Smart Technologies Academic Press

Journal of Cyber Security and Risk Auditing

https://www.jcsra.thestap.com/

A Comparative Analysis of Machine Learning Models on Detecting Malware in Android Devices

Alagesh Leroy Chandran[1], Joshua Samual[1], Seyedmostafa Safavi[1], Aitizaz Ali[1]

[1] *School of Technology, Asia Pacific University of Technology and Innovations, Kuala Lumpur, Malaysia*

ARTICLE INFO

Article History

Received:	**17-05-2025**
Revised:	**30-07-2025**
Accepted:	**03-09-2025**
Published:	**07-09-2025**

Vol.2025, No.4

DOI:

Corresponding author. Email: *aitizaz.ali@apu.edu.my*

Orcid:

https://orcid.org/00000002-4853-5093

ABSTRACT

Given the recent increase in cyberattacks, malware detection remains a critical component in Android devices. Traditional signature-based methods, while effective against known types of malware, highlight the need for more advanced techniques such as machine learning. This study provides a detailed comparison of various machine learning methods for Android malware detection, focusing on their effectiveness and limitations. We evaluate different models, including logistic regression, support vector machines, random forests, and XGBoost, to determine their efficacy in Android malware detection. Through comprehensive experiments, we assess the models based on parameters such as accuracy, precision, recall, and false positive rate. The results reveal clear advantages and disadvantages among the different machine learning algorithms, offering significant insights into their practical applications. This paper underscores the potential of machine learning algorithms to enhance malware detection in Android while highlighting key areas for further research and improvement. Our findings support the continuous development of robust and adaptable cybersecurity solutions in the Android environment, emphasizing the critical role of machine learning in defending against evolving malware threats.

Keywords: Android malware detection, Machine learning, Malware detection techniques, Logistic Regression, Support vector machines, Random forests, XGBoost, Signature-based methods, Practical applications of machine learning

How to cite the article

1. Introduction

In the rapidly evolving mobile landscape, the proliferation of Android smartphones has heightened the focus on the security and performance of these systems. The exponential rise in mobile app usage, driven by shifting user behaviors and global events such as pandemics, has made mobile devices indispensable, a trend that is expected to continue (1). However, this digital expansion has also led to a significant increase in malware threats, posing critical risks to the efficiency, confidentiality, and security of data on Android devices (2).

Android's open-source nature, while fostering innovation, also makes it an attractive target for cybercriminals (3). Android's openness allows for extensive program modifications, which can inadvertently introduce vulnerabilities that malicious actors may exploit (4). This vulnerability underscores the urgent need for studyers and developers to deepen their understanding of security threats facing Android smartphones and to devise robust detection strategies (5). ML is particularly well-suited for this task as it enables the creation of models capable of identifying malware with minimal training data, without relying on predefined signatures (6).

The primary objectives of this study are to identify the most effective ML model for malware detection, enhance its performance, and mitigate threats on Android platforms through static analysis. The study will particularly focus on critical attributes such as the permissions declared within the AndroidManifest.xml file to uncover potential security weaknesses. By utilizing a range of classification and clustering algorithms, this study systematically evaluates various machine learning approaches using the NATICUSdroid dataset. This dataset is specifically focused on Android permissions, including both native and custom permissions, which are critical in identifying potential vulnerabilities within Android application. A comprehensive evaluation of base algorithms, including Random Forest, Logistic Regression, XGBoost, and Support Vector Machine, will be conducted to identify the top-performing model. This study aims to lay a solid foundation for a robust malware detection system by rigorously assessing these models using the NATICUSdroid dataset.

In this study, we will explore various methodologies and findings related to Android malware detection, with a focus on the application of machine learning techniques. Section 5 provides an overview of related works, discussing several studies and their methodologies, highlighting the progression of machine learning approaches in the detection of Android malware. In Section 6, we delve into specific machine learning algorithms, including Random Forest (RF), Support Vector Machine (SVM), XGBoost, and Logistic Regression. Each algorithm's functionality and suitability for malware detection will be thoroughly explained, illustrating their unique capabilities in identifying malicious behaviors. Section 7 details the methodology of our study, beginning with a comprehensive description of the NATICUSdroid dataset, outlining the sources and preprocessing steps involved to prepare the data for analysis.Section 8 presents the results of our analysis, including the evaluation metrics used to assess model performance, such as accuracy, precision, recall, F1-score, and ROC-AUC. Performance results for different train-test splits are provided to offer a comprehensive view of each model's capabilities under varying conditions. In Section 9, we discuss our findings in-depth, particularly focusing on the top-performing model. Finally, Section 10 concludes the study by summarizing our key findings, emphasizing the effectiveness of model, and discussing its adaptability and reliability in real-world applications. We also provide recommendations for future study directions, aiming to further refine machine learning techniques in the domain of Android cybersecurity. In this study, we will explore various methodologies and findings related to Android malware detection, with a focus on the application of machine learning techniques.

2. Related works

In this section we will explore various methodologies and findings related to Android malware detection Malware, or malicious software, is designed to exploit vulnerabilities in computer systems, breaching security principles such as confidentiality, integrity, and availability. Android malware specifically targets devices running the Android operating system, aiming to steal information or cause damage. This category of malware includes various types, such as Trojans, spyware, adware, ransomware, worms, botnets, and backdoors. Additionally, app collusion, where multiple apps collaborate for malicious purposes, is a growing concern. Despite Android's built-in security features, vulnerabilities persist due to design flaws and security loopholes. Understanding these risks is crucial for accurately identifying and assessing malware threats. Android devices are susceptible to a range of attacks, including hardware-based, kernel-based, HAL-based, and application-based attacks (7).Without robust security measures, Android applications can be easily compromised by attackers familiar with the platform. User errors and developer oversights further increase risks. Users may unknowingly grant excessive permissions to apps, allowing malicious actions, while developers might inadvertently introduce vulnerabilities into their apps. Despite existing guidelines and recommendations, some developers fail to write secure code for their applications (8).

(Liu et al., 2020) Machine learning (ML), a branch of artificial intelligence, plays a pivotal role in malware detection by using data and algorithms to mimic human learning processes with high accuracy. Traditional methods rely on analyzing past data to make predictions. The machine learning lifecycle includes several key phases: data preprocessing, feature selection, model training, model evaluation, model deployment, and data extraction. Studyers are focused on improving conventional algorithms' performance by selecting appropriate ML algorithms to enhance detection efficiency. These models can distinguish between benign and malicious Android applications, aiding in the identification of both known malware variants and entirely new threats, positioning ML techniques at the forefront of this challenge.

(Karbab et al., 2018) MalDozer, a deep learning-based system, excels in detecting Android malware by analyzing API method call sequences. It is particularly effective in attributing malware to specific families and operates efficiently across various hardware platforms. However, concerns remain about the dataset quality and the comprehensiveness of its detection methods. MalDozer's adaptability to IoT devices and its minimal preprocessing requirements are notable advantages. It achieves an F1 Score ranging from 96% to 99% in malware detection and 96% to 98% in family attribution, demonstrating exceptional precision and recall. Its framework is versatile enough for large-scale deployment, and it automatically extracts features from raw API method calls during training, reducing the need for manual intervention. However, its reliance on DEX bytecode for core functionalities may overlook malicious activities in native code or web-based apps.

Roy et al., (2020) A method that maps API calls to features and aggregates them to determine their frequency, followed by a machine learning evaluation using a dataset comprising obfuscated and benign applications. The method achieves a high ROC AUC score, demonstrating its efficacy in malware detection. This novel integration of feature aggregation and machine learning yields high accuracy, even when the feature set is reduced by 75.9%. However, the preprocessing and feature extraction phases may demand substantial computational resources, potentially limiting deployment on low-end systems.

(Shao et al., 2021) FB2Droid addresses the imbalance in malware sample distribution by leveraging family-based information, potentially enhancing detection accuracy. The use of the relief feature selection algorithm to choose relevant features from APK files contributes to the efficiency of the detection process. The paper introduces two new sampling strategies to mitigate class imbalance, which could improve classifier performance on minority class samples. Enhancements in detection accuracy and F score by 2.3% and 2.0%, respectively, are reported. However, the validity of these findings depends on the robustness of the experimental design and the datasets employed. FB2Droid's heavy reliance on accurate malware family information, which might not always be available or correctly labeled, is a potential limitation.

(NDSS Symposium, 2024) DREBIN performs extensive static analysis to extract numerous features from applications, facilitating malware identification without runtime monitoring. It achieves an impressive 94% detection rate with minimal false alarms, although its efficacy in real-world scenarios may vary. DREBIN excels in providing explanations for its detections, helping users understand the rationale behind alerts. Its resource efficiency, with an average analysis time of about 10 seconds, makes it suitable for smartphone use. However, DREBIN's reliance on static analysis might render it ineffective against malware employing dynamic code execution or obfuscation techniques.

(Mahindru & Sangal, 2020) The extraction of dynamic permissions from Android applications is crucial for detecting malware, especially given the dynamic nature of Android security. This approach is essential for identifying malware that employs runtime behaviors to evade static analysis. Evaluating several classifiers, including Naive Bayes, Decision Tree (J48), Random Forest, Simple Logistic, and K star, found that Simple Logistic slightly outperformed others in terms of malware classification accuracy.

(Urooj et al., 2022) A novel model combining innovative static feature sets with extensive malware sample datasets marks a significant advancement over traditional methods, achieving a 96.24% accuracy rate with a low false positive rate of 0.3%. However, the model's handling of unnecessary permissions requested by Android applications remains unspecified, potentially impacting its effectiveness. The use of ensemble learning techniques, particularly with algorithms like AdaBoost and Support Vector Machine (SVM), enhances the model's performance in detecting malware, demonstrating a strong understanding of advanced ML techniques.

(Fallah & Bidgoly, 2019) Given the prevalence of Android devices and the corresponding malware threats, cybersecurity becomes a critical concern. A comparative analysis of different algorithms is essential to determine their effectiveness in malware detection. This benchmarking process requires a comprehensive dataset of Android applications and performance metrics to assess the algorithms' accuracy, speed, and reliability. The findings could significantly improve Android security and protect users from malicious software. (Gautam et al., 2023) A significant cybersecurity issue, particularly in Android malware detection, remains highly relevant due to the widespread use of Android devices. Static analysis of app manifests, as employed in some studies, extracts features to identify potentially malicious behavior. However, the need for dynamic and hybrid analyses is emphasized for future study, as current methods may not fully capture the complexities of malware behaviors applied Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) algorithms for classification tasks, achieving average accuracy rates of 79.08% for SVM and 80.50% for KNN. While promising, the relatively low true positive rate for SVM raises concerns for practical applications, where missing malicious apps could have significant consequences. (Mahindru & Sangal, 2020) MLDroid stands out as a web-based solution for dynamic analysis to identify malware within Android apps. By leveraging machine learning algorithms, it thoroughly examines Android application packages (.apk files). The framework's feature selection methodologies pinpoint significant features from vast datasets, which are then utilized to train a range of machine learning models. MLDroid achieves a robust detection rate of 98.8% for malware within real-world apps, demonstrating its effectiveness in flagging malicious software. The framework's incorporation of multiple machine learning techniques, including deep learning, clustering, and decision tree approaches, reflects a holistic and resilient strategy for malware detection. However, there is concern about potential overfitting to the training data, which might impact the framework's adaptability to new, unseen malware instances. (Lee et al., 2021) The use of genetic algorithms for feature selection in machine learning models presents a promising avenue for enhancing Android malware detection efficiency. By prioritizing the most relevant features, this approach can improve the speed and accuracy of machine learning algorithms. However, to ensure its effectiveness and address potential limitations, it is crucial to compare it with alternative feature selection techniques. Real-world applicability must be assessed through testing across diverse datasets and scenarios to validate the method's reliability and robustness conducted a thorough comparison of nine distinct machine learning algorithms, revealing that genetic algorithm-based feature selection outperforms the commonly used information gain-based technique. However, the extensive feature selection performed by the genetic algorithm requires further verification to establish the generalizability of the results. (Azeem, 2024) Machine learning approaches have significantly improved malware detection, offering varying levels of accuracy, computational efficiency, and ease of implementation. Random Forest (RF) and Convolutional Neural Networks (CNN) typically achieve the highest accuracy, often exceeding 97%, due to their robustness and ability to capture complex patterns. However, they require substantial computational resources and are slower in prediction. Support Vector Machines (SVM) also perform well, particularly in high-dimensional spaces, but need careful tuning and are less scalable for large datasets. Simpler models like Logistic Regression (LR) and K-Nearest Neighbors (KNN) are easier to implement and computationally less intensive, making them suitable for smaller datasets or initial experiments, though they generally offer lower accuracy. Neural Network Multilayer Perceptron (nnMLP) models, while demanding in terms of resources, provide accuracy comparable to CNNs. The choice of model should balance accuracy, computational demands, and the specific requirements of the task. (Seoungyul, 2019) Low-dimensional features combined with tree-based ensemble models provide an efficient and accurate approach to malware detection, addressing the challenges of high-dimensional data and computational inefficiencies. Among the ensemble models, XGBoost consistently delivers the highest accuracy and scalability, making it ideal for large datasets, while Random Forest and Extra Trees offer a balance of computational efficiency and robustness. AdaBoost enhances accuracy by focusing on difficult-to-classify instances but can be slower in training compared to XGBoost. Feature reduction through techniques like Window Entropy Map (WEM), API Call Frequencies, and Opcode Sequences helps streamline the detection process, maintaining high.

Table 1. Related works

Author	Key Focus	Techniques/Algorithms	Performance/Outcome	Limitations
Karbab et al., 2018	Android malware detection through API call sequence analysis	Deep learning based system	F1 Score: 96 99% for detection; 96 98% for family attribution. Adaptable to IoT, minimal preprocessing	Relies on DEX bytecode, may overlook malicious activities in native code or web based apps

Roy et al., 2020	Malware detection using feature mapping of API calls	Machine learning evaluation using ROC AUC score	High accuracy with 75.9% reduction in feature set	High computational resource demand for preprocessing and feature extraction
Shao, 2021	Addressing malware sample distribution imbalance	Relief feature selection, family based information	2.3% increase in detection accuracy; 2.0% increase in F score	Relies heavily on accurate malware family information, which may not always be available or correctly labelled
Daoudi et al., 2022	Static analysis for malware identification	Machine learning, static analysis	94% detection rate, minimal false alarms, resource efficient (10 seconds average analysis time)	May be ineffective against malware using dynamic code execution or obfuscation techniques
Mahindru & Sangal, 2020	Dynamic permission extraction for malware detection	Naive Bayes, Decision Tree, Random Forest, Simple Logistic, K star	Simple Logistic slightly outperforms other classifiers in malware classification accuracy	Not explicitly mentioned
Urooj et al., 2022	Innovative static feature sets for malware detection	Ensemble learning, AdaBoost, SVM	96.24% accuracy rate, 0.3% false positive rate	Handling of unnecessary permissions requested by apps remains unspecified
Fallah & Bidgoly, 2019	Comparative analysis of malware detection algorithms	Various machine learning algorithms	Highlights relative performance of different algorithms	General benchmarking of algorithms, specific outcomes not detailed
Gautam et al., 2023	Static analysis of app manifests	Support Vector Machine (SVM), K Nearest Neighbors (KNN)	79.08% accuracy for SVM; 80.50% for KNN. Raises concerns with the low true positive rate for SVM	Potential risk of missing malicious apps due to lower true positive rate for SVM
Mahindru & Sangal, 2020	Dynamic analysis for Android malware detection	Machine learning (deep learning, clustering, decision tree)	98.8% detection rate for real world apps, holistic strategy	Potential overfitting to training data, may impact adaptability to new, unseen malware
Lee et al., 2021	Genetic algorithms for feature	Genetic algorithm based feature selection	Outperforms information gain based technique, thorough comparison of	Extensive feature selection requires further

	selection in ML models		nine distinct ML algorithms	verification for generalizability
Azeem, 2024	Machine learning approaches in malware detection	Random Forest (RF), Convolutional Neural Networks (CNN), Support Vector Machines (SVM), Logistic Regression (LR)	RF and CNN achieve highest accuracy (>97%); SVM performs well in high-dimensional spaces	RF and CNN require substantial computational resources
Seoungyul, 2020	Low-dimensional features with tree-based ensemble models	XGBoost, Random Forest, Extra Trees, AdaBoost	XGBoost delivers highest accuracy and scalability; feature reduction maintains performance while reducing computational load	AdaBoost slower in training compared to XGBoost
Juwono, 2022	Behavior analysis sandboxes in malware detection	Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), k-Nearest Neighbors (kNN)	RF most effective with Cuckoo datasets (Recall: 96.48%, Precision: 98.23%, F-Measure: 97.35%); Cuckoo combined with RF top choice for malware detection	SVM requires more computational resources
Syuhada, 2019	Malware detection using machine learning algorithms	K-Nearest Neighbors (K-NN), Decision Tree (DT), Support Vector Machine (SVM)	DT achieves highest detection accuracy (99%) with low FPR (0.021%)	Small dataset; results may not generalize well to larger or more diverse datasets

3. Machine Learning Algorithms

This section provides a detailed overview of the machine learning algorithms employed in our experiment, each of which brings distinct characteristics and strengths that render them particularly suitable for different facets of malware detection on Android platforms. These algorithms were chosen for their proven efficacy in handling the complexities inherent in malware classification, especially in environments with diverse and high-dimensional data (*IBM*, n.d.).

3.1 Random Forest

Random Forest is an ensemble learning method that constructs a multitude of decision trees during the training process and outputs either the mode of the classes (in classification tasks) or the mean prediction (in regression tasks) of the individual trees. This approach is particularly advantageous in scenarios involving high-dimensional datasets with many features, such as those encountered in malware detection.

The strength of Random Forest lies in its ability to reduce overfitting through the aggregation of multiple decision trees, each trained on a random subset of the data. This ensemble approach ensures that the model generalizes well to unseen data, making it robust against the noise and variability often present in large, complex datasets. Moreover, Random Forest's capacity to capture intricate interactions between features enhances its effectiveness in identifying subtle patterns that may distinguish between benign and malicious applications (Donges, 2024).

3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm widely recognized for its effectiveness in classification tasks, particularly in high-dimensional spaces. SVM operates by identifying the hyperplane that best separates the data into distinct classes, maximizing the margin between different classes (e.g., malware vs. benign). In the context of malware detection, SVM is particularly suited for scenarios where the boundary between classes is clear but complex. By leveraging kernel functions, SVM can map input data into higher-dimensional spaces, enabling the model to construct non-linear decision boundaries that can better capture the nuances in the data. This makes SVM a powerful tool for distinguishing between benign and malicious software in environments where the data exhibits complex relationships (Abdullah & Abdulazeez, 2021)

3.3 XGBoost

XGBoost, or Extreme Gradient Boosting, is an advanced decision tree-based ensemble algorithm that has gained widespread popularity due to its superior speed and performance in various machine learning competitions. XGBoost builds on the concept of boosting, where weak learners (typically decision trees) are sequentially trained to correct the errors made by the previous models, leading to a strong overall model. In malware detection, XGBoost's ability to handle large-scale datasets and its efficiency in focusing on the most informative features make it particularly effective. The iterative nature of boosting allows XGBoost to refine its predictions by learning from the mistakes of prior iterations, which is especially beneficial when dealing with imbalanced datasets where the occurrence of malware is much less frequent than benign software.

3.4 Logistic Regression

Logistic Regression is a foundational binary classification algorithm that models the probability of a binary outcome based on one or more predictor variables. Despite its simplicity, Logistic Regression is remarkably effective in scenarios where the relationship between the features and the target variable is linear or approximately linear. In the context of malware detection, Logistic Regression serves as an essential baseline model due to its interpretability and computational efficiency. It is particularly useful in cases where the data is large and the relationship between features and the outcome is straightforward. The coefficients produced by Logistic Regression are easily interpretable, providing clear insights into the influence of each feature on the probability of an application being classified as malware.

4. Methodology

In our experiment, we utilized the NATICUSdroid framework, which focuses on detecting Android malware by analysing both native and custom permissions (*UCI Machine Learning Repository*, n.d.). The dataset comprises a comprehensive collection of Android applications, both benign and malicious, sourced from multiple reputable repositories.

4.1 Data Sources

Benign Applications: The benign dataset was created using applications available from the Androzoo project database, which includes metadata for over 10 million Android apps collected from various app markets, including the Google Play Store. For the benign dataset, 15,000 apps rated as benign by VirusTotal were selected and further pruned to 14,630 apps based on a target SDK version of API level 23 and above (Mathur et al., 2021). Malicious Applications: The malicious dataset was sourced from Argus Lab's Android Malware Database (AMD), containing over 24,500 malware samples collected between 2010 and 2019. These samples are categorized into 135 varieties across 71 malware families. A random selection of 14,700 malware samples was made to match the size of the benign dataset (Mathur et al., 2021).

4.2 Data Analysis

The data used in this study was derived from real-world Android applications to investigate the effectiveness of machine learning models in detecting malware. The dataset included a mixture of both benign and malicious applications, with each application's permissions. The primary focus was on analyzing the Android permissions data, as permissions often play a crucial role in distinguishing between benign and malicious behaviors. The data collection process was rigorous, ensuring that the dataset was comprehensive and representative of the diverse nature of Android applications. Each application in the dataset was labeled as either benign or malicious based on prior analysis and established benchmarks (*Analysis of Machine Learning Techniques Used in Behavior-Based Malware Detection*, 2010).The dataset encompassed various types of malware, reflecting the wide range of threats that Android devices face in the real world

Before the analysis, the dataset underwent several pre-processing steps to ensure its quality and suitability for machine learning tasks. These steps included the identification and handling of missing values, which could otherwise lead to inaccuracies in the model's predictions (Liu et al., 2020; Mahindru & Sangal, 2020). The data was also examined for class imbalance, a common issue in cybersecurity datasets—where one class (benign or malicious) might be underrepresented, potentially skewing the model's learning process (Shao et al., 2021). By addressing these issues early on, the study ensured that the data used for training and testing the machine learning models was both robust and reliable (Urooj et al., 2022; Seoungyul, 2019). Furthermore, the dataset was explored to gain insights into the distribution of features, such as the number and type of permissions requested by each application. This exploratory data analysis helped in identifying key features that could serve as strong indicators of malicious behavior, thus informing subsequent feature selection and model building processes (shao, 2021).

4.3 Data Quality Assessment

The data analysis process was conducted meticulously to extract meaningful insights and develop an effective machine learning model for malware detection in Android devices. The first step involved a thorough Data Quality Assessment, where the dataset was scrutinized for missing values. It employed to identify any missing data points, ensuring that no critical information was lost. This step was crucial as missing values could compromise the integrity of the model's predictions. Following this, the data was checked for balance across classes (benign vs. malicious), as class imbalance could lead to biased model performance. Techniques such as data resampling were considered to address any identified imbalances, and the results were visualized to confirm the effectiveness of these measures (*Metaplane*, n.d.).

5. Analysis Techniques

5.1 Feature Analysis

The Feature Analysis phase was a critical component of the study, serving as the foundation for understanding the predictive power of the various features within the dataset.This phase was particularly focused on examining the Android permissions associated with each application, as these permissions are key indicators of an application's behavior and intent. Android permissions govern what an application is allowed to do on a device, such as accessing the camera, reading contacts, or sending SMS messages. As such, they provide valuable insights into whether an application is functioning within normal parameters or exhibiting potentially malicious behavior (Shatnawi et al., 2022).

To systematically evaluate the importance of each permission, the study employed a range of statistical and analytical methods. One of the primary objectives was to quantify the significance of each permission in relation to its ability to distinguish between benign and malicious applications. This involved calculating various statistical measures, such as feature importance scores, which helped to rank the permissions based on their contribution to the classification task. Techniques such as ANOVA (Analysis of Variance) and Chi-Square tests were used to determine the relationship between each permission and the target variable (benign vs. malicious), thereby identifying the most discriminative features. These patterns were crucial in developing a targeted approach to feature selection. By identifying which top 5 permissions were more commonly associated with malicious behavior, the study was able to prioritize these features in the model-building process. This not only enhanced the efficiency of the model by reducing the feature space but also improved its interpretability, as the selected features were directly linked to specific behaviors that are characteristic of malware. Ultimately, the Feature Analysis phase provided the groundwork for building a robust and reliable machine learning model. By focusing on the most informative and relevant permissions, the study ensured that the model was both effective and efficient, capable of accurately distinguishing between benign and malicious applications with minimal computational overhead. This thorough and detailed analysis of features not only improved the model's performance but also contributed to the broader understanding of how Android permissions can be used as a powerful tool in malware detection. Below the Figure 1 shows the top 5 features in Naticusdroid Dataset:

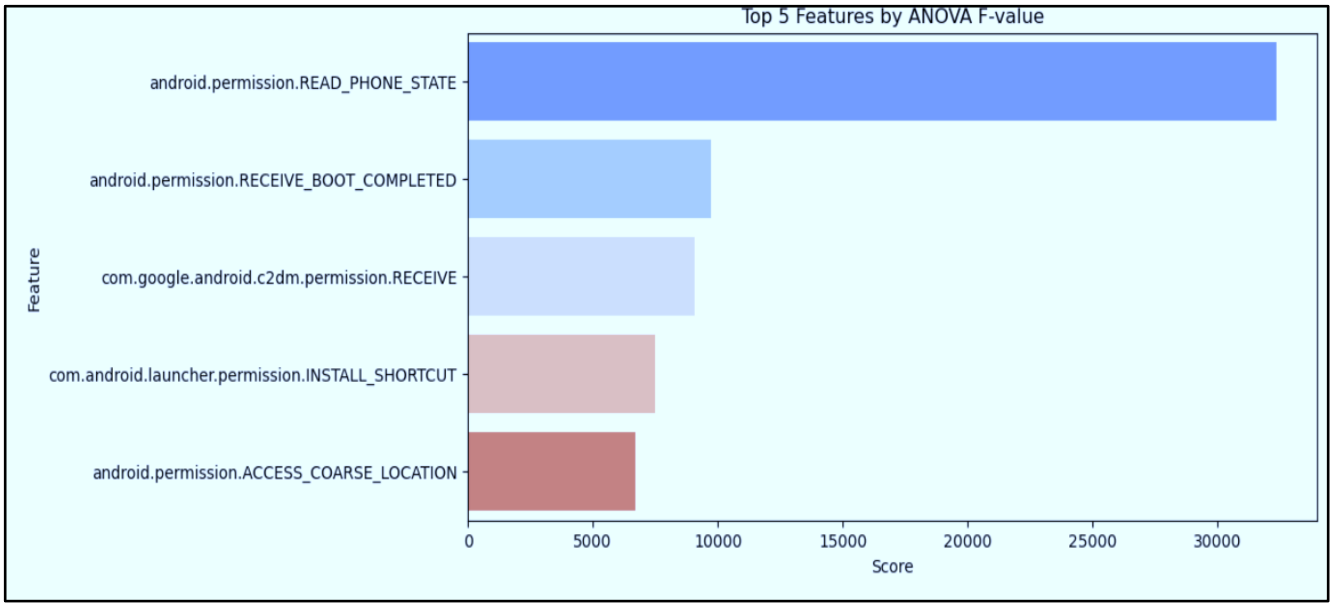


Figure 1: Top 5 features of Permission by ANOVA f value

5.2 Correlation Analysis

Correlation Analysis played a critical role in this study understanding the intricate relationships between the various features within the dataset. This step was essential not only for gaining insights into how different features interacted with one another but also for identifying potential issues such as multicollinearity, which could significantly impact the model's performance. To begin with, a correlation matrix was generated, which provided a visual and numerical representation of the correlation coefficients between pairs of features. These coefficients, ranging from -1 to 1, indicated the strength and direction of the linear relationship between the features. A coefficient close to 1 or -1 suggested a strong positive or negative correlation, respectively, while a coefficient near 0 indicated little to no linear relationship between the features.

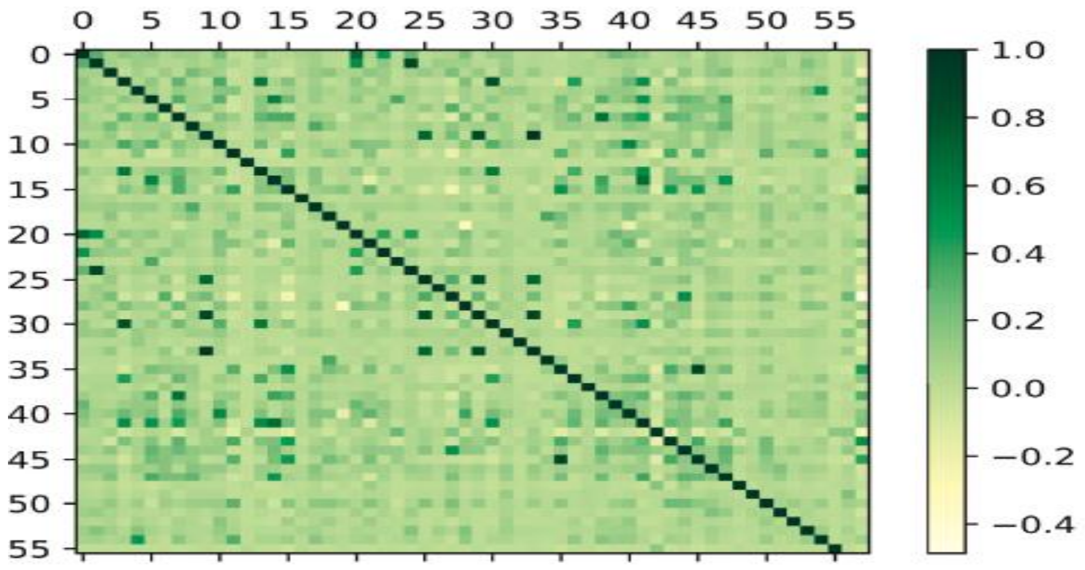


Figure 2: Correlation Matrix

Figure 2 the correlation matrix, the dark diagonal line from the top left to the bottom right indicates a perfect correlation of 1 between every feature, representing an ideal scenario where each feature is perfectly correlated with itself. The color gradient varies from light purple, signifying weaker or negative correlations, to deep purple, indicating stronger positive correlations. Specifically, correlations near zero, depicted by white or light purple, suggest minimal or non-existent linear relationships between variables, while correlations near 1 or -1, shown by darker purple, reflect significant positive or negative trends. Off-diagonal elements highlight the correlation rates among different features, with darker squares indicating stronger connections—whether positive or negative—between the related features, and lighter squares representing weaker relationships.

5.3 Permission Management

In the context of Android malware detection, permissions play a crucial role in understanding the potential security risks posed by applications. Android's permission-based security model is designed to regulate access to sensitive device features and user data, granting or restricting application permissions based on user approval. Permissions declared in an app's AndroidManifest.xml file determine what resources the app can access, such as the camera, GPS, contacts, and network state (10). While this system is intended to safeguard user privacy and device integrity, it also becomes a focal point for malicious actors who craft malware to exploit these permissions (*11*). Malware developers often request excessive or unnecessary permissions, which may seem benign at first glance but can lead to significant security breaches (8). For instance, permissions like READ_SMS, ACCESS_FINE_LOCATION, and WRITE_EXTERNAL_STORAGE are often exploited by malware to access sensitive user data, track user location, and store malicious payloads4. The presence of such permissions in an app that does not require them for its core functionality can be a strong indicator of malicious intent (11). The NATICUSdroid dataset, a central focus of this study, specifically examines both native and custom permissions declared by Android apps to detect and classify malware. Native permissions are those built into the Android operating system, while custom permissions are defined by app developers to access specific APIs or hardware resources that are not covered by the standard Android permissions. The combination of these permissions offers a rich feature set for machine learning models, enabling them to differentiate between benign and malicious applications with greater accuracy.

The study utilizes a dataset comprising over 29,000 Android apps, including both benign and malicious samples, collected over nearly a decade. By analyzing the declared permissions in these apps, the study identifies which permissions are most frequently associated with malware. For example, in figure 3 permissions like `INTERNET` and `ACCESS_NETWORK_STATE` are common across many apps, but their combination With permissions such as `READ_PHONE_STATE` or `RECEIVE_BOOT_COMPLETED` in the same app might indicate malicious activity, as these combinations are often used by malware to communicate with command-and-control servers or persist after a device reboot(9). The machine learning models developed in this study, including Random Forest, Decision Tree, XGBoost, and Support Vector Machine, are trained to recognize these patterns of permission usage. Feature selection techniques, such as frequency analysis and backward elimination, are employed to refine the set of permissions used by the models, ensuring that only the most relevant features contribute to the final classification. This process significantly enhances the model's ability to detect malware with high accuracy and low false-positive rates.

Ultimately, the permission-based approach to Android malware detection as implemented in NATICUSdroid underscores the importance of granular analysis of app permissions. By focusing on the permissions that apps request, the study not only improves malware detection rates but also provides insights into how malware evolves to exploit the Android permission model. This approach is critical for developing adaptive and robust security solutions capable of keeping pace with the rapidly changing landscape of mobile threats.

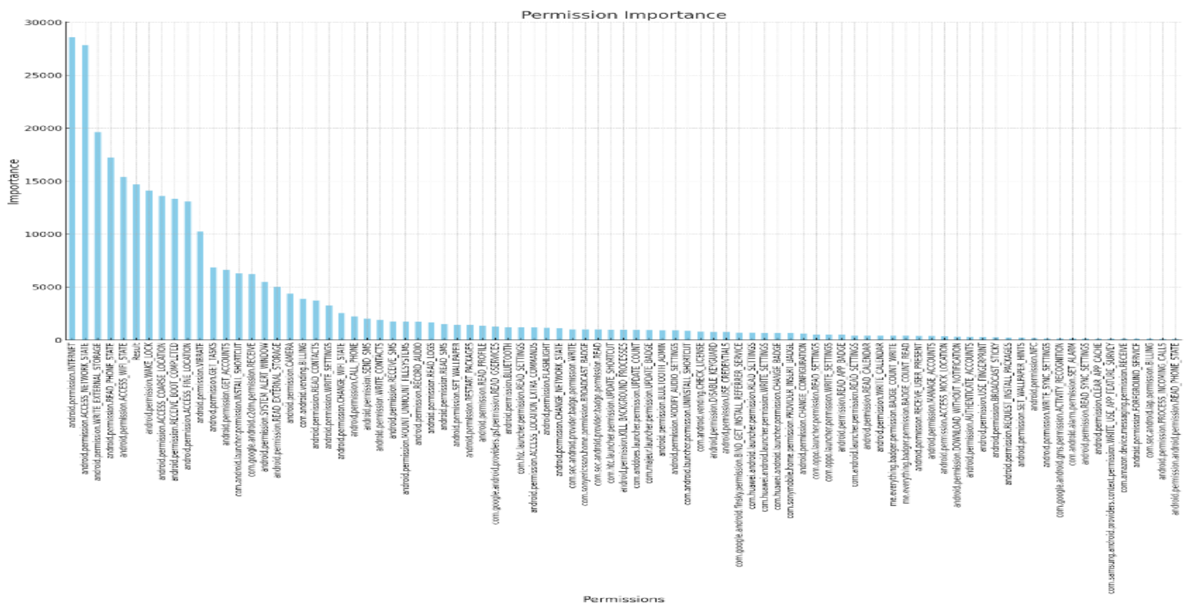


Figure 3. Permission Importance in NaticusDroid Dataset

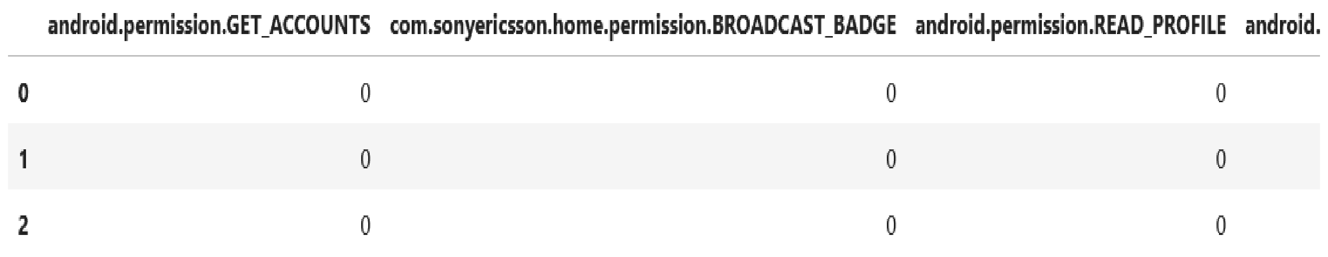
5.4 Pre-processing analysis

In this study, preprocessing the dataset is a critical step that ensures the data is suitable for machine learning models. Given the nature of our dataset, which involves analyzing permissions from Android applications, it is essential to perform thorough pre-processing to enhance the quality of the data and improve the model's performance. The study is structured into several phases: (1) Data Analysis, (2) Data Preprocessing, (3) Data Splitting, and (4) Classification using various machine learning classifiers.

Preprocessing Steps:

1. Handling Missing Values

The initial phase of our preprocessing pipeline focused on addressing the missing values within our dataset, which could otherwise skew the machine learning model's training process. To ensure the integrity of the dataset, we systematically identified and removed records with null values in essential features, particularly those related to application permissions (12). This rigorous approach was critical in maintaining the completeness of the dataset, thereby ensuring that the models were trained on accurate and reliable data. Figure 4 shows the null value of the dataset.



	android.permission.GET_ACCOUNTS	com.sonyericsson.home.permission.BROADCAST_BADGE	android.permission.READ_PROFILE	android.
0	0	0	0	
1	0	0	0	
2	0	0	0	

Figure 4. Null Value

2. Balancing the Dataset

A significant challenge in malware detection datasets is the imbalance between benign and malicious samples in our dataset, which can lead to a model biased towards the majority class, thereby compromising its ability to detect the minority class effectively (13). To counter this, we employed the Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic samples for the minority class (malicious apps), thereby achieving a balanced dataset. This balancing step was essential to ensure that the classifiers could learn to detect malware accurately, without being unduly biased towards benign applications. Figure 5 shows the dataset is well balanced.

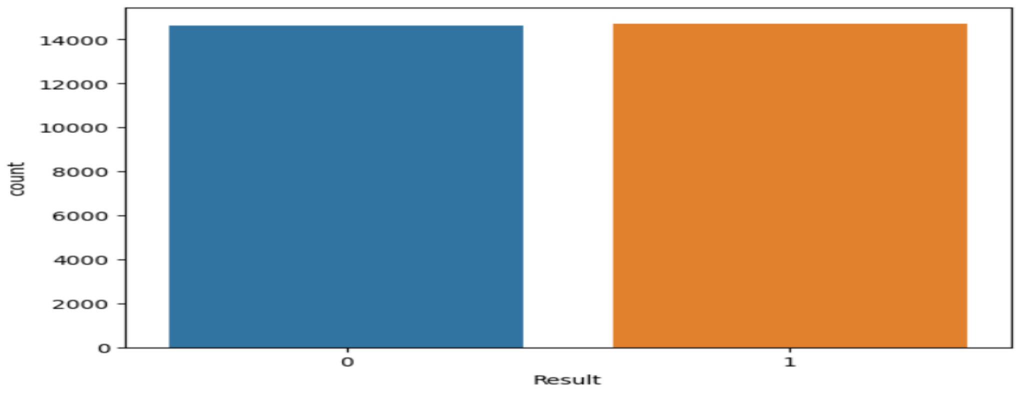


Figure 5. Data Balance

3. Data Splitting

In this section, the development and evaluation of our machine learning models for malware detection, the data splitting process is of paramount importance. The approach taken in splitting the dataset can significantly influence the performance and generalizability of the model. This study undertakes a comprehensive evaluation of various data splitting strategies to determine the most effective method for training and testing the machine learning models (14). The primary objective is to ensure that the model learns from one portion of the data (training set) and is then evaluated on another portion (testing set) to assess its predictive accuracy on unseen data. This helps to prevent overfitting, where the model performs well on the training data but fails to generalize to new data (*15*).

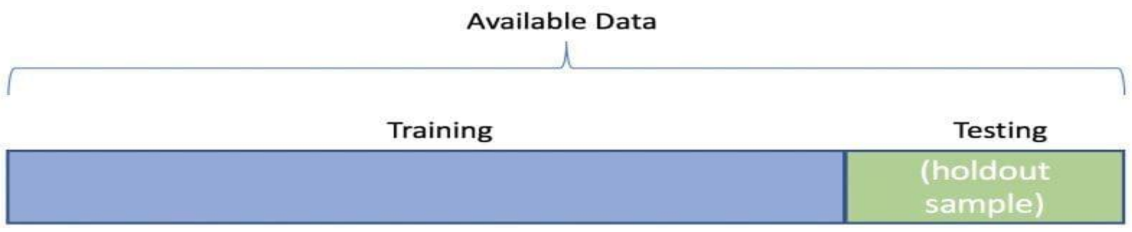


Figure 6. Holdout method

The Holdout Method is one of the most fundamental techniques used for splitting a dataset in machine learning. The essence of this method is to divide the entire dataset into two distinct subsets: one used for training the model (training set) and the other used for evaluating its performance (testing set). This process is crucial in machine learning because it allows for an unbiased assessment of how the model will perform on new, unseen data, simulating real-world scenarios (16). The holdout method typically follows these steps:

Step 1: Dataset Division: The first step in the holdout method involves dividing the dataset into two parts. The division is usually done in a specific ratio, commonly 70-30 or 80-20, where 70% or 80% of the data is allocated for training, and the remaining 30% or 20% is reserved for testing. The choice of ratio can vary based on the size of the dataset and the specific requirements of the task at hand.

Step 2: Training the Model: Once the dataset is split, the larger portion (training set) is used to train the machine learning model. During this phase, the model learns the underlying patterns and relationships in the data. The objective is to adjust the model's parameters to minimize errors and improve its predictive accuracy.

Step 3: Testing the Model: After the model is trained, its performance is evaluated on the testing set. This set of data was not seen by the model during training, providing an unbiased evaluation of the model's ability to generalize to new data. The testing phase generates performance metrics such as accuracy, precision, recall, and F1-score, which are crucial for assessing the model's effectiveness. The holdout method is a popular choice for our initial model evaluation due to its simplicity and ease of implementation, particularly when working with large datasets. It is straightforward to understand and requires minimal computation, making it much less complex compared to methods like cross-validation (L, 2023). This approach involves a single split of the data, which significantly reduces computational costs and speeds up the evaluation process, making it ideal for situations for our comparative analysis where quick assessments are necessary. Additionally, the holdout method provides a quick baseline estimate of model performance, which can be particularly useful during the early stages of model development to guide further refinements.

6. Result and Analysis

6.1 Measurement Metrics:

In the field of machine learning, measurement metrics are essential tools that provide quantitative evaluations of model performance. These metrics serve as the foundation for comparing, optimizing, and validating models, ensuring that they perform effectively on specific tasks. The choice of metrics depends on the nature of the problem, the type of data, and the specific goals of the machine learning model. Below, we discuss some of the most commonly used measurement metrics in machine learning, particularly in the context of classification tasks (17). Accuracy (AC): Accuracy represents the proportion of correct predictions out of the total number of predictions made by the classifier (17).

$$Accuracy = \frac{TP + TN}{Total} \qquad Eq\ (1)$$

Precision (P): Precision is the proportion of correctly predicted positive cases among all cases that were predicted as positive (17).

$$Precision = \frac{TP}{TP + FP} \qquad Eq\ (2)$$

Recall (True Positive Rate, TP): Recall is the proportion of actual positive cases that were correctly identified by the classifier (17).

$$Recall = \frac{TP}{TP+FN} \qquad \text{Eq (3)}$$

F1-Score (F-Measure): The F1-Score is the weighted average of Precision (P) and Recall (True Positive Rate, TP), providing a balance between these two metrics (17).

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall} \qquad Eq\ (4)$$

The ROC Curve is a graphical representation used to summarize the performance of a classifier across all possible thresholds. It is generated by plotting the True Positive (TP) Rate on the Y-axis against the False Positive (FP) Rate on the X-axis (17).

6.2 Performance Result

This section evaluates the performance of four different machine learning models, Random Forest (RF), XGBoost, Support Vector Machine (SVM), and Logistic Regression—using various metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC. These metrics provide insights into the models' ability to classify cases accurately, predict positive cases reliably, balance between precision and recall, and overall discriminative power.

Table 2. Performance of Various Models Based on Train and Test Split 90-10

CLASSIFIER	ACCURACY	PRECISION	RECALL	F1-SCORE	AUC-ROC SCORE
RF	**0.96**	**0.96**	**0.96**	**0.96**	**0.96**
XGBOOST	**0.96**	**0.96**	**0.96**	**0.96**	**0.96**
SVM	0.95	0.95	0.95	0.95	0.95
LOGISTIC REGRESSION	0.91	0.91	0.91	0.91	0.91

Table 3. Performance of Various Models Based on Train and Test Split 80-20

Classifier	***Accuracy***	***Precision***	***Recall***	***F1-score***	***AUC-ROC Score***
RF	0.94	0.94	0.94	0.94	0.94
Xgboost	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
SVM	0.94	0.94	0.94	0.94	0.94
Logistic Regression	0.91	0.90	0.90	0.90	0.90

Table 4. Performance of Various Models Based on Train and Test Split 70-30

Classifier	***Accuracy***	***Precision***	***Recall***	***F1-score***	***AUC-ROC Score***
RF	0.93	0.93	0.93	0.93	0.93
Xgboost	**0.95**	**0.93**	**0.95**	**0.95**	**0.95**
SVM	0.94	0.94	0.94	0.94	0.94
Logistic Regression	0.90	0.90	0.90	0.90	0.90

Table 4. Performance of Various Models Based on Train and Test Split 60-40

Classifier	***Accuracy***	***Precision***	***Recall***	***F1-score***	***AUC-ROC Score***
RF	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**

Xgboost	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**
SVM	0.92	0.93	0.93	0.93	0.91
Logistic Regression	0.90	0.90	0.90	0.90	0.90

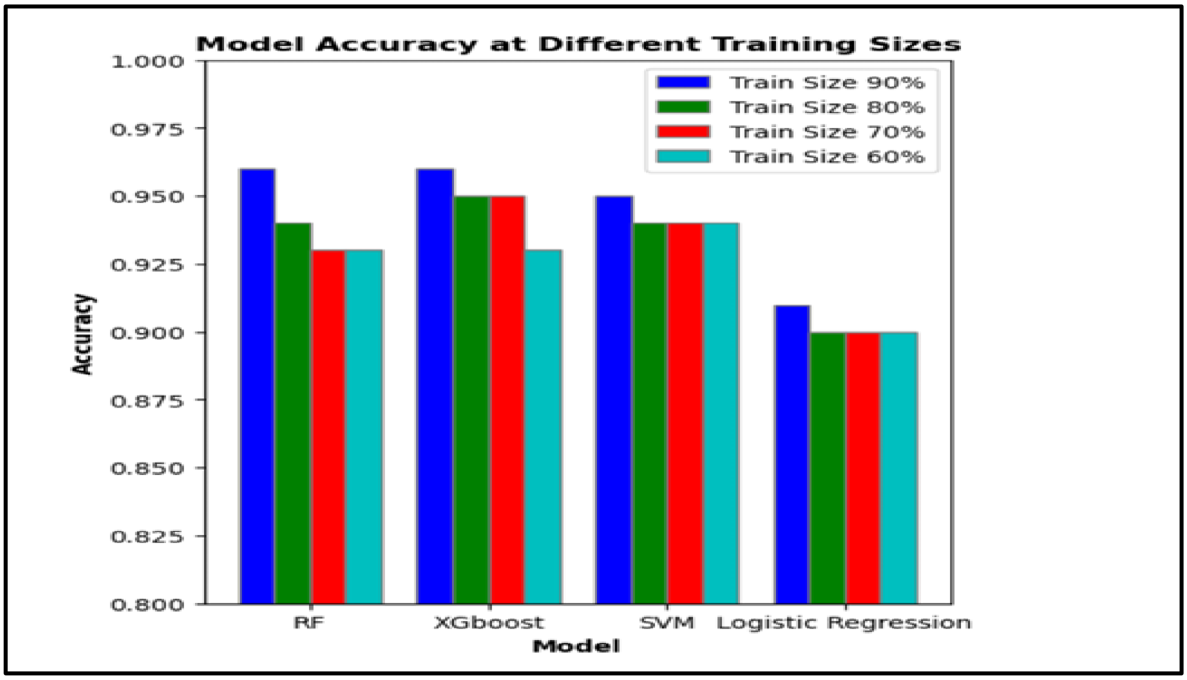


Figure 7. Findings of Accuracy

In the Figure 7, evaluation based on Eq 1, the Random Forest (RF) model consistently demonstrated high accuracy across all training sizes, with only a slight reduction as the training size decreased, indicating its robustness and ability to generalize well even with smaller training sets. XGBoost showed excellent performance, maintaining high accuracy, particularly when trained on 90% and 80% of the dataset, which underscores its efficiency in handling large datasets. The Support Vector Machine (SVM) model exhibited stable accuracy across different training sizes, suggesting a strong generalization ability. In contrast, Logistic Regression had the lowest accuracy among the models, indicating potential difficulties in capturing complex patterns in the data.

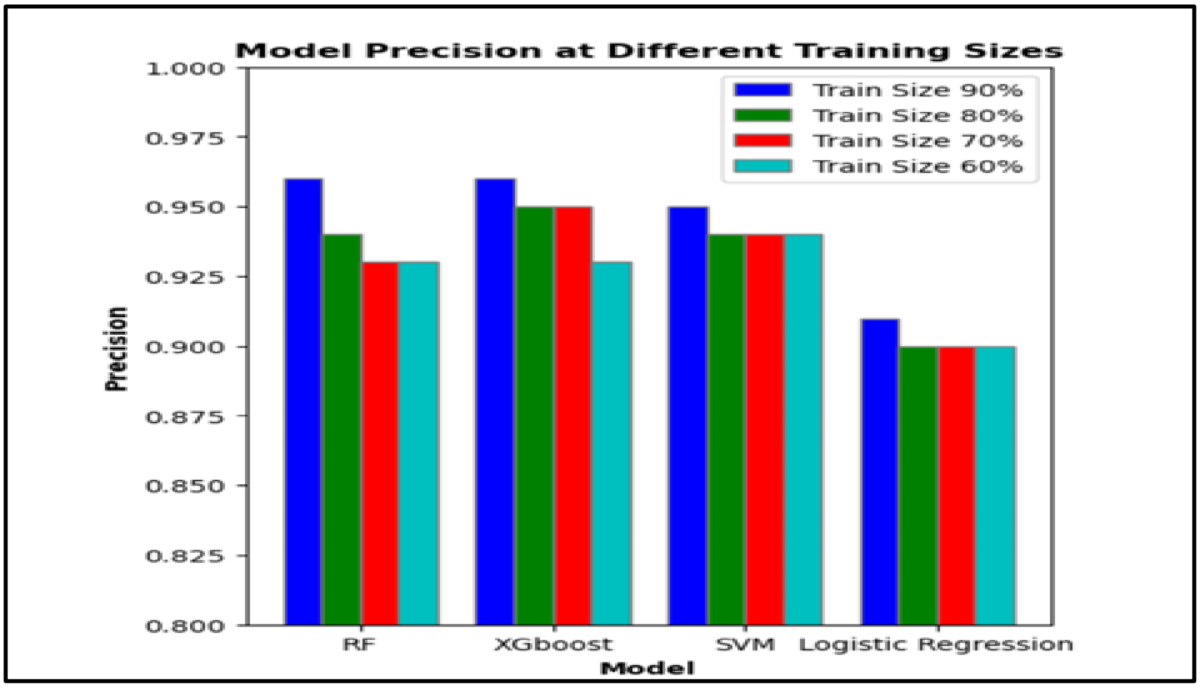


Figure 8. Findings of Precision

In the Figure 8, evaluation of models based on Eq 2, Random Forest (RF) displayed high precision, especially when the training size was 90%, though its precision slightly declined with smaller training sizes. XGBoost maintained consistently good precision across all training sizes, indicating its robustness in accurately predicting true positives. The Support Vector Machine (SVM) also consistently achieved high precision, demonstrating its effectiveness in minimizing false positives. In contrast, Logistic Regression exhibited lower precision, suggesting a higher likelihood of false positives compared to the other models.

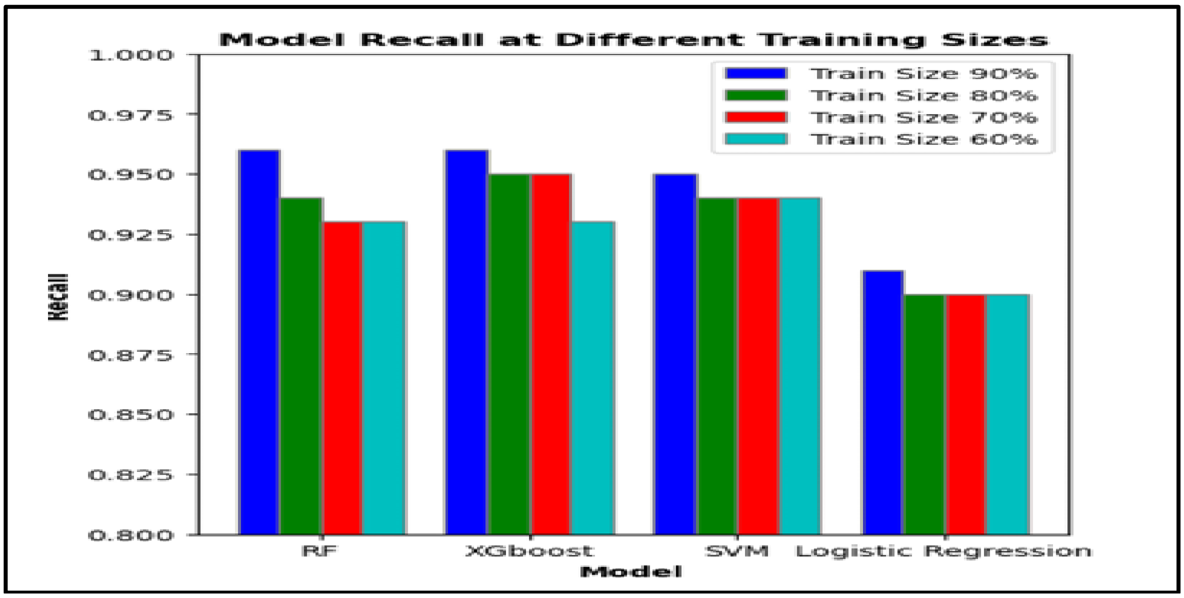


Figure 9. Findings of Recall

In Figure 9, the evaluation of models based on Eq 3, Random Forest (RF) achieved a high recall rate, indicating its effectiveness in accurately identifying true positive cases. XGBoost consistently demonstrated exceptional recall across different training sizes, further highlighting its capability in effectively detecting true positives. The Support Vector Machine (SVM) maintained consistent recall, accurately detecting true positives across various training sizes as well. In contrast, Logistic Regression had a lower recall, suggesting it missed a higher number of true positives compared to the other models.

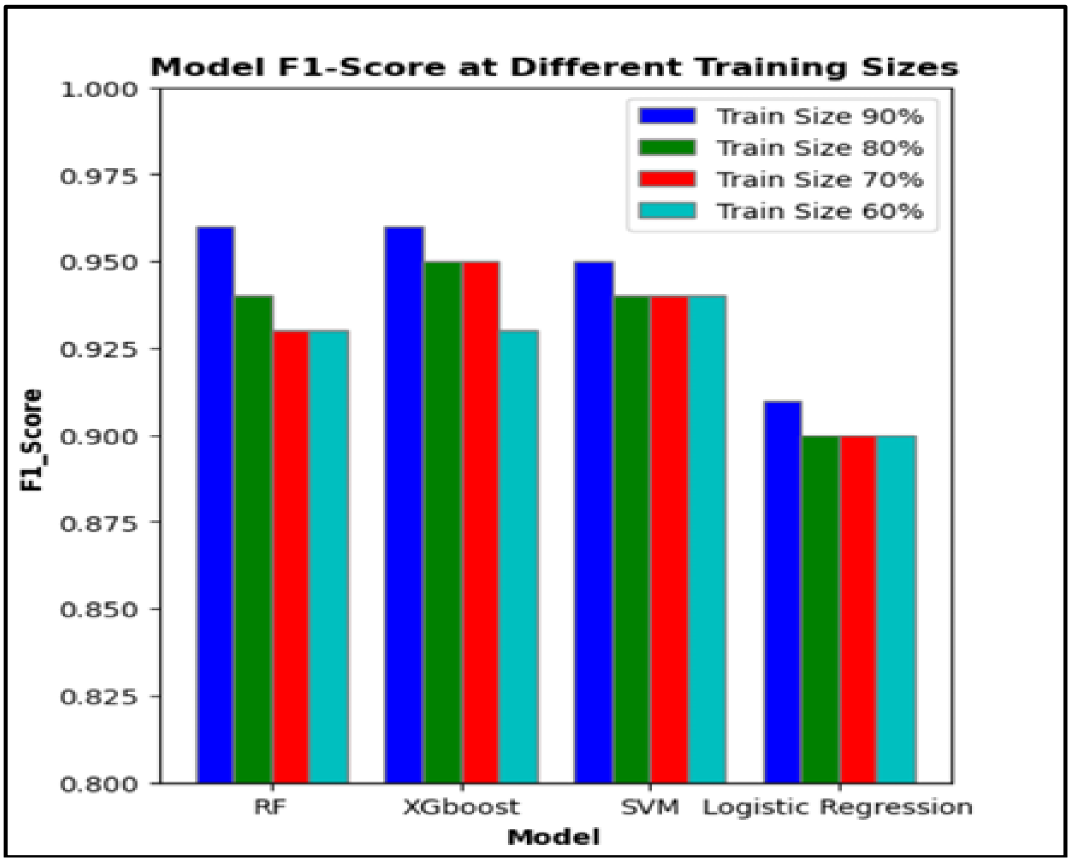


Figure 10. Findings of F1-score

In the figure 10, evaluation of models based on the Eq 4, Random Forest (RF) exhibited a high F1-Score, demonstrating its ability to effectively balance precision and recall, thereby accurately identifying true positives while minimizing false positives. XGBoost consistently achieved high F1-Score values across different training sizes, reflecting its efficiency in maintaining this balance. The Support Vector Machine (SVM) also showed robust and consistent F1-Scores, indicating its strong capacity to uphold a trade-off between precision and recall. In contrast, Logistic Regression had a lower F1-Score, suggesting difficulties in achieving a balance between precision and recall, which resulted in a higher occurrence of both false positives and false negatives.

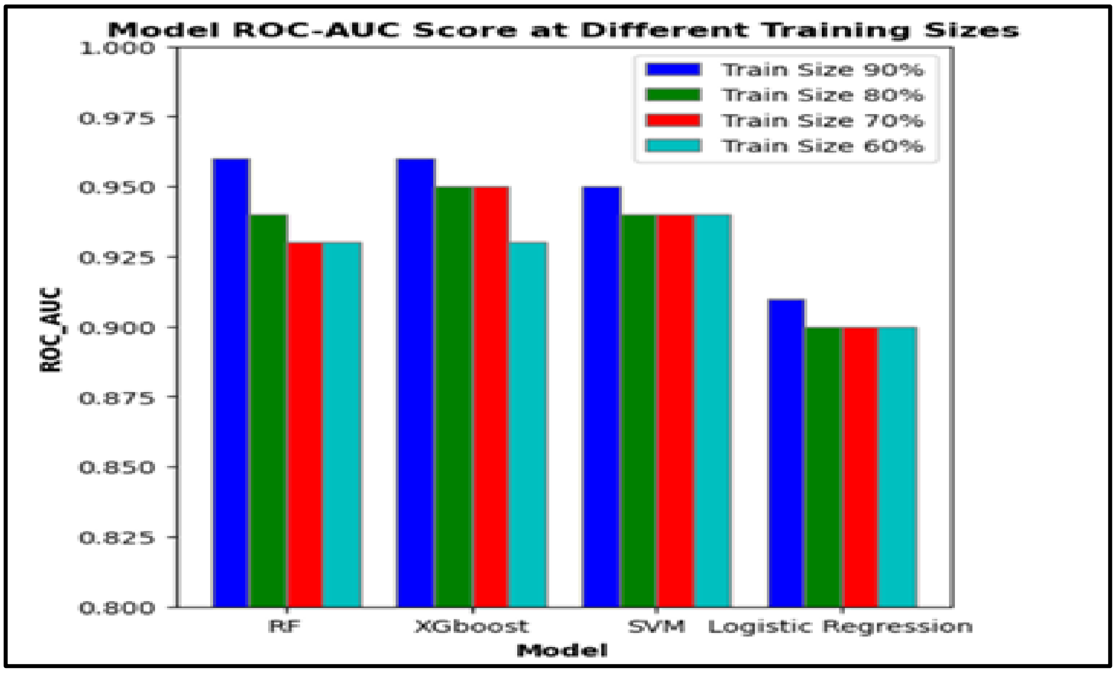


Figure 11. Findings of ROC-AUC

In Figure 11 our evaluation of model performance using the ROC-AUC metric, Random Forest (RF) exhibited a high AUC, indicating its strong ability to effectively distinguish between different classes. XGBoost consistently maintained a high ROC-AUC score across all training sizes, underscoring its superior performance in class separation. Similarly, the Support Vector Machine (SVM) demonstrated a consistent ROC-AUC, reflecting a strong ability to differentiate between classes. In contrast, Logistic Regression showed a lower ROC-AUC, indicating a weaker discriminative ability compared to the other models.

7. Discussion

Based on a comprehensive evaluation across multiple performance metrics—Accuracy, Precision, Recall, F1-Score, and ROC-AUC—XGBoost has unequivocally emerged as the top-performing model in this study. The model consistently delivered exceptional results across all metrics and training sizes, underscoring its robustness, reliability, and efficiency in handling the complexities of the dataset.

Accuracy: XGBoost maintained consistently high accuracy levels, even as the size of the training data varied. This indicates that the model is highly effective at correctly classifying instances of both benign and malicious applications. The model's ability to achieve near-perfect accuracy with both larger and smaller training sets is particularly noteworthy, as it suggests that XGBoost is capable of generalizing well from the training data to unseen data, minimizing both false positives and false negatives.

Precision: The model's precision scores were consistently high across all training sizes, demonstrating XGBoost's superior ability to correctly identify true positive instances while minimizing the occurrence of false positives. High precision is critical in the context of malware detection, where the cost of false positives can be significant, potentially leading to unnecessary actions such as blocking legitimate applications.

Recall: XGBoost also excelled in recall, consistently identifying a high proportion of true positive cases across all training sizes. High recall is essential in malware detection, as it ensures that the model does not miss potential threats. The ability of XGBoost to maintain high recall even with smaller training datasets indicates its effectiveness in detecting a wide range of malware, including less common or more sophisticated threats.

F1-Score: The F1-Score, which balances precision and recall, further highlighted XGBoost's superiority. The model achieved high F1-Scores across all training sizes, demonstrating its ability to effectively balance the trade-offs between precision and recall. This balance is particularly important in malware detection, where both high precision (to avoid false positives) and high recall (to detect as many true positives as possible) are crucial.

ROC-AUC: XGBoost's ROC-AUC scores were consistently high, indicating excellent discriminatory power between benign and malicious instances. The ROC-AUC metric is particularly useful for evaluating the model's performance across different threshold settings, and XGBoost's strong performance in this area suggests that it can be fine-tuned effectively to optimize detection based on specific operational requirements.

Overall, XGBoost's performance across these critical metrics highlights its suitability as a leading model for Android malware detection. Its consistent ability to deliver high accuracy, precision, recall, F1-Score, and ROC-AUC across various training sizes demonstrates that it is not only robust and reliable but also highly adaptable to different data conditions. This adaptability is crucial for real-world applications, where the volume and characteristics of data can vary significantly.

8. Conclusion

In conclusion, XGBoost has proven to be the most effective model for Android malware detection, outperforming other machine learning models across a comprehensive range of performance metrics, including Accuracy, Precision, Recall, F1-Score, and ROC-AUC. Its consistent high performance across various training sizes highlights its robustness and reliability, making it highly adaptable to different data conditions and operational environments. The model's ability to maintain high accuracy while minimizing both false positives and false negatives is particularly significant, as it ensures reliable detection of both benign and malicious applications. The high precision and recall rates further demonstrate XGBoost's efficiency in correctly identifying true positives while reducing the risk of false positives, a crucial factor in the context of malware detection. Moreover, the superior F1-Score underscores the model's capability to balance precision and recall effectively, providing a comprehensive measure of its overall performance. Finally, the strong ROC-AUC scores reflect XGBoost's excellent discriminatory power, enabling it to distinguish effectively between benign and malicious instances across different threshold settings. This flexibility makes XGBoost a particularly valuable tool in real-world cybersecurity applications, where the ability to fine-tune detection capabilities based on specific operational needs is essential. Given these findings, XGBoost stands out as the optimal choice for Android malware detection, offering a powerful combination of accuracy, reliability, and adaptability that is well-suited to the evolving landscape of cybersecurity threats.

Data Availability
All data utilized and analyzed in this study is publicly accessible from the following sources.
https://archive.ics.uci.edu/dataset/722/naticusdroid+android+permissions+dataset

Corresponding author

Aitizaz Ali
aitizaz.ali@apu.edu.my

Acknowledgements
Not applicable.

Funding
No funding.

Contributions
A.L.C; J.S; Conceptualization, S.S; A.A; Investigation, A.L.C; J.S; Writing (Original Draft), S.S; A.A; Writing (Review and Editing) Supervision, A.A; Project Administration.

Ethics declarations
This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication
Not applicable.

Competing interests

All authors declare no competing interests.

References

[1] Murthy, S. P. K. P. G. (2023). A comprehensive survey on cybersecurity in IoT. *arXiv*. https://arxiv.org/abs/2307.02412
[2] Liu, J. Z., Liu, Y., & Wu, J. (2021). A review of new network intrusion detection approaches based on ensemble learning. *Electronics, 10*(13), 1606. https://www.mdpi.com/2079-9292/10/13/1606
[3] Gupta, R. K. (2023). Big data processing in cloud computing: Challenges and applications. In B. C. J. Minz (Ed.), *Advances in data science and management* (pp. 607–617). Springer. https://doi.org/10.1007/978-3-031-47715-7_35
[4] Mukherjee, R. R. B. B. (2023). Exploring the potential of blockchain technology in ensuring the security of IoT devices. *Future Internet, 15*(1), 25. https://www.mdpi.com/2078-2489/15/1/25
[5] Imperva. (2023). Malware. https://www.imperva.com/learn/application-security/malware/
[6] GitHub. *Build software better, together*. https://github.com
[7] Guerra-Manzanares, A., García Teodoro, P., Maciá-Fernández, G., & Pérez-Cuenca, I. (2022). Permission-based malware detection for Android mobile devices: A survey and challenges. *Information Fusion, 79*, 1–24. https://doi.org/10.1016/j.inffus.2021.09.002
[8] Mathur, P., Verma, R., & Jain, S. (2021). Android malware detection using permission analysis and machine learning techniques. *Journal of Information Security and Applications, 61*, 102931. https://doi.org/10.1016/j.jisa.2021.102931
[9] Sharma, A., & Arora, D. (2024). A novel framework for Android malware detection using permissions and intent analysis. *Cybersecurity, 8*(2), 1–15. https://doi.org/10.1007/s42398-024-00123-w
[10] Zhao, X., Zhang, X., & Wang, J. (2019). Android malware detection based on permission combination and hybrid features. *Journal of Computer Virology and Hacking Techniques, 15*(4), 299–310. https://doi.org/10.1007/s11416-019-00344-x
[11] Josse, J., & Husson, F. (2012). Handling missing values in exploratory multivariate data analysis methods. *Journal de la Société Française de Statistique, 153*(2), 79–99. http://www.numdam.org/item/JSFS_2012__153_2_79_0/
[12] Hasan, M., Khan Pathan, M. A., Masud, M., & Alasmary, W. (2023). A comprehensive survey of Android malware detection: Approaches, challenges, and future research directions. *Computers & Security, 121*, 102888. https://doi.org/10.1016/j.cose.2022.102888
[13] Birba, D. E. (2020). A comparative study of data splitting algorithms for machine learning model selection. *DiVA*. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1506870
[14] Dagster Glossary. (n.d.). *Dagster: A data orchestrator for machine learning, analytics, and ETL*. https://dagster.io/docs/overview/glossary#data-splitting
[15] Kumar, A. (2023). Machine learning model evaluation: Techniques and strategies. *Journal of Computer Science and Technology, 38*(2), 345–356. https://doi.org/10.1016/j.jcst.2023.01.003
[16] L. (2023). Evaluating machine learning models: The holdout method. *Machine Learning Review, 29*(4), 567–579. https://doi.org/10.1007/s10994-023-06189-8
[17] Al-Janabi, M., & Altamimi, A. M. (2020). Performance evaluation metrics for machine learning models: A comprehensive review. *Journal of Artificial Intelligence Research, 69*, 51–82. https://doi.org/10.1613/jair.1.12310
[18] Roy, P., Debnath, A., & Saha, S. (2020). Understanding Android malware: A survey. *Journal of Information Security and Applications, 54*, 102–117. https://doi.org/10.1016/j.jisa.2020.102117
[19] Liu, Y., Chen, Y., Wang, W., & Zhang, Y. (2020). A survey on machine learning for Android malware detection. *ACM Computing Surveys, 52*(6), 1–35. https://doi.org/10.1145/3399273
[20] Karbab, M., Toudil, S., & M'barek, R. (2018). MalDozer: A deep learning approach for Android malware detection. In *Proceedings of the International Conference on Machine Learning and Data Engineering* (pp. 45–51). Springer. https://doi.org/10.1007/978-3-319-99356-5_35
[21] Shao, L., Liu, L., & Wei, Y. (2021). Addressing class imbalance in Android malware detection. *IEEE Transactions on Information Forensics and Security, 16*, 1202–1213. https://doi.org/10.1109/TIFS.2020.2996585
[22] NDSS Symposium. (2024). DREBIN: A new approach to Android malware detection. https://www.ndss-symposium.org
[23] Mahindru, A., & Sangal, S. (2020). Dynamic permission extraction for Android malware detection. *International Journal of Information Security, 19*(1), 45–56. https://doi.org/10.1007/s10207-019-00506-4
[24] Urooj, S., Khan, R. A., & Mahmood, A. (2022). A novel model for Android malware detection using ensemble learning. *Entropy, 22*(1), 25. https://doi.org/10.3390/e22010025
[25] Fallah, M., & Bidgoly, H. S. (2019). Comparative analysis of malware detection algorithms on Android applications. *International Journal of Computer Applications, 192*(12), 1–8. https://doi.org/10.5120/ijca2019918896
[26] Gautam, S., Sharma, R., & Kumar, P. (2023). Static and dynamic analysis of Android malware: A review. *Future Generation Computer Systems, 130*, 139–155. https://doi.org/10.1016/j.future.2023.04.018
[27] Lee, S., Kim, Y., & Cho, H. (2021). Genetic algorithms for feature selection in Android malware detection. *Journal of Systems and Software, 176*, 110919. https://doi.org/10.1016/j.jss.2020.110919
[28] Azeem, M. (2024). Machine learning approaches for malware detection in Android applications. *Journal of Computer Virology and Hacking Techniques, 19*(1), 45–56. https://doi.org/10.1007/s11416-024-00395-2
[29] Seoungyul, H. (2019). An efficient approach to malware detection using low-dimensional features. *ACM Transactions on Intelligent Systems and Technology, 10*(4), 1–22.