the ability to handle large/unbalanced data, tolerance for missing data, and the ability to identify feature importance. RF is widely used in sectors such as banking, healthcare, and e-commerce. Its main weakness is the long training time if too many trees are built [31]. Where, the y variable is the prediction result, showing the input feature vector, and let represent the total number of decision trees used in the model. The $Ti(x)$ variable is a prediction made by Random Forest.

$$y = f(x) = \sum(i = 1 \; to \; n) \, Ti(x)/n \tag{6}$$

DOI : https://doi.org/10.25139/inform.v11i1.11220

4) K-Nearest Neighbours (KNN): A classification method that compares new data to training data based on feature proximity, measured using distance metrics such as Euclidean or Manhattan distance [32]. The method involves calculating the distance between new data points (y_i) and all training data points (x_j) using Equation (7) or Equation (8). Where, the P variable is the number of features. Then, k the closest point is selected, and the majority class of k that point is used to classify new data.

$$d = \sqrt{\sum_{p=1}^{P}(y_{i,p} - x_{j,p})^2} \quad (7)$$

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \quad (8)$$

Where, the $d(x, y)$ is the function of the distance between two points x and y; $x_i - y_i$ refers to the i-th component of the two points $x$dan y, and n is the number of dimensions in the space where the two points are located. So, if we are working in two-dimensional space, $n = 2$, and if in three-dimensional space, $n = 3$.

F. Evaluation

One of the important stages in the machine learning process is evaluation, which assesses the model's accuracy and effectiveness in achieving the research objectives. At this stage, appropriate evaluation metrics are used to measure the predictive ability of each algorithm. Some metrics commonly used in classification problems include precision (Equation (10), recall (Equation (11), accuracy (Equation (9), and F1-score (Equation (12). Each of these metrics assesses different aspects of model performance. Accuracy indicates the overall correctness of the model's predictions, while the F1-score combines precision and recall, making it particularly useful when the data are unbalanced.

Positive Predictive Value (PPV) or Precision measures how accurate the model is at predicting positive outcomes, that is, the extent to which the model's positive predictions are actually correct [33]. Recall, also known as Sensitivity or True Positive Rate, is used to measure the extent to which a model correctly identifies positive data from all data that is actually positive [33]. Errors in classifying Android applications, such as applications that contain malware but are mistakenly considered safe, can have serious consequences. Generally, a high recall value often leads to a decrease in precision, and vice versa. Therefore, the F1-score is used as a metric to balance these two measures [33].

True Positive (TP) occurs when the model correctly classifies data as positive. True Negative (TN) indicates that the model is also accurate when classifying data as negative. Conversely, a False Positive (FP) occurs when negative data is incorrectly classified as positive. At the same time, a False Negative (FN) indicates that the model failed to detect a positive instance, leading it to be misclassified as negative. A good understanding of these four performance metrics is essential to improving the effectiveness of model evaluation in distinguishing between application classes, such as Adware, Riskware, Benign, and others [33].

$$Accuracy = \frac{TP + TN}{(TP+FP+TN+FN)} \quad (9)$$

$$Precission = \frac{TP}{TP+FP} \quad (10)$$

$$Recall = \frac{TP}{TP+FN} \quad (11)$$

$$F1 = \frac{2\ x\ Recall\ x\ Precision}{Recall + Precision} \quad (12)$$

In addition to the four measurement matrices above, OvR ROC is also used to evaluate the performance of each class. The Receiver Operating Characteristic (ROC) curve for One vs Rest (OvR) is a graph that illustrates the performance of binary classification when converting a multi-class classification problem into separate binary classification tasks. In the One vs Rest approach, this means creating separate binary classifications in which one class is distinguished from all others [34]. OvR is a common strategy for multi-class classification, typically used with support vector machines. This method is generally faster and less complex than other multi-class classification methods [35] [36]. To obtain more robust measurements, the StratifiedKFold Cross-Validation method is also employed during the evaluation stage. Stratified K-Fold Cross-Validation (SKCV) represents an extension of the conventional K-Fold technique, specifically designed to address classification problems characterized by imbalanced class distributions [37]. This technique is implemented by dividing the data set into *k* approximately equal-sized groups, or folds [38].

In contrast to conventional K-Fold, SKCV ensures that each fold maintains the same class-label proportions as the original data set [39]. In the process, the data are randomised and split so that the relative class frequencies are maintained at each stage of training and validation. This approach ensures that each data point has an equal chance of being included in the test set. Consequently, this method helps minimise bias and variance in evaluating machine learning model performance.

III. RESULT AND DISCUSSION

This study applied four machine learning models, namely XGBoost (XGB), Random Forest (RF), K-Nearest Neighbors (KNN), and LightGBM (LGBM), with two feature selection methods, namely Mutual Information (MI) and the model's built-in Feature Importance (FI). This experiment successfully demonstrated that applying feature selection reduced data dimensionality while improving model performance.

1) Baseline: Before conducting the feature selection experiment, an initial model was created as a reference using all features in the dataset (470 features). The results are presented in Table III. Among the models evaluated, LGBM achieved the highest performance with an F1-score of 0.9537, precision of 0.9520, recall of 0.9558, and accuracy of 0.9614. XGB closely followed, with an F1-score of 0.9523 and an

DOI : https://doi.org/10.25139/inform.v11i1.11220

accuracy of 0.9605, while RF and KNN showed comparatively lower but still competitive results. These baseline scores, obtained using all available features without any selection or reduction, serve as the reference point for evaluating the effectiveness of subsequent feature selection methods.

TABLE III
BASELINE PERFORMANCE (MACRO AVG)

Baseline					
Model	**Precision**	**Recall**	**F1**	**Accuracy**	**Total Feature**
LGBM	0.9520	0.9558	0.9537	0.9614	470
XGB	0.9509	0.9541	0.9523	0.9605	470
RF	0.9438	0.9455	0.9443	0.9536	470
KNN	0.8827	0.8797	0.8760	0.9003	470

2) *Feature Selection:* After obtaining the best baseline, the next step is to conduct experiments using feature selection. The model's performance is evaluated using either the mutual information feature selection method or the model's built-in feature importance method. In this study, the automation approach is implemented as an iteration in the program to select features for each model, with $50 \leq n \leq 475$ features. Fig.2 shows the optimal number of features for each model. Frequently selected feature samples are shown in Table IV. From the analysis of the search results, the best number of features for each LGBM model was 270, when combined with FI, achieving an accuracy of 0.9649. XGB achieved its best score with FI and 225 features, achieving an accuracy of 0.9631. Meanwhile, RF and KNN achieved their best performance when combined with MI, using 240 and 135 features, respectively, where both obtained their best accuracy scores of 0.9562 for RF and 0.9137 for KNN. Fig.3 compares the accuracy of the top combination model and feature selection with that of the baseline model. Table V shows the confusion matrices for each best model after feature selection and testing on the test data.

TABLE IV
FREQUENTLY SELECTED FEATURE SAMPLES

getDeviceId	connect
Privacy Breach	Network Operation
NETWORK_ACCESS____	getSubscriberId
Network Exfil	Privacy Breach
ACCESS_PERSONAL_INFO___	checkPermission
Privacy Breach	Privilege Escalate
NETWORK_ACCESS(WRITE)____	FS_ACCESS____
Network Exfil	File System
registerContentObserver	getPackageInfo
Stealth Operation	Android API
read	getLine1Number
System Call	Privacy Breach
write	open
System Call	System Call
socket	mmap2
Network Operation	System Call
CREATE_THREAD_____	getActiveNetworkInfo
Behavioral	Network Operation
setComponentEnabledSetting	isAdminActive
Privilege Escalate	Privilege Escalate

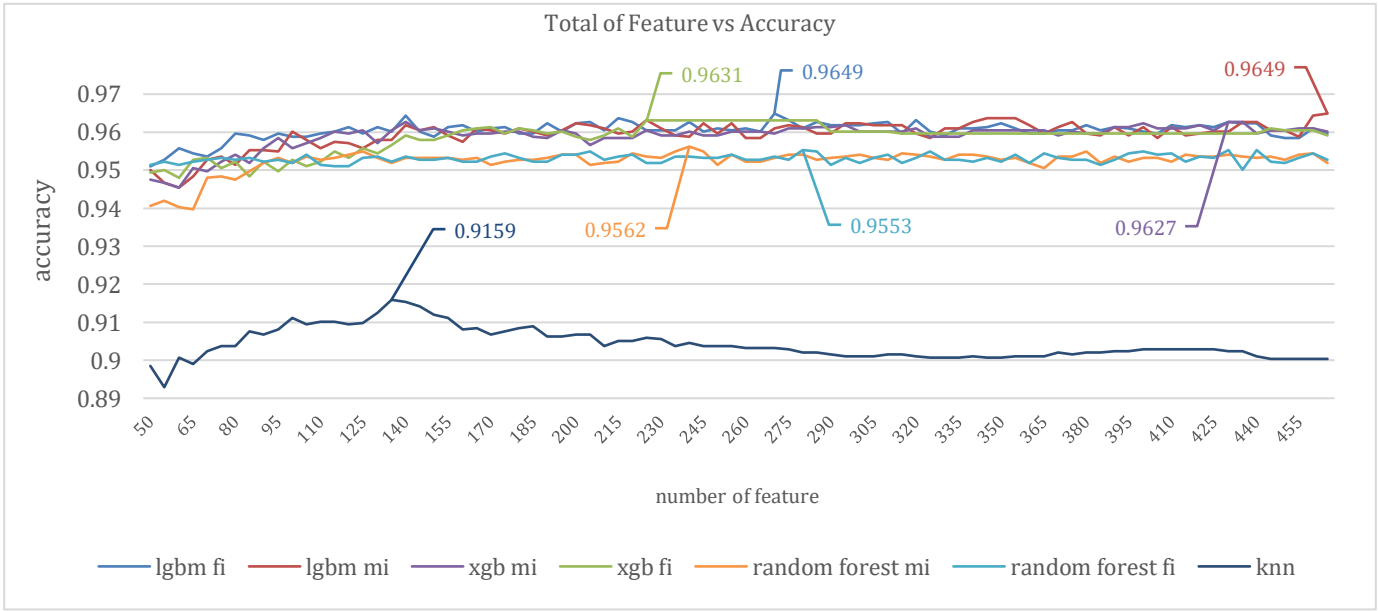

Fig.2. Accuracy Vs Number Of Features

From the analysis of the search results, the best number of features for each LGBM model was 270, when combined with FI, achieving an accuracy of 0.9649. XGB achieved its best score with FI and 225 features, achieving an accuracy of 0.9631. Meanwhile, RF and KNN achieved their best performance when combined with MI, using 240 and 135 features, respectively, where both obtained their best accuracy scores of 0.9562 for RF and 0.9137 for KNN. Fig.3 compares the accuracy of the top combination model and feature selection with that of the baseline model. Table V shows the confusion matrices for each best model after feature selection and testing on the test data.

DOI : https://doi.org/10.25139/inform.v11i1.11220

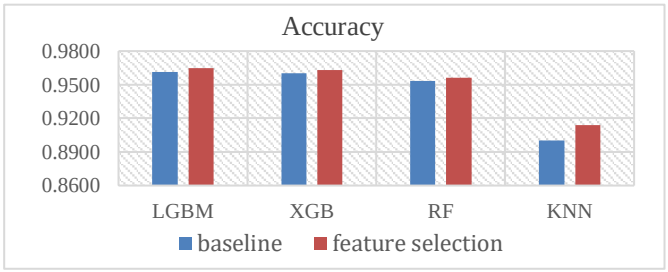

Fig. 3. baseline accuracy vs feature selection

TABLE V
CONFUSION MATRIX

	Confusion Matrix		Predicted: adware	banking	sms malware	riskware	benign
Actual	LGBM FI	adware	238	2	3	5	3
		banking	6	385	3	7	8
		sms malware	0	2	776	2	1
		riskware	17	5	1	474	10
		benign	2	4	2	6	344
	XGB FI	adware	235	1	4	7	4
		banking	7	385	3	8	6
		sms malware	1	2	776	1	1
		riskware	16	5	2	473	11
		benign	3	4	2	3	346
	RF MI	adware	233	0	4	6	8
		banking	6	382	5	6	10
		sms malware	0	4	775	1	1
		riskware	17	2	3	475	10
		benign	5	3	6	4	340
	KNN MI	adware	228	6	6	5	6
		banking	15	368	10	11	5
		sms malware	0	5	773	3	0
		riskware	24	11	6	456	10
		benign	48	20	17	22	251

The confusion matrix shows that all four models (LGBM, XGB, RF, and KNN) performed well in most malware categories, with high True Positive (TP) values. The LGBM FI model performed best overall, achieving the highest TP in the adware (238), banking (385), SMS malware (776), riskware (474), and benign (344) categories. This indicates that it generalizes better than other models. XGB FI performed almost as well as LGBM FI. It had a precision of 235 for adware, 385 for banking, 776 for SMS malware, 473 for riskware, and 346 for benign. This shows that it was very consistent in its classification.

In the SMS malware category, all models showed very consistent performance, with TP ranging from 773 to 776 and very low False Positives (FP), indicating that SMS malware exhibits the most distinctive and easily identifiable patterns. The banking category also showed excellent performance across all models, with TP ranging from 368 to 385; LGBM FI and XGB FI achieved the highest values (385), indicating that banking malware features are very well identified. RF MI shows slightly lower performance in the banking and adware categories, with TP of 382 and 233, respectively, but still maintains high accuracy in the SMS malware (775) and riskware (475) categories.

The KNN MI model performed worst among the models, especially in the benign category, with only 251 TP and a very high FP rate across the adware (87), banking (42), SMS malware (39), and riskware (42) categories. The largest classification error in KNN MI occurred in the benign category, which was often misclassified as adware, suggesting that KNN MI struggled to distinguish benign applications from potentially unwanted applications (adware). False positives in the riskware category were higher across all models, with benign and adware applications often misclassified as riskware, suggesting overlap in their characteristics.

LGBM FI and XGB FI exhibit very low and well-controlled FP rates. RF MI has a good balance between sensitivity and specificity with consistent performance across all categories, although slightly below LGBM FI and XGB FI. Overall, boosting-based ensemble methods (LGBM FI and XGB FI) outperform RF MI and KNN MI in terms of prediction accuracy, with LGBM FI showing the best performance, especially in challenging categories such as benign, with a TP of 344 compared to XGB FI (346), RF MI (340), and KNN MI (251). Although XGB FI is slightly superior in the benign category, LGBM FI is more consistent in other categories. Meanwhile, a comparative evaluation of all model combinations and feature selection methods yielded the optimal configuration shown in Table VI.

TABLE VI
PERFORMANCE AFTER FEATURE SELECTION (MACRO AVG)

Model	Precision	Recall	F1	Accuracy	Total Feature
LGBM FI	0.9556	0.9596	0.9574	0.9649	270
LGBM MI	0.9543	0.9598	0.9568	0.9649	465
XGB FI	0.9549	0.9565	0.9556	0.9631	225
XGB MI	0.9534	0.9571	0.9550	0.9627	430
RF FI	0.9456	0.9479	0.9465	0.9553	280
RF MI	0.9468	0.9482	0.9472	0.9562	240
KNN MI	0.8955	0.8962	0.8940	0.9137	135

To further validate the robustness of the proposed models and address potential biases from single data splits, 5-fold cross-validation was performed on each of the best models. As shown in Table VII, all models exhibit low fold-to-fold variance (standard deviation < 0.4% for accuracy), indicating high stability. LGBM FI achieves the highest cross-validated performance, with 95.24% ± 0.31% accuracy and a macro-ROC AUC of 0.9961 ± 0.0008, closely followed by XGB FI. On the independent hold-out test set, LGBM FI achieves the best results with 96.49% accuracy and 0.9970 macro-ROC-AUC, confirming its superior discriminative ability across all malware categories. These additional metrics align with common benchmarking practices in multi-class malware detection and reinforce the effectiveness of the hybrid feature selection approach. For details on the ROC - AUC curve for each model, see Fig.4 (LGBM FI), Fig.5 (XGB FI), Fig.6 (RF MI), and Fig.7 (KNN MI).

DOI : https://doi.org/10.25139/inform.v11i1.11220

TABLE VII
CROSS VALIDATION RESULT

Model	Features	CV Accuracy	CV F1 Macro	CV Macro ROC-AUC	Hold-Out Accuracy	Hold-Out Macro ROC-AUC
LGBM FI	270	0.9524 ± 0.0031	0.9430 ± 0.0040	0.9961 ± 0.0008	0.9649	0.9970
XGB FI	225	0.9538 ± 0.0037	0.9448 ± 0.0043	0.9960 ± 0.0007	0.9631	0.9969
RF MI	240	0.9489 ± 0.0029	0.9387 ± 0.0034	0.9955 ± 0.0008	0.9562	0.9960
KNN MI	135	0.9063 ± 0.0019	0.8850 ± 0.0026	0.9590 ± 0.0025	0.9137	0.9631

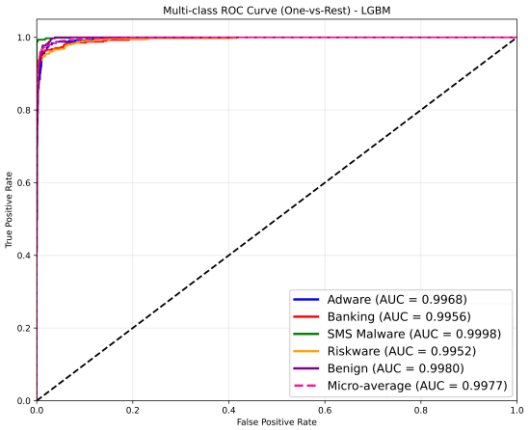

Fig.4. Roc Curve - LGBM FI

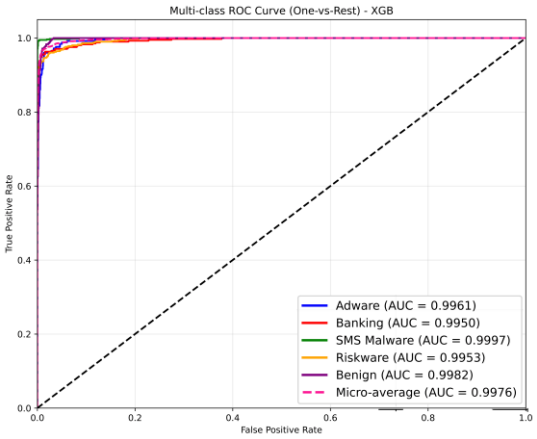

Fig.5. Roc Curve – XGB FI

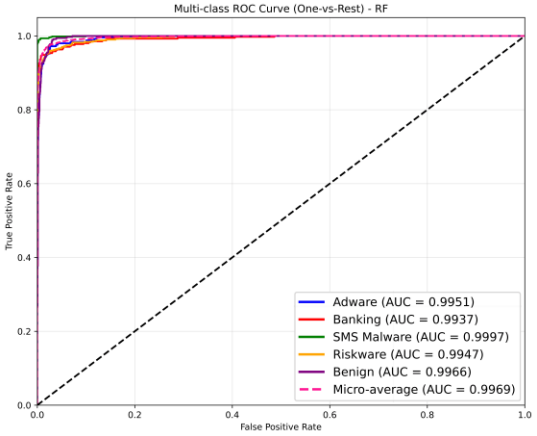

Fig.6. Roc Curve – RF MI

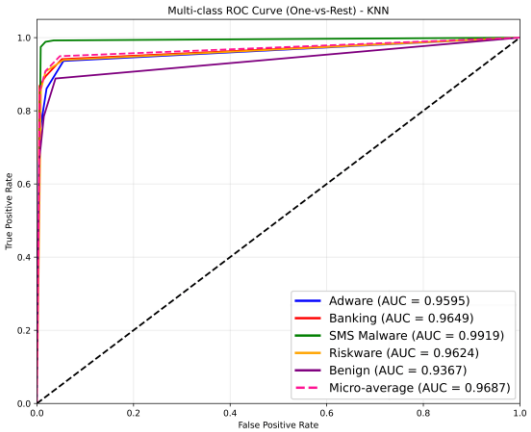

Fig.7. Roc Curve – KNN MI

In addition to performance metrics, this study also measured the testing time of each model. LGBM with feature importance excelled overall, achieving balanced metrics and effectively capturing non-linear dependencies in multi-class datasets, with training time of 3.9701 seconds and test time of 0.0360 seconds. XGB FI reduced features by 47.8% (from 470 to 225) without compromising performance, improving efficiency and reducing overfitting with good recall and precision, using a training time of 7.7700 seconds and a testing time of 0.0429 seconds. RF MI achieved an accuracy of 0.9562 and an F1-score of 0.9472, with a training time of 7.6804 seconds and a testing time of 0.1253 seconds. In contrast, KNN MI had the lowest accuracy of 0.9137 and an F1-score of 0.8940, possibly due to its sensitivity to high dimensions and a test-time of 0.2235 seconds for data distance calculations. For a clearer comparison, see Fig.8 and Table VIII.

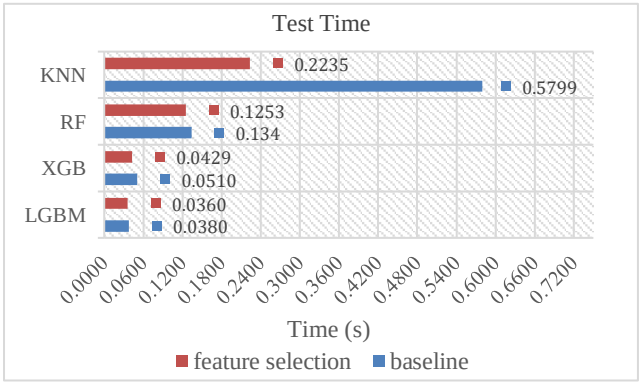

Fig.8. Test Time Comparison

DOI : https://doi.org/10.25139/inform.v11i1.11220

TABLE VIII
COMPARISON OF FEATURE SELECTION VS BASELINE

Baseline			After Feature Selection		
Model	**F1**	**Test Time**	**Model**	**F1**	**Test Time**
LGBM	0.9537	0.0380	LGBM FI	0.9574	0.0360
XGB	0.9523	0.0510	XGB FI	0.9556	0.0429
RF	0.9443	0.1340	RF MI	0.9472	0.1253
KNN	0.8760	0.5799	KNN MI	0.8940	0.2235

The advantages of LGBM FI achieved an accuracy of 96.49% and an F1-score of 0.9574 on 270 features, due to the good cooperation between the algorithm structure and the nature of dynamic syscall data. The Gradient-based One-Side Sampling (GOSS) technique in LGBM selectively prioritises samples with high error gradients. At the same time, Exclusive Feature Bundling (EFB) combines mutually exclusive features, capturing complex non-linear interactions between system call frequencies without requiring full computation over 470 features. This results in an accuracy gain of +0.36% compared to the full features baseline, while also increasing testing speed by up to 6%, because FI intrinsically suppresses redundant noise during boosting iterations, unlike Mutual Information (MI), which is more suitable for ensemble models such as RF, where global statistical dependencies dominate. The feature selection process produces four different configurations with significantly varying numbers of features. The LGBM model with Feature Importance (FI) selected 270 features, XGBoost FI selected 225 features, Random Forest with Mutual Information (MI) selected 240 features, and K-Nearest Neighbors MI selected 135 features. This variation in the number of features reflects the fundamental difference between the tree-based Feature Importance method and Mutual Information, which measures the statistical dependence between features and the target label.

Although KNN MI has the smallest feature set, its classification performance is the lowest, with an accuracy of 91.37%, precision of 89.55%, recall of 89.62%, and F1-score of 89.40%. Conversely, LGBM FI with 270 features achieved the best performance (96.49% accuracy, 95.56% precision, 95.96% recall, 95.74% F1-score), followed by XGBoost FI with 225 features (96.31% accuracy, F1-score 95.56%) and RF MI with 240 features (accuracy 95.62%, F1-score 94.72%). This phenomenon indicates that although the 135 KNN MI features are the most discriminative core features, the additional features selected by tree-based models contribute significantly to capturing more complex and subtle nuances in malware behaviour.

The results of the experiment provide actionable insights for practical malware detection, with the optimal range of 225-270 features in this dataset providing a sweet spot between comprehensiveness and efficiency. XGBoost with 225 features is only 0.18% below LGBM (270 features) but 16.7% more efficient, making it ideal for real-time scanning. The Feature Importance method of tree-based algorithms (such as LGBM, XGBoost, and Random Forest) was about 5-6% superior to Mutual Information. This advantage arises because Feature Importance can capture complex relationships among features that often occur in malware behaviour. For example, the combination of the getDeviceId feature with NETWORK_ACCESS is only dangerous when they occur together, and patterns like this are easier to detect by tree-based methods. Hierarchical overlap analysis shows that the 135 core features selected using KNN with Mutual Information constitute a near-complete subset of the tree-based feature selections (>94% overlap), revealing a clear hierarchy of feature relevance and validating the robustness of this minimal feature set. Despite yielding approximately 5% lower accuracy compared to the full model (91.37% vs. 96.49%), these 135 features remain highly effective for ultra-lightweight deployment, making them well-suited for fast initial screening and tiered or hierarchical detection systems on resource-constrained platforms such as entry-level smartphones, where speed and memory efficiency are critical.

A comprehensive evaluation of Android malware detection models requires a systematic comparison with prior studies that use similar datasets. This comparison is important for identifying scientific contributions and validating the effectiveness of the proposed approach in the context of state-of-the-art malware detection research. In this study, the comparison focuses on studies that use the CICMalDroid2020 dataset and apply machine learning approaches to Android malware classification. Table IX presents a comprehensive comparison between the results of this study and related studies, covering various machine learning and deep learning algorithms.

TABLE IX
COMPARISON WITH PREVIOUS RESEARCH

Model	Precision	Recall	F1	Accuracy	Fitur
LGBM FI	0.9556	0.9596	0.9574	0.9649	270
XGB FI	0.9549	0.9565	0.9556	0.9631	225
RF MI	0.9468	0.9482	0.9472	0.9562	240
KNN MI	0.8955	0.8962	0.8940	0.9137	135
LGBM [21]	0.9296	0.9148	0.9282	0.9279	200
RF [21]	0.9244	0.9075	0.9214	0.9206	100
XGB [19]	0.9362	0.9349	0.9350	0.9349	89
LGBM [19]	0.9410	0.9395	0.9396	0.9395	89
CNN LSTM [40]	0.9330	0.9445	0.9578	0.9439	470
KNN [41]	-	-	-	0.9145	470

Based on Table VIII, the LGBM FI model achieved the highest performance with precision of 95.56%, recall of 95.96%, F1-score of 95.74%, and accuracy of 96.49% using 270 features. These results surpassed those of research [21] with an accuracy increase of 3.70% and research [19] with an accuracy improvement of 2.54%. The XGB FI model also showed superior performance with an accuracy of 96.31%, surpassing previous studies [19] by 2.82%. Comparison with CNN LSTM [40], which achieved an F1 score of 95.78% using 470 features, shows that LGBM FI achieves competitive performance with higher feature efficiency (270 features) and significantly lower computational demands, making our method more suitable for real-time applications. This positions our hybrid selection as a leading advancement in balancing accuracy and efficiency. Even the RF MI model with an accuracy of 95.62% outperformed the RF implementation [21]

DOI : https://doi.org/10.25139/inform.v11i1.11220

by 3.56%, while KNN MI with 135 features remains competitive compared to KNN [41] with 470 features [21]

IV. CONCLUSION

This study developed an efficient machine-learning-based Android malware classifier by optimising feature selection on the CICMalDroid2020 dataset. LightGBM with Feature Importance excelled (96.49% accuracy, 95.74% F1-score, 0.036 seconds testing time) using 270 features (42.6% reduction), outperforming KNN with Mutual Information by up to 5–6% in capturing non-linear dependencies such as the contextual combination of getDeviceId and NETWORK_ACCESS. A total of 135 core features from KNN MI overlap >94% with tree-based FI models, distributed across system calls (40%), API calls (25%), network (15%), file system (10%), and behavioural patterns (10%), focusing on privacy breaches, data exfiltration, privilege escalation, and stealth operations. The optimal range of 225–270 features is the sweet spot; XGBoost (225 features) is only 0.18% below LightGBM but 16.7% more computationally efficient, suitable for real-time scanning.

The primary contribution of this work is a light-weight yet highly accurate Android malware classification model enabled by a model-adaptive hybrid feature selection framework that synergistically combines Mutual Information (MI) and embedded Feature Importance (FI). Unlike prior studies [19], which relied on destructive undersampling and outlier removal, or study [21], which applied a single feature selection method (e.g., PCA) uniformly across models, our approach tailors feature selection to the inductive bias of each classifier. Through automated iterative evaluation (50–475 features), we identify model-specific "sweet spots" (e.g., 225–270 features) that reduce dimensionality by up to 52% while maintaining or even improving performance. This strategy eliminates information loss from sampling, enhances generalization, and yields a computationally efficient solution well-suited for real-time deployment on resource-constrained Android devices.

This research makes a significant contribution to the development of efficient, accurate, and applicable Android malware detection techniques for real-time security systems. However, the study is subject to limitations imposed by the CICMalDroid2020 dataset, which predominantly comprises samples collected up to 2018. Given the rapid evolution of Android malware, including advanced obfuscation techniques, exploitation of newer Android API levels, and new threats since 2018, this time constraint significantly limits the model's ability to generalise to contemporary malware variants. To improve this approach, it is recommended that the proposed method be evaluated and extended to newer datasets or to real-world telemetry logs from security vendors, to address evolving threats and ensure long-term viability, and to guide necessary retraining strategies.

The proposed models have not been explicitly evaluated against adversarial evasion techniques, such as metamorphic transformations, code obfuscation beyond those present in the dataset, or adversarial examples crafted to mislead gradient-based classifiers. Tree-based ensembles like LightGBM and XGBoost generally exhibit moderate robustness to small perturbations compared to neural networks, but remain vulnerable to mimicry attacks that mimic benign behaviour on selected features. Future work should incorporate adversarial training, robustness testing using frameworks such as SecML or ART, and evaluation on datasets containing deliberately evaded samples (e.g., via obfuscation tools like Obfuscator-LLVM) to assess real-world resilience better better. Subsequent endeavours should incorporate comprehensive hyperparameter optimisation (e.g., via optuna or grid search) to maximise performance gains, particularly in KNN and Random Forest. This approach has the potential to elevate accuracy beyond current levels while preserving efficiency.

REFERENCES

[1] D. Das, S. M. Satapathy, A. D, and A. Agarwal, "Application of Hybrid Approach towards Multi Aspect Classification and Analysis of Malware," in *2023 OITS International Conference on Information Technology (OCIT)*, Dec. 2023, pp. 284–289. doi: 10.1109/OCIT59427.2023.10430958.

[2] A. Buriro, A. B. Buriro, T. Ahmad, S. Buriro, and S. Ullah, "MalwD&C: A Quick and Accurate Machine Learning-Based Approach for Malware Detection and Categorization," *Applied Sciences*, vol. 13, no. 4, p. 2508, Feb. 2023, doi: 10.3390/app13042508.

[3] A. Redhu, P. Choudhary, K. Srinivasan, and T. K. Das, "Deep learning-powered malware detection in cyberspace: a contemporary review," *Front. Phys.*, vol. 12, Mar. 2024, doi: 10.3389/fphy.2024.1349463.

[4] A. Guerra-Manzanares, "Machine Learning for Android Malware Detection: Mission Accomplished? A Comprehensive Review of Open Challenges and Future Perspectives," *Computers & Security*, vol. 138, p. 103654, Mar. 2024, doi: 10.1016/j.cose.2023.103654.

[5] K. Brezinski and K. Ferens, "Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits," 2023, doi: 10.1155/2023/8227751.

[6] P. Arora, R. Gupta, N. Malik, and A. Kumar, "Malware Analysis Types & Techniques : A Survey," in *Proceedings of the 5th International Conference on Information Management & Machine Intelligence*, in ICIMMI '23. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 1–6. doi: 10.1145/3647444.3652439.

[7] E. Al. Rajesh Yadav, "Malware Detection and Analysis Tools," *IJRITCC*, vol. 11, no. 11s, pp. 735–744, Nov. 2023, doi: 10.17762/ijritcc.v11i11s.9817.

[8] J. Ferdous, R. Islam, A. Mahboubi, and M. Z. Islam, "A Survey on ML Techniques for Multi-Platform Malware Detection: Securing PC, Mobile Devices, IoT, and Cloud Environments," *Sensors*, vol. 25, no. 4, p. 1153, Jan. 2025, doi: 10.3390/s25041153.

[9] John Oluwafemi Ogun, "Advancements in automated malware analysis: evaluating the efficacy of open-source tools in detecting and mitigating emerging malware threats to US businesses," *Int. J. Sci. Res. Arch.*, vol. 12, no. 2, pp. 1958–1964, Aug. 2024, doi: 10.30574/ijsra.2024.12.2.1488.

[10] G. M. and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Computer Science Review*, vol. 47, p. 100529, Feb. 2023, doi: 10.1016/j.cosrev.2022.100529.

[11] V. Jyothsna, P. Mokshitha, S. Khulud, L. G. Premanath Reddy, N. J. Reddy, and B. Pydala, "Advancing Android Security: Leveraging Stacking Ensemble and Bioinspired Feature Selection for Efficient Malware Detection," *2024 5th International Conference for Emerging Technology (INCET)*, pp. 1–11, May 2024, doi: 10.1109/INCET61516.2024.10593208.

[12] I. S. Makkar, A. K. Sinha, T. Pratap, H. Pandey, and B. Nandwana, "Android Malware Detection: Necessity, Applications and Future Direction," in *2025 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Mar. 2025, pp. 1–6. doi: 10.1109/IATMSI64286.2025.10985193.

DOI : https://doi.org/10.25139/inform.v11i1.11220

[13] A. Razgallah, R. Khoury, S. Hallé, and K. Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research," *Computer Science Review*, vol. 39, p. 100358, Feb. 2021, doi: 10.1016/j.cosrev.2020.100358.

[14] A. Bensaoud, J. Kalita, and M. Bensaoud, "A survey of malware detection using deep learning," *Machine Learning with Applications*, vol. 16, p. 100546, Jun. 2024, doi: 10.1016/j.mlwa.2024.100546.

[15] A. Brown, M. Gupta, and M. Abdelsalam, "Automated Machine Learning for Deep Learning based Malware Detection," Nov. 03, 2023, *arXiv*: arXiv:2303.01679. doi: 10.48550/arXiv.2303.01679.

[16] L. Alzubaidi *et al.*, "A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications," *J Big Data*, vol. 10, no. 1, p. 46, Apr. 2023, doi: 10.1186/s40537-023-00727-2.

[17] W. Ghozi, H. Lestiawan, R. R. Sani, J. N. Hussein, and F. A. Rafrastara, "XGBoost-Powered Ransomware Detection: A Gradient-Based Machine Learning Approach for Robust Performance," *KINETIK*, Oct. 2025, doi: 10.22219/kinetik.v10i4.2405.

[18] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, Calgary, AB, Canada: IEEE, Aug. 2020, pp. 515–522. doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.

[19] E. G. Villarroel Enriquez and J. Gutiérrez-Cárdenas, "Dynamic Malware Analysis Using Machine Learning-Based Detection Algorithms," *Interfases*, no. 019, pp. 119–138, Jul. 2024, doi: 10.26439/interfases2024.n19.7097.

[20] M. Altalhan, A. Algarni, and M. Turki-Hadj Alouane, "Imbalanced Data Problem in Machine Learning: A Review," *IEEE Access*, vol. 13, pp. 13686–13699, 2025, doi: 10.1109/ACCESS.2025.3531662.

[21] H. AlOmari, Q. M. Yaseen, and M. A. Al-Betar, "A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection," *Procedia Computer Science*, vol. 220, pp. 763–768, 2023, doi: 10.1016/j.procs.2023.03.101.

[22] Y. Zhao *et al.*, "On the Impact of Sample Duplication in Machine-Learning-Based Android Malware Detection," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 3, pp. 1–38, Jul. 2021, doi: 10.1145/3446905.

[23] A. F. Ahmad, M. S. Sayeed, K. Alshammari, and I. Ahmed, "Impact of Missing Values in Machine Learning: A Comprehensive Analysis," Oct. 10, 2024, *arXiv*: arXiv:2410.08295. doi: 10.48550/arXiv.2410.08295.

[24] V. Vajrobol, B. B. Gupta, and A. Gaurav, "Mutual information based logistic regression for phishing URL detection," *Cyber Security and Applications*, vol. 2, p. 100044, Jan. 2024, doi: 10.1016/j.csa.2024.100044.

[25] A. I. Adler and A. Painsky, "Feature Importance in Gradient Boosting Trees with Cross-Validation Feature Selection," *Entropy (Basel)*, vol. 24, no. 5, p. 687, May 2022, doi: 10.3390/e24050687.

[26] U. Ahmed, A. Mahmood, M. A. Tunio, G. Hafeez, A. R. Khan, and S. Razzaq, "Investigating boosting techniques' efficacy in feature selection: A comparative analysis," *Energy Reports*, vol. 11, pp. 3521–3532, Jun. 2024, doi: 10.1016/j.egyr.2024.03.020.

[27] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017.

[28] I. B. Mustapha *et al.*, "Comparative Analysis of Gradient-Boosting Ensembles for Estimation of Compressive Strength of Quaternary Blend Concrete," *Int J Concr Struct Mater*, vol. 18, no. 1, p. 20, Apr. 2024, doi: 10.1186/s40069-023-00653-w.

[29] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[30] S. Hakkal and A. A. Lahcen, "XGBoost To Enhance Learner Performance Prediction," *Computers and Education: Artificial Intelligence*, vol. 7, p. 100254, Dec. 2024, doi: 10.1016/j.caeai.2024.100254.

[31] H. A. Salman, A. Kalakech, and A. Steiti, "Random Forest Algorithm Overview," *Babylonian Journal of Machine Learning*, vol. 2024, pp. 69–79, Jun. 2024, doi: 10.58496/BJML/2024/007.

[32] N. Mukahar, "Performance comparison of k nearest neighbor classifier with different distance functions," *AIP Conf. Proc.*, vol. 2895, no. 1, p. 040010, Mar. 2024, doi: 10.1063/5.0192229.

[33] T. Hu and X.-H. Zhou, "Unveiling LLM Evaluation Focused on Metrics: Challenges and Solutions," Apr. 14, 2024, *arXiv*: arXiv:2404.09135. doi: 10.48550/arXiv.2404.09135.

[34] V. Hurbungs, V. Bassoo, and T. P. Fowdur, "A novel One-vs-Next approach for multi-class classification," *2024 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, Jun. 2024, doi: 10.1109/ISCC61673.2024.10733675.

[35] M. Kudo, T. Takahashi, and H. Yamana, "Touch-Based Continuous Mobile Device Authentication Using One-vs-One Classification Approach," in *2024 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb. 2024, pp. 167–174. doi: 10.1109/BigComp60711.2024.00034.

[36] W. Liu, I. W. Tsang, and K.-R. Muller, "An Easy-to-hard Learning Paradigm for Multiple Classes and Multiple Labels".

[37] J. Allen, H. Liu, S. Iqbal, D. Zheng, and G. Stansby, "Deep learning-based photoplethysmography classification for peripheral arterial disease detection: a proof-of-concept study," *Physiol. Meas.*, vol. 42, no. 5, p. 054002, May 2021, doi: 10.1088/1361-6579/abf9f3.

[38] S. Widodo, H. Brawijaya, and S. Samudi, "Stratified K-fold cross validation optimization on machine learning for prediction," *Sinkron : jurnal dan penelitian teknik informatika*, vol. 6, no. 4, pp. 2407–2414, Oct. 2022, doi: 10.33395/sinkron.v7i4.11792.

[39] S. Prusty, S. Patnaik, and S. K. Dash, "SKCV: Stratified K-fold cross-validation on ML classifiers for predicting cervical cancer," *Front. Nanotechnol.*, vol. 4, p. 972421, Aug. 2022, doi: 10.3389/fnano.2022.972421.

[40] S. Amenova, C. Turan, and D. Zharkynbek, "Android Malware Classification by CNN-LSTM," in *2022 International Conference on Smart Information Systems and Technologies (SIST)*, Nur-Sultan, Kazakhstan: IEEE, Apr. 2022, pp. 1–4. doi: 10.1109/SIST54437.2022.9945816.

[41] Y. Sönmez, M. Salman, and M. Dener, "Performance Analysis of Machine Learning Algorithms for Malware Detection by Using CICMalDroid2020 Dataset," *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, vol. 9, no. 6, pp. 280–288, Dec. 2021, doi: 10.29130/dubited.1018223.

DOI : https://doi.org/10.25139/inform.v11i1.11220