J-INTECH (Journal of Information and Technology)
Accredited Sinta 4 Ministry of Higher Education, Science and Technology Republic of Indonesia SK No. 10/C/C3/DT.05.00/2025
E-ISSN: 2580-720X || P-ISSN: 2303-1425

Optimization of Android Malware Detection Based on Permissions Using LightGBM with Hyperparameter Tuning and Chi-Square Feature Selection

Dede Wintana[1*], Satia Suhada[2], Gunawan[3]

[1,2,3] University Bina Sarana Informatika, , Department Of Enginering and Informatic, University Jl. Cemerlang No.8 Sukakarya, sukabumi – West Java, Indonesia

Keywords
Android; LightGBM; Malware

Corresponding Author:
dede.dwe@bsi.ac.id

Abstract
This study aims to develop an efficient Android malware detection framework using the LightGBM algorithm optimized through Chi-Square feature selection and hyperparameter tuning. Although machine learning techniques have been widely applied in Android malware detection, many previous studies primarily focus on classification accuracy while paying limited attention to feature reduction efficiency, computational complexity, and permission-based feature interpretability. Therefore, this study proposes a lightweight permission-based detection framework that integrates Chi-Square feature selection with optimized LightGBM classification to improve detection performance while reducing irrelevant features. The dataset used was obtained from Kaggle and consists of 29,300 Android applications, including 14,630 benign applications and 14,700 malware samples. Each application is represented using 86 binary permission-based features extracted from the AndroidManifest.xml file. The research stages include preprocessing, feature selection, training-testing data splitting, hyperparameter optimization, and model evaluation. Experimental results show that the proposed model achieves an accuracy of 0.96, precision of 0.95, recall of 0.96, F1-score of 0.96, and ROC-AUC of 0.9885. These findings indicate that the proposed framework effectively distinguishes malware from benign applications while maintaining computational efficiency and improving permission-based malware detection interpretability. The novelty of this study lies in the integration of Chi-Square feature selection and Grid Search-optimized LightGBM to develop a lightweight permission-based Android malware detection framework that reduces feature dimensionality while maintaining high detection performance and interpretability.

1. Introduction

In the rapidly evolving mobile ecosystem, the increasing use of Android-based smartphones has driven greater attention toward system security and performance. The surge in mobile application usage, fueled by changes in user behavior as well as global events such as the pandemic, has made mobile devices an essential

necessity and is expected to continu. [1]. Currently, Android is one of the most dominant and widely used operating systems on smartphones [2] . The Android operating system continues to dominate a wide range of devices and plays a highly significant role globally. With a smartphone market share exceeding 85%, Android has become a primary target for malware developers, prompting substantial attention from both industry and researchers toward its security aspects.[3][4].

As the number of Android users continues to grow, concerns about device security are also increasing, particularly regarding the infiltration of malicious software (malware). This situation makes Android security a significant challenge for practitioners and researchers in the field of cybersecurity [4] . The spread of malicious software on smartphones can lead to various serious consequences, including unauthorized access to users' personal data, monitoring of activity and geographic location, social media account breaches, misuse of banking accounts, unauthorized message sending, and degraded device performance, including reduced memory capacity and battery life[5].

Attacks targeting the Android platform include privacy breaches that occur when users grant risky (dangerous) permissions, allowing malicious applications to access sensitive data and personal information without the users' awarenes[6]. Permissions requested by applications, such as access to user location, phone information, and the network status of mobile devices [7] . Due to these security vulnerabilities, hackers can easily access personal data on mobile devices[8].

Malicious software, including Trojans, ransomware, spyware, and worms, often exploits users' limited understanding of the Android permission system, thereby posing a threat to the security and confidentiality of user data [9] . Permissions are a crucial feature in Android applications, enabling access to sensitive data and device resources. Permission analysis can reveal access patterns that do not align with an application's intended functionality, allowing apps with inappropriate permission requests to be identified as potential malware. Therefore, permission-based Android malware detection methods are receiving increasing attention [10] . User privacy is a primary concern in the use of mobile applications. There is no guarantee that these applications do not store or utilize user data for specific purposes through certain mechanisms or algorithms. The Android operating system implements a permission mechanism as an effort to enhance security and protect user data, allowing users to grant access to resources either at the time of installation or during application runtime. However, in practice, this permission system is often misused through requests for additional permissions that are not relevant to the core functionality of the application. As a result, some applications are designed to not function properly unless all requested permissions are granted by the user[11].

Machine learning has experienced rapid development in recent years and has had a significant impact across various fields, such as finance, medicine, and cybersecurity. In the context of malware detection, machine learning–based approaches have proven capable of achieving high levels of accuracy. However, limitations in terms of transparency remain a major challenge. Conventional machine learning models generally cannot provide easily interpretable explanations for the reasoning behind their predictions, which limits their applicability in high-security environments where understanding the decision-making process is crucial for supporting threat mitigation and response[12]. In recent years, various machine learning–based models have been extensively explored for detecting Android malware. However, selecting the most suitable model remains a challenge due to the wide range of factors that need to be considered[13].

Among various machine learning approaches for malware detection, LightGBM has gained considerable attention due to its efficiency and scalability. As a gradient boosting framework based on decision tree learning, LightGBM introduces optimization techniques such as Exclusive Feature Bundling (EFB) and Gradient-Based One-Side Sampling (GOSS) to reduce computational overhead while maintaining strong predictive capability. These characteristics enable the algorithm to effectively process sparse and high-dimensional feature spaces, which are frequently encountered in Android permission datasets. Owing to its favorable balance between accuracy, training efficiency, and scalability, LightGBM has been widely adopted in

cybersecurity applications, including malware classification, intrusion detection, and threat intelligence analysis[14].

Despite its strong predictive capability, the classification performance of LightGBM against complex and obfuscated malware can be influenced by its configuration settings. Therefore, appropriate hyperparameter optimization and model design are essential to improve decision boundary formation and maximize detection effectiveness [15] . Among the key factors affecting the performance of malware classification models, hyperparameter tuning plays a fundamental role in optimizing model behavior and improving classification accuracy across diverse threat scenarios[16].

Therefore, the model evaluation and selection process does not only focus on choosing the algorithm, but also on determining the appropriate parameter configuration. The process of finding the best combination of hyperparameters is known as hyperparameter tuning. Hyperparameters themselves are parameters that are set before the training process and are not learned directly from the data during training. In general, hyperparameter optimization is required to obtain the most optimal combination in order to improve the overall detection performance and resilience of machine learning models against zero-day variants [17].

In this study, prior to model testing, a Chi-square test was conducted as part of the filter-based feature engineering workflow. The Chi-square test is a nonparametric statistical method highly effective for identifying the statistical dependence between discrete categorical inputs and binary target classes. When handling expansive Android permission vectors, the Chi-square approach can be applied to rank features based on their independence scores. This method is effective in filtering out redundant features and is highly robust in situations involving imbalanced frequency distributions, which are highly prevalent in real-world malware-to-benign ratios [18].

2. Research Method

In this study, several research stages were carried out, starting from dataset collection, preprocessing, and statistical testing using the Chi-square method for feature selection. This was followed by data splitting into training and testing sets, as well as hyperparameter tuning to obtain the most optimal combination in order to improve the performance and effectiveness of the LightGBM model. Finally, the LightGBM approach was applied to the NATICUSdroid dataset.

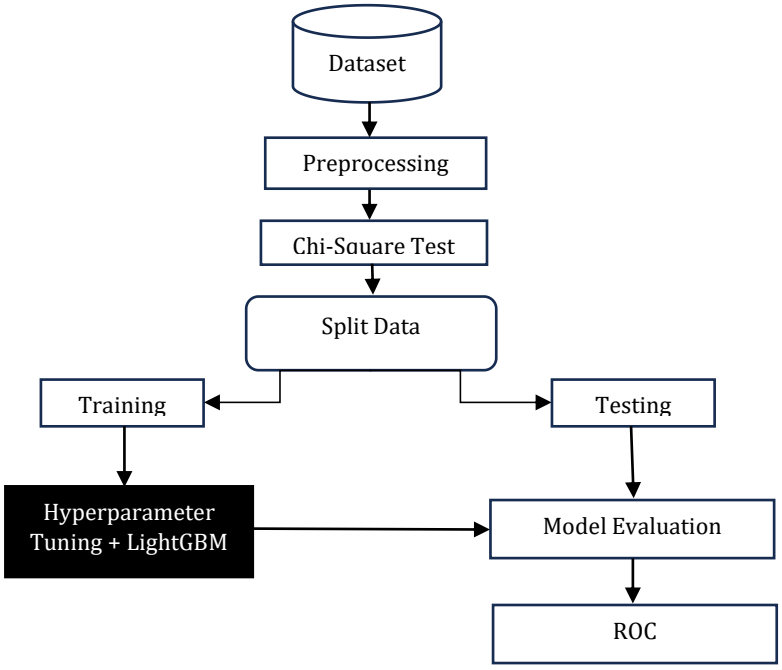

Figure 1. Research Framework

2.1 Dataset

This dataset consists of 29,300 Android applications, including 14,630 benign apps from the AndroZoo project and 14,700 malware samples from Argus Lab's Android Malware Database (AMD). The benign applications were obtained from the Google Play Store, while the malware samples were randomly selected from the AMD collection. Each application was analyzed by extracting permissions from the AndroidManifest.xml file, resulting in 86 features after the filtering process. Within this structural paradigm, each permission is represented in binary form (0) or (1) to distinguish between benign and malware applications, with an additional column serving as the target label for supervised classification[20].

2.2 Preprocessing

Data preprocessing is the initial stage performed before the model training process begins. Its main purpose is to ensure that the data is clean, consistent, and properly formatted for further analysis. The first step in this process is handling invalid data and missing values.

2.3 Feature Selection using Chi-Square Test

Feature selection aims to improve model performance by reducing the number of features used, thereby making the computational process more efficient. One commonly used method is the Chi-Square test. This method utilizes the Chi-Square statistic to measure the degree of dependency between each feature (term) and its class label, allowing the most relevant features to be selected. Using the following equation:

$$x^2 = \sum_{i=0}^{n} \frac{(O_1 - E_1)^2}{E_1} \qquad (1)$$

Description:
X^2 : Chi-square distribution
O_1 : Observed value
E_1 : Expected value

2.4 Training and Testing Data Split

The division of data into training and testing sets, Data partitioning is essential for preventing overfitting and minimizing bias in model selection. Typically, a dataset is divided into training, validation, and test subsets. The training set is used to fit the model, the validation set is utilized for hyperparameter optimization and model selection, and the test set provides an unbiased estimate of the model's performance on unseen data. Nevertheless, when the available dataset is limited, cross-validation techniques can be employed as an alternative to repeatedly split the data into training and validation folds. This approach enables a more robust evaluation of model performance while maximizing the use of available samples and reducing the likelihood of overfitting[19].

2.5 Hyperparameter Tuning

In gradient-boosting frameworks such as LightGBM, hyperparameter tuning plays a fundamental role in achieving optimal model performance. Hyperparameters, including learning rate, tree depth, and the number of leaves, are determined before the training process and significantly affect the model's learning behavior and predictive capability. Careful optimization of these settings is necessary to balance model complexity and generalization, thereby reducing the risk of overfitting or underfitting and improving performance on previously unseen data [20] . To optimize the performance of the LightGBM classifier, a Grid Search strategy was employed. Grid Search systematically evaluates all possible combinations of predefined hyperparameter values and selects the configuration that maximizes the chosen evaluation metric. The hyperparameter search space consisted of three parameters: the number of estimators (100 and 200), maximum tree depth (5, 10, and 15), and learning rate (0.01, 0.05, and 0.10). This resulted in a total of 18 hyperparameter combinations being evaluated. For each combination, stratified 10-fold cross-validation was performed to ensure robust performance estimation and preserve the class distribution within each fold. The average F1-score across the ten folds was used as the optimization criterion. The configuration yielding the highest mean F1-score was selected as the final model. This approach ensures that the selected hyperparameters provide a

balanced trade-off between precision and recall, which is particularly important in Android malware detection where both false positives and false negatives can significantly affect classification performance. The selection of LightGBM hyperparameters was guided by previous studies indicating that parameters such as num_leaves, max_depth, learning_rate, and n_estimators are among the most influential factors affecting model complexity, convergence behavior, and predictive accuracy [21].

2.6 LightGBM algorithm

LightGBM is a decision tree–based gradient boosting framework designed to improve the scalability and efficiency of traditional Gradient Boosting Decision Trees (GBDT). Since its introduction in 2017, the algorithm has been extensively adopted in both classification and regression applications due to its favorable balance between computational efficiency and predictive performance. To overcome the high computational cost associated with conventional GBDT models, LightGBM employs a histogram-based optimization strategy that groups continuous feature values into discrete bins. This approach substantially reduces the complexity of feature evaluation during tree construction, leading to faster training times and lower memory requirements, particularly when processing large and high-dimensional datasets [22].

In general, LightGBM is a fast, distributed, and high-performance gradient boosting framework based on decision tree algorithms. This algorithm can be applied to various machine learning tasks, such as ranking, classification, regression, and others. Assume a dataset defined as ($N = \{1, 2, ..., n\}$) and a LightGBM model consisting of ($T = \{1, 2, ..., t\}$) decision trees. After (t) iterations, the final prediction is obtained by summing the outputs of all previous trees up to the (t)-th iteration.

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_1(x_i) \quad (2)$$

Where $\hat{y}_i^{(t)}$ represents the predicted value of the (i)-th sample at iteration (t), $\hat{y}_i^{(t-1)}$ denotes the model output from the previous iteration, and $f_1(x_i)$ represents the newly built model. Based on Equation (1), each new prediction is formed from a combination of the residual and the previous prediction result. Furthermore, the overall training process is described in Equation (2). The regularization component, as shown in Equation (3), serves to reduce model complexity, thereby improving its generalization ability on other datasets.

$$\begin{cases} \hat{y}_i^{(o)} = 0 \\ \hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \end{cases} \quad (3)$$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_1, \hat{y}_1^{(t-1)} + f_t(x_1)) + \sum_{t=1}^{T} \Omega(fi) \quad (4)$$

$$\Omega(f) = \gamma^T + \frac{1}{2}\lambda||\omega||^2$$

Where (y_i) represents the actual value, y_i^t denotes the predicted value, and $\sum l$ represents the sum of the losses between each y_i and y_i^t.. Meanwhile, $\Omega(fi)$ refers to the regularization term. (T) is the number of leaves, (ω) represents the leaf weight, and (λ) is a coefficient, with default values set as ($\gamma = 0$) and ($\lambda = 1$). LightGBM has a characteristic that distinguishes it from other tree boosting methods, namely by splitting the tree in a leaf-wise manner (leaf-wise tree growth). Therefore, when growth is performed on the same leaf in LightGBM, the leaf-wise approach is able to reduce the loss value more effectively compared to the level-wise method. This allows LightGBM to achieve higher accuracy compared to other boosting algorithms.

2.7 Model Evaluation

The performance evaluation of the classification model is conducted using a confusion matrix. Through this method, the level of agreement between the predicted results and the actual data can be determined. In addition, various evaluation metrics such as MSE, recall, F1-score, and accuracy can also be calculated from the confusion matrix.

3. Result and Discussions

3.1 Dataset

In this study, the dataset used is a publicly available dataset obtained from Kaggle. The dataset contains 29,300 Android applications, consisting of 14,630 benign applications from the AndroZoo project and 14,700 malware samples from Argus Lab's Android Malware Database (AMD). Each application was processed by extracting permissions from the AndroidManifest.xml file, resulting in 86 permission-based features after the filtering stage. All permissions were represented in binary form (0 or 1), indicating the absence or presence of a specific permission within an application, with one additional column used as the target class label. However, the processed dataset does not provide detailed metadata such as malware family distribution, application collection period, or Android API version coverage. Therefore, these aspects could not be analyzed further in this study and may represent a limitation regarding dataset representativeness and model generalization assessment.

Table 1. Dataset Information

Dataset Information	Description
Total applications	29,300
Benign applications	14,630
Malware applications	14,700
Feature type	Android permissions
Number of features	86
Feature representation	Binary (0/1)
Benign source	AndroZoo
Malware source	AMD

As shown in Table 1, the dataset consists of 29,300 Android applications, including 14,630 benign applications and 14,700 malware applications, indicating a relatively balanced class distribution. Such balance is essential for reducing classification bias and improving the reliability of the machine learning model during training.

The benign application samples were obtained from the AndroZoo project, while the malware samples were collected from Argus Lab's Android Malware Database (AMD). The dataset uses Android permission attributes extracted from the AndroidManifest.xml file as the main feature type. After the preprocessing and filtering stages, a total of 86 permission-based features were retained for the classification process.

3.2 Model Testing

In this study, model testing was conducted using Google Colab. The process began with preprocessing to ensure that the data was clean, consistent, and properly formatted for further analysis. The first step in this process involved handling invalid data and missing values. Next, feature selection was performed using the Chi-square method, which aims to improve model performance by reducing the number of features used, thereby making the computational process more efficient.

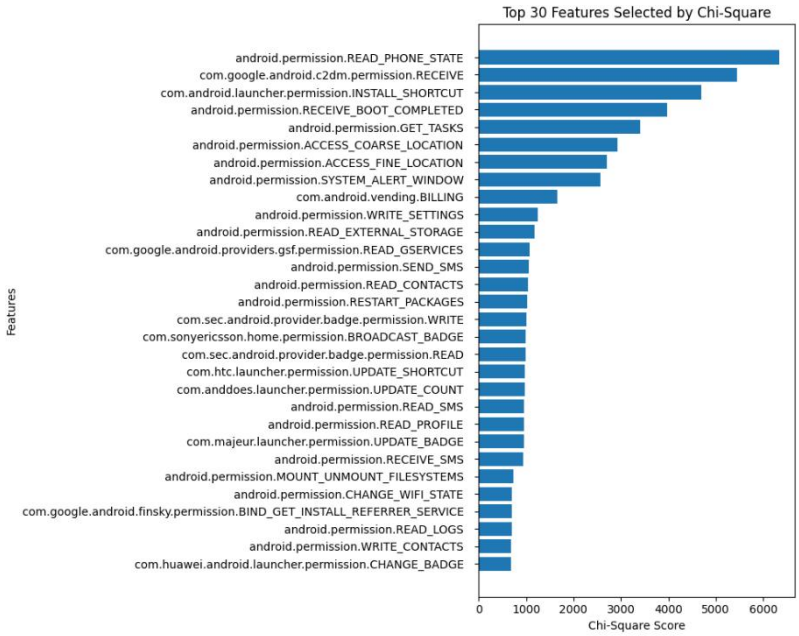

Figure 2. Chi-Square Ranking

Feature importance analysis is employed to determine the significance of individual features in the classification process. This approach facilitates the selection of the most discriminative attributes, leading to simpler models, reduced computational costs, and faster prediction times[23].

The results presented in Figure 2 indicate that READ_PHONE_STATE achieved the highest Chi-Square score among all selected permissions. The prominence of the READ_PHONE_STATE permission observed in this study is consistent with previous research reporting it as one of the most frequently requested permissions in malicious Android applications [24] . Since this permission provides access to sensitive device and user information, such as network status, device identifiers, phone numbers, and account information, it is often exploited by malware developers for information gathering and user tracking. Consequently, its high ranking in the feature importance analysis confirms its relevance as a key indicator for distinguishing malicious applications from benign ones.

The next stage involves the hyperparameter tuning process, which is conducted to control model complexity, bias level, and generalization ability. Improper selection of hyperparameter values can lead to overfitting or underfitting, which ultimately reduces model accuracy when applied to new or unseen data. By selecting an optimal combination of hyperparameters, the model is expected to achieve a balance between bias and variance, resulting in more stable and accurate performance.

Table 2. LightGBM Hyperparameter Configuration

Parameter	Value
learning_rate	0.05
n_estimators	200
max_depth	10
num_leaves	50
min_child_samples	20

The Table 2 LightGBM Hyperparameter Configuration presents the optimal hyperparameter configuration obtained from the hyperparameter tuning process for the LightGBM model. The tuning process aims to identify the most suitable parameter combination in order to improve classification performance and

enhance the model's generalization capability. The learning_rate parameter was set to 0.05, allowing the model to learn gradually and reducing the risk of overfitting during the boosting process. The number of boosting iterations, represented by n_estimators, was set to 200 to provide sufficient learning capacity while maintaining computational efficiency.

The max_depth parameter was configured with a value of 10 to control the maximum depth of each decision tree, preventing the model from becoming excessively complex. Meanwhile, num_leaves was set to 50, enabling the model to capture more detailed decision boundaries and improve classification accuracy. Additionally, the min_child_samples parameter was assigned a value of 20 to ensure that each leaf node contains a sufficient number of samples, thereby improving model stability and reducing the possibility of overfitting. Overall, this hyperparameter configuration contributed significantly to achieving high classification performance in Android malware detection.

After the hyperparameter tuning process is completed, the next step is data processing using the LightGBM algorithm to evaluate the results of the model testing, as follows:

Table 3. The results of data processing using the LightGBM method

Metric	Value
Accuracy	0.96
Precision	0.95
Recall	0.96
F1-Score	0.96
ROC-AUC	0.9885

Table 3 presents the performance evaluation results of the proposed LightGBM model for Android malware detection. The evaluation was conducted using several classification metrics, including Accuracy, Precision, Recall, F1-Score, and ROC-AUC, to comprehensively assess the effectiveness of the model. The model achieved an Accuracy value of 0.96, indicating that 96% of the applications were correctly classified as either benign or malware. The Precision score of 0.95 demonstrates that the model is highly reliable in identifying malware applications with a low false positive rate.

Similarly, the Recall value of 0.96 indicates that the model successfully detected most malware samples, reflecting its strong capability in minimizing false negatives. This is particularly important in cybersecurity applications, where undetected malware can pose serious security threats. The F1-Score of 0.96 shows a balanced performance between Precision and Recall, indicating that the model maintains both high detection capability and prediction reliability. Furthermore, the ROC-AUC score of 0.9885 demonstrates excellent classification performance and strong discriminative ability in distinguishing between malware and benign applications.

Overall, these results indicate that the proposed LightGBM model optimized with hyperparameter tuning and Chi-Square feature selection is highly effective for permission-based Android malware detection.

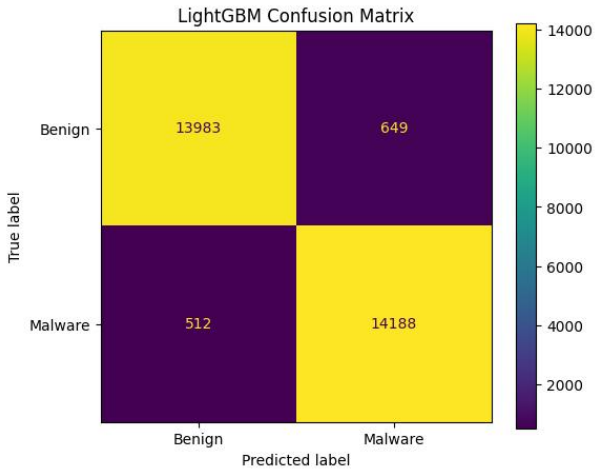

Figure 3 Confusion Matrix LightGBM

Based on the evaluation metrics obtained, the LightGBM model demonstrates an outstanding performance, achieving an AUC score of 0.9885 using only the 30 best features selected via the Chi-square method. The effectiveness of using only 31 selected Android features observed in this study is consistent with previous research demonstrating that a reduced feature space can maintain high malware detection performance while improving computational efficiency [25] [10].

The crucial distinction lies in the computational efficiency. While the MWHFST framework demands a prolonged training time due to its complex and repetitive wrapper-based loop processes, the approach proposed in this study utilizes a statistical filter (Chi-square) paired with the high-speed gradient-boosting capability of LightGBM.

Based on the confusion matrix results from testing using LightGBM, the LightGBM model optimized with hyperparameter tuning demonstrated very strong performance, achieving a ROC-AUC score of 0.9885. The recall value for the malware class reached 0.96, indicating that the model is capable of detecting most malicious applications with a low error rate. This shows that the LightGBM method is highly effective in handling permission-based datasets with binary features.

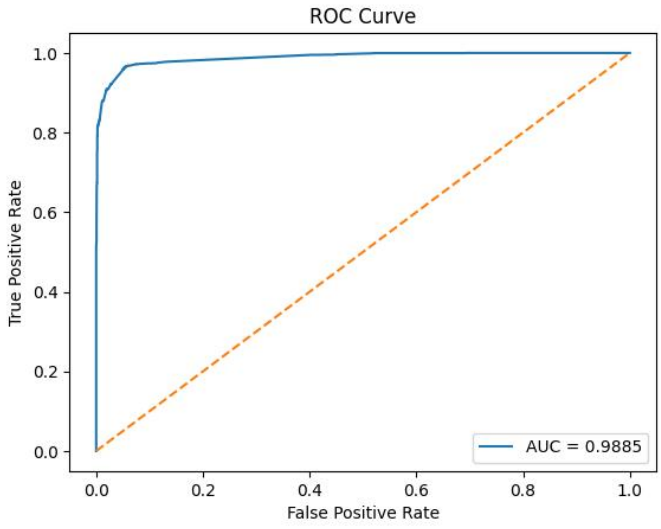

Figure 4. ROC - AUC Curve

Based on the evaluation results using the ROC Curve, the LightGBM model achieved an AUC value of 0.9885, indicating a very high classification ability in distinguishing between malware and non-malware applications. The ROC curve, which is close to the upper-left corner, indicates that the model has a high true positive rate with a low false positive rate, making it effective in detecting malware with minimal misclassification errors.

The approach proposed in this study demonstrates superior predictive sensitivity while simultaneously offering a significant computational advantage. Recent research has demonstrated that reducing the feature space through the Multi-Wrapper Hybrid Feature Selection Technique (MWHFST) can achieve a peak classification accuracy of 98.85% on the large-scale Kronodroid dataset using only 31 pivotal features [26]. The model proposed in this research strongly supports and elevates this benchmark, achieving a highly comparable performance (an AUC of 0.9885) using an even leaner data representation of only 30 features selected through a single-stage Chi-square statistical filter.

To assess the effectiveness of LightGBM for Android malware detection, comparative experiments were conducted using Random Forest and Logistic Regression as benchmark classifiers. The performance of each model was evaluated based on Accuracy, Precision, Recall, F1-Score, and AUC metrics, with the comparative results illustrated in Figure 5.

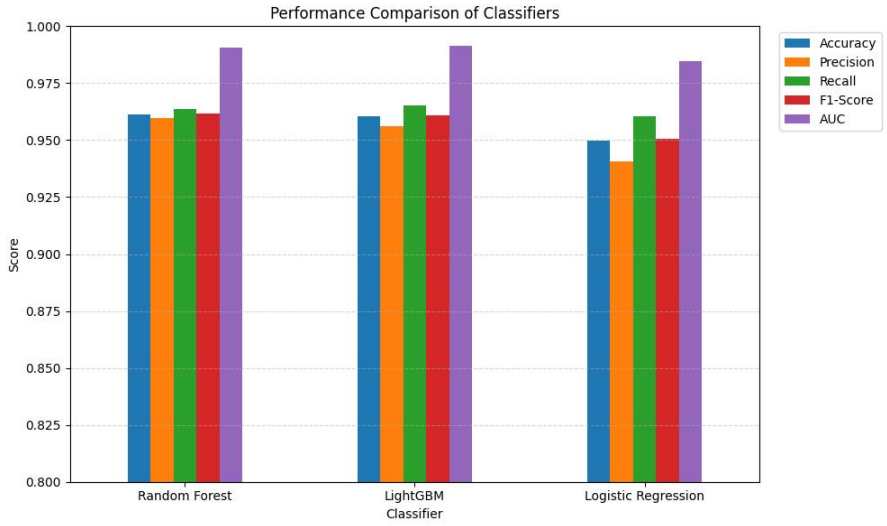

Figure 5. Comparison image of Random Forest, LightGBM and Logistic Regression Models

Figure 5 presents the performance comparison of three classification algorithms, namely Random Forest, LightGBM, and Logistic Regression, based on five evaluation metrics: Accuracy, Precision, Recall, F1-Score, and Area Under the Curve (AUC). Overall, both Random Forest and LightGBM demonstrated superior performance compared to Logistic Regression across all evaluation metrics.

LightGBM achieved the highest overall performance, obtaining an accuracy of approximately 0.96, a precision of 0.95, a recall of 0.96, an F1-score of 0.96, and the highest AUC value of 0.9885. Similarly, Random Forest produced competitive results, with an accuracy of 0.96, precision of 0.95, recall of 0.96, F1-score of 0.96, and an AUC of 0.98. The minimal difference between these two ensemble-based methods indicates their strong capability in distinguishing malware from benign Android applications. In contrast, Logistic Regression exhibited comparatively lower performance, achieving an accuracy of 0.950, precision of 0.94, recall of 0.96, F1-score of 0.95, and an AUC of 0.979. Although its performance remained relatively high, the lower precision and AUC values suggest a reduced ability to separate malware and benign samples when compared with the ensemble learning approaches.

The results indicate that ensemble-based classifiers, particularly LightGBM, provide more effective predictive performance for Android malware detection. The superior AUC score achieved by LightGBM demonstrates its

stronger discriminative capability, while its balanced precision, recall, and F1-score indicate reliable classification performance with a low false-positive rate and a high malware detection rate.

4. Conclusions and Future Works

This study investigated the effectiveness of combining Chi-square feature selection and LightGBM with hyperparameter tuning for Android malware detection. The experimental results demonstrate that the proposed framework achieves excellent classification performance, with an accuracy of 0.96,Pecission of 0.95, recall of 0.96, F1-score of 0.96, and ROC-AUC of 0.9885. Notably, using only the top 30 permission features selected through the Chi-square method, the model achieved an AUC score of 0.9885, indicating that a reduced set of highly informative permissions can effectively distinguish malware from benign applications while improving computational efficiency. Furthermore, LightGBM outperformed Logistic Regression and demonstrated competitive performance compared to Random Forest, highlighting its suitability for permission-based Android malware detection.

The main contribution of this study lies in demonstrating that the integration of filter-based feature selection and optimized gradient boosting can provide a lightweight yet highly accurate malware detection framework. By reducing feature dimensionality while maintaining strong predictive performance, the proposed approach offers a practical solution for resource-efficient mobile security applications. However, several limitations should be acknowledged. First, the reported performance is dependent on the characteristics of the dataset used and may not directly generalize to heterogeneous real-world environments. Second, the model relies exclusively on static permission-based features, limiting its ability to detect advanced malware that employs obfuscation, reflection, or dynamic code-loading techniques. Third, the continuously evolving nature of Android malware may reduce the effectiveness of permission-based patterns when encountering previously unseen zero-day threats.

Future research should focus on evaluating the proposed framework using larger and more diverse benchmark datasets to assess its generalization capability. In addition, incorporating dynamic behavioral features, such as API call sequences, system logs, and network traffic patterns, may improve detection performance against sophisticated malware variants. The application of robust validation strategies, including k-fold cross-validation, as well as the investigation of data imbalance handling techniques and advanced feature engineering methods, may further enhance model reliability. Finally, testing the framework on recent real-world Android applications would provide valuable insights into its practical effectiveness in operational mobile security environments.

5. References

[1] A. L. Chandran, J. Samual, S. Safavi, and A. Ali, "A Comparative Analysis of Machine Learning Models on Detecting Malware in Android Devices," *Journal of Cyber Security and Risk Auditing*, vol. 2025, no. 4, pp. 327–346, 2025, doi: 10.63180/jc sra.thestap.2025.4.10

[2] B. Urooj, M. A. L. I. Shah, C. Maple, M. K. Abbasi, and S. Riasat, "Malware Detection : A Framework for Reverse Engineered Android Applications Through Machine Learning Algorithms," *Computer Security and Reliability*, vol. 10, no. 1, pp. 89031–89050, 2022.

[3] C. C. Obidiagha and M. Rahouti, "DeepImageDroid : A Hybrid Framework Leveraging Visual Transformers and Convolutional Neural Networks for Robust Android Malware Detection," *International Journal of Digital Crime and Forensics*, vol. 12, no. November, pp. 156285–156306, 2024, doi: 10.1109/ACCESS.2024.3485593.

[4] H. Rafiq, N. Aslam, M. Aleem, and B. Issac, "AndroMalPack : enhancing the ML - based malware classification by detection and removal of repacked apps for Android systems," *Scientific Reports*, pp. 1–18, 2022, doi: 10.1038/s41598-022-23766-w.

[5] S. Bulut and A. Korkmaz, "Comparative Analysis of Machine Learning Models for Android Malware Detection Selma," *Sakarya University Journal of Science*, vol. 28, no. 3, pp. 517–530, 2024, doi: 10.1016/j.procs.2023.03.101.

[6] A. Muzaffar, H. Ragab, M. A. Lones, and H. Zantout, "Computers & Security An in-depth review of machine learning based Android malware detection," vol. 121, 2022.

[7] J. Kim, Y. Ban, E. Ko, and H. Cho, "MAPAS : a practical deep learning-based android malware detection system," *International Journal of Information Security*, vol. 21, no. 4, pp. 725–738, 2022, doi: 10.1007/s10207-022-00579-6.

[8] S. Poornima and R. Mahalakshmi, "Automated malware detection using machine learning and deep learning approaches for android applications," *Measurement: Sensors journal*, vol. 32, no. May 2023, pp. 1–8, 2024, doi: 10.1016/j.measen.2023.100955.

[9] M. Khalid, A. Sajid, M. Usman, M. Saeed, M. M. Nadeem, and I. Sharma, "Android Security Vulnerabilities, Malware, Anti-Malware Solutions, and Evasion Techniques," vol. 4, no. 4, pp. 129–145, 2024.

[10] A. Banik and J. P. Singh, "Android Malware Detection by Correlated Real Permission Couples Using FP Growth Algorithm and Neural Networks," *IEEE Access*, vol. 11, no. October, pp. 124996–125010, 2023, doi: 10.1109/ACCESS.2023.3323845.

[11] B. Mishra, A. Agarwal, A. Goel, D. Singh, and H. Lee, "Privacy Protection Framework for Android," *INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT*, vol. 10, no. 2, pp. 7973–7988, 2022, doi: 10.1109/ACCESS.2022.3142345.

[12] H. Manthena, S. Shajarian, and J. C. Kimmell, "Explainable Artificial Intelligence (XAI) for Malware Analysis : A Survey of Techniques , Applications , and Open Challenges," *IEEE Access*, vol. 13, no. February, pp. 61611–61640, 2025, doi: 10.1109/ACCESS.2025.3555926.

[13] H. Alomari, Q. M. Yaseen, M. A. Al-betar, and H. Alomari, "A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection," *Procedia Computer Science*, vol. 2020, no. 2019, pp. 763–768, 2023, doi: 10.1016/j.procs.2023.03.101.

[14] S. Zhou, H. Li, X. Fu, D. Han, and X. He, "Novel Multi-Classification Dynamic Detection Model for Android Malware Based on Improved Zebra Optimization," *Sensors*, vol. 24, no. 18, pp. 1–30, 2024, doi: 10.3390/s24185975.

[15] M. Dhalaria and E. Gandotra, "A Hybrid Approach for Android Malware Detection and Family Classification," vol. 6, 2020, doi: 10.9781/ijimai.2020.09.001.

[16] E. Kavalcı Yılmaz and H. Bakır, "Hyperparameter Tunning and Feature Selection Methods for Malware Detection," *Politeknik Dergisi (Journal of Polytechnic)*, vol. 27, no. 1, pp. 343–353, 2024, doi: 10.2339/politeknik.1243881.

[17] Y. Sharma and A. Arora, "IPAnalyzer: A novel Android malware detection system using ranked Intents and Permissions," *Multimedia Tools and Applications*, vol. 83, no. 33, pp. 78957–79008, 2024, doi: 10.1007/s11042-024-18511-6.

[18] G. Ahn, K. Kim, and W. Park, "applied sciences Malicious File Detection Method Using Machine Learning and Interworking with MITRE ATT & CK Framework," *Applied Sciences*, vol. 12, no. 21, pp. 1–22, 2022, doi: 10.3390/app122110761.

[19] V. R. Joseph, "Optimal ratio for data splitting," *Statistical Analysis and Data Mining*, vol. 15, no. 4, pp. 531–538, 2022, doi: 10.1002/sam.11583.

[20] W. Pannakkong, K. Thiwa-anont, K. Singthong, P. Parthanadee, and J. Buddhakulsomsiri, "Hyperparameter Tuning of Machine Learning Algorithms Using Response Surface Methodology : A Case Study of ANN , SVM , and DBN," vol. 2022, 2022, doi: 10.1155/2022/8513719.

[21] A. Open, A. Journal, S. Samantaray, S. Mohapatra, and M. Mishra, "High-precision forecasting of Indian stock market indices using weighted ensemble of hyperparameter-tuned LightGBM models with diverse technical indicators," vol. 2583, 2025, doi: 10.1080/21642583.2025.2567887.

[22] D. Demirtürk, Ö. Mintemur, and A. Arslan, "Optimizing LightGBM and XGBoost Algorithms for Estimating Compressive Strength in High-Performance Concrete," *Arabian Journal for Science and Engineering*, vol. 51, no. 4, pp. 4401–4423, 2026, doi: 10.1007/s13369-025-10217-7.

[23] K.-H. J. Fahad Akbar, Mehdi Hussain, Rafia Mumtaz, Qaiser Riaz, Ainuddin Wahid Abdul Wahab, "Permissions-Based Detection of Android Malware Using Machine Learning," *Symmetry*, vol. 14, no. 2, pp. 1–9, 2022, doi: 10.3390/sym14040718.

[24] J. Mohamad, A. Id, M. Faizal, A. Razak, S. Awang, and S. R. Tuan, "A static analysis approach for Android permission-based malware detection systems," *PLOS ONE*, vol. 16, no. 9, pp. 1–23, 2021, doi: 10.1371/journal.pone.0257968.

[25] Y. Wu, M. Li, Q. Zeng, T. Yang, J. Wang, Z. Fang, and L. Cheng, "DroidRL: Feature selection for android malware detection with reinforcement learning," *Computers & Security*, vol. 128, p. 103126, 2023, doi: 10.1016/j.cose.2023.103126.

[26] S. Sharma, R. Chhikara, and K. Khanna, "A novel feature selection technique : Detection and classification of Android malware," *Egyptian Informatics Journal*, vol. 29, no. December 2024, p. 100618, 2025, doi: 10.1016/j.eij.2025.100618.