Analysis and Detection of Malware in Android Application using Machine Learning

K.J S Upendra[1], Smily Tappeta[2], Nitin Sigilipelli[3], Karuna Kumar Varjiparthi[4], Sai Kiran Miriyala[5], Venkata Manindra Guptha Pragallapati[6]

[1,2,3,4,5,6]*Artificial Intelligence and Data Science, Vishnu Institute of Technology, Bhimavaram, Andhra pradesh, India*

***Abstract*— Android smart phones are now frequently the subject of sophisticated malware assaults due to their rapid popularity, which presents serious security risks. The use of machine learning (ML) techniques for malware analysis and detection in Android applications is examined in this study. For feature extraction, a large dataset of both malicious and benign apps is used, with an emphasis on permissions, API calls, and behavioral patterns. To get precise predictions, sophisticated classifiers like Support Vector Machines (SVM), Random Forest, and Neural Networks are used. Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are two feature selection approaches used to improve detection performance. To increase resilience and scalability, cutting-edge techniques like federated learning, ensemble learning, and graph neural networks (GNNs) are being investigated. The efficiency of the suggested strategy is demonstrated by the experimental findings, which show a high classification accuracy of over 95%. The potential of incorporating ML models into Android security frameworks for real-time malware detection is highlighted by this study. The results open the door for scalable and flexible cyber security solutions that take into account the always changing mobile threat landscape.**

Keywords—Android malware detection, Machine learning, Feature extraction, Neural networks, Federated learning, Ensemble learning, Cyber security.

I. INTRODUCTION

Android devices rapid rise has revolutionized the mobile technology market and made smart phones a necessary component of everyday life. According to recent data, Android is the most popular operating system for smart phones worldwide, powering billions of devices. Android apps are now a vital tool for commercial operations, financial transactions, entertainment, and communication due to their extensive use [1]. But this pervasiveness has a drawback as well: as reliance on Android apps grows, malevolent actors have been drawn to take advantage of the platform's flaws. As a result, malware assaults now primarily target Android devices, creating serious security issues for both users and developers. Malicious software intended to interfere with, harm, or obtain unauthorized access to a user's device is known as Android malware [2]. Android malware, in contrast to conventional malware, takes advantage of the operating system's special qualities, notably its open-source nature and the simplicity with which apps may be distributed through third-party app stores. As a result, there is now a greater variety of malware, from trojans and ransom ware to spyware and adware. In addition to jeopardizing user privacy, these risks lead to monetary losses, identity theft, and harm to the organization's brand. Traditional malware detection methods, such as signature-based and heuristic-based approaches, have served as the foundation for combating these threats [3]. Signature-based methods rely on predefined patterns to identify known malware, while heuristic-based methods use rule-based algorithms to detect suspicious behavior. However, these approaches have significant limitations [4]. Signature-based methods are ineffective against zero-day attacks and polymorphic malware that continuously evolve to evade detection. Heuristic methods, though more adaptive, often suffer from high false-positive rates, leading to user inconvenience and mistrust.

Machine learning (ML) has become a potential paradigm for Android malware detection in light of these limitations. By using computational algorithms to examine and discover patterns in large datasets, machine learning (ML) techniques make it possible to

identify malware that is both known and undiscovered [5]. ML models can identify anomalies that conventional approaches might miss by concentrating on behavioral analysis and feature extraction. App permissions, API requests, network activities, and system interactions are among the key features examined in ML-based malware detection. These characteristics enable precise differentiation between harmful and benign apps by offering insightful information about the purpose and behavior of programs [6]. The purpose of this study is to investigate the use of cutting-edge machine learning algorithms for Android malware analysis and detection. In particular, the study uses classifiers like Random Forest, Support Vector Machines (SVM), and Neural Networks to achieve high accuracy in differentiating between malicious and legitimate applications [7]. To improve detection performance and lower computational cost, feature selection techniques such as Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are used to find the most pertinent features. This study also explores cutting-edge approaches including ensemble learning, graph neural networks (GNNs), and federated learning to handle the dynamic nature of malware. While GNNs examine intricate correlations between features, including API call sequences, ensemble learning integrates many models to increase accuracy and robustness. By using data from several devices, federated learning provides a decentralized method of training models while protecting user privacy. These cutting-edge methods ensure resilience against new threats by offering a scalable and flexible platform for real-time malware detection [8].

This research is important because it has the ability to incorporate machine learning models into Android security frameworks, providing dependable and proactive solutions to protect users and companies. The suggested method shows its effectiveness in reducing the dangers related to Android malware by attaining a classification accuracy of above 95% [9]. The research also emphasizes the necessity of ongoing innovation in cyber security, stressing the value of intelligent and adaptable solutions to combat the constantly shifting terrain of mobile threats. To sum up, this study lays the groundwork for creating malware detection systems that are accurate, scalable, and real-time, thereby bridging the gap between conventional and contemporary cyber security techniques [10]. The results open the door for further developments in Android security and highlight the vital role machine learning plays in tackling the problems brought on by a world that is becoming more linked by the day.

II. LITERATURE SURVEY

Numerous studies have investigated cutting-edge methods for Android malware detection that make use of machine learning (ML) techniques in recent years. Gouri et al. (2024) used classifiers including Support Vector Machines (SVM), Random Forest, and Neural Networks to create a prediction model that improves malware detection by examining app permissions and metadata. Although it has trouble with dynamic malware that changes over time, their technology overcomes the drawbacks of conventional detection techniques [11].

Atif Raza et al. (2024) achieved a 95 % detection accuracy by converting APK files into graphs using deep learning techniques, notably Graph Convolutional Networks (GCNs). Although their methodology outperforms conventional machine learning techniques, it has issues with scalability and computing efficiency in large-scale applications [12].

Darji (2024) conducted static feature analysis on Android APKs using Decision Tree, K-Nearest Neighbors, Linear SVM, and Random Forest classifiers. This method provided efficient malware identification, but it had drawbacks, including a high false-positive rate and obfuscated malware. To increase the precision of Android malware detection [13].

Jaikishan Mohanty and Divyashikha Sethia (2024) presented an ensemble learning technique that combines decision tree classifiers with stacking models like Gradient Boosting, AdaBoost, and Bagging. Despite lengthier training periods and less scalability due to the increased model complexity, their strategy obtained 82.38% accuracy. In order to detect malware using static analysis [14].

Raghuraman et al. (2024) suggested a hybrid deep learning model that combines Convolutional Neural Networks (CNN) with Gated Recurrent Units (GRU). Although this model increased accuracy and resilience, it had limitations in terms of adaptation to changing threats and computing cost. In the meantime [15].

Shuguang Xiong and Huitao Zhang (2024) combined the K-Nearest Neighbors, Decision Tree, and Logistic Regression classifiers to create a multi-model fusion approach. Compared to single-model techniques, this fusion strategy improved detection accuracy, but it also increased computing complexity. By using feature selection strategies based on feature relevance scores from Gradient Boosting [16].

Amarjyoti Pathak et al. (2024) aimed to enhance Android malware detection. Their method cut down on training time without sacrificing detection effectiveness, but it was heavily dependent on domain expertise and might have missed important details. In order to improve malware detection [17].

Vishal Kisan Borate et al. (2024) investigated a number of machine learning techniques, such as decision trees and neural networks; nevertheless, their method had issues with data imbalance and real-time application. Through hyperparameter tuning [18].

Ahmed Alhussen (2024) suggested optimizing Long Short-Term Memory (LSTM) and Neural Network models for malware detection, which resulted in nearly flawless accuracy but required a significant amount of processing power [19].

Lastly, dynamic malware analysis was carried out by Galia Villarroel et al. (2024) utilizing algorithms including XGBoost, LightGBM, and Random Forest; LightGBM produced the best performance metrics. However, their strategy ran into issues with real-time application limits and high computing load. Together, these studies demonstrate the growing difficulty of detecting Android malware and the potential of machine learning in tackling new security risks [20].

III. DATASET COLLECTION AND PREPROCESSING

To guarantee a thorough representation of Android apps, the dataset utilized in this study was assembled from a variety of sources. Benign apps were gathered from reliable resources, such as the Google Play Store, which offers apps that have been validated and are often used. The malicious samples came from malware sources that were openly accessible, such as AndroZoo, Drebin, and VirusShare. Numerous malware samples, including trojans, spyware, and ransom ware, can be found in these repositories. Third-party app stores were also included in order to find possibly dangerous apps that established markets sometimes ignore. Metadata, static information (such requested permissions and manifest files), and dynamic features (like API call sequences and runtime behavioral patterns) make up the dataset. Based on its behavior and the classification of the source repository, each program was classified as either benign or malicious. After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

App Name	Permissions Count	API Calls Count	Behavioral Pattern	Label
App1	12	25	Frequent network access	Malicious
App2	5	8	Standard functionality	Benign
App3	20	30	Suspicious data usage	Malicious
App4	7	12	Limited access	Benign
App5	15	40	Abnormal permissions	Malicious
App6	3	5	Minimal activity	Benign
App7	18	22	Data Ex filtration	Malicious
App8	6	10	Normal behavior	Benign
App9	13	35	Excessive permissions	Malicious
App10	8	15	Standard functionality	Benign

To guarantee the quality, balance, and applicability of the data for machine learning tasks, a number of methodical measures were taken during the dataset gathering and preprocessing stages. To preserve data integrity, programs with missing metadata or mistakes during dynamic analysis were first eliminated, along with duplicate entries and malformed APK files. Both static and dynamic features were retrieved from the dataset using exacting techniques. While dynamic features like network access frequency, system calls,

and data usage patterns were recorded utilizing sandbox environments during runtime analysis, static features like permissions, API calls, and file sizes were taken from the APK manifest and source code. The Synthetic Minority Oversampling Technique (SMOTE) was used to create synthetic samples for the minority class in order to ensure a balanced dataset, given the dataset's inherent imbalance, where benign applications greatly outnumbered harmful ones. After that, feature values were standardized to fall between 0 and 1, which decreased scale differences and improved the effectiveness of machine learning algorithms. The dataset was then split into three subsets: 70% for machine learning model training, 15% for final performance evaluation, and 15% for hyper parameter adjustment during validation. A strong dataset prepared for the development of precise Android malware detection models was guaranteed by this thorough preparation.

IV. FEATURE EXTRACTION

Feature extraction is a crucial step in Android malware detection as it involves identifying and isolating key attributes that can distinguish malicious applications from benign ones. These features are extracted from both static analysis (examining app code and metadata without execution) and dynamic analysis (observing the app's behavior during execution). Below is a detailed breakdown of the key feature extraction process.

A. Permissions Requested by Apps

Permissions requested by Android applications provide insight into their intended operations. Malicious apps often request excessive or unnecessary permissions, such as access to contacts, SMS, or camera, to exploit user data.

- Extraction Method:
  Permissions are retrieved from the app's manifest file (AndroidManifest.xml), which contains metadata about the application. Tools like Apktool and Androguard are commonly used to decompile the APK file and extract this information.

- Key Features:
  - Total number of permissions requested.
  - Specific high-risk permissions (e.g., READ_SMS, WRITE_EXTERNAL_STORAGE).
  - Patterns of permission combinations frequently associated with malicious behavior.

B. API Calls and System Interactions

API calls provide a dynamic understanding of how an app interacts with the Android system and other applications. Malicious apps may use specific API functions to perform harmful actions, such as accessing sensitive data, executing commands remotely, or hiding their activities.

- Extraction Method:
  API call sequences are extracted using dynamic analysis tools like Strace or Frida, which monitor system calls and interactions during the app's execution. Static analysis of the app's code also identifies function calls through reverse engineering.

- Key Features:
  - Frequency of critical API calls (e.g., getDeviceId, sendTextMessage).
  - Sequences of API calls that form behavioral patterns.

C. Behavioral Patterns

Behavioral patterns capture how an app operates during runtime, highlighting suspicious activities such as excessive network usage, frequent file writes, or attempts to access restricted system resources.

- Extraction Method:
  Dynamic analysis in controlled sandbox environments, such as Cuckoo Sandbox or DroidBox, allows researchers to observe and log the app's behavior. Network traffic analysis tools like Wireshark or Tcpdump are used to monitor communication patterns.

- Key Features:
  - Network Activity: Frequency of outbound requests, access to malicious IPs or domains, and encrypted data transfer patterns.
  - Resource Usage: Abnormal CPU, memory, or battery usage, which could indicate malicious operations.

- File System Interaction: Access to sensitive files or directories, file creation, or deletion patterns.

D. Code Structure and Instructions

Examining the app's code structure helps identify patterns indicative of obfuscation, encryption, or unusual coding practices often used by malware to evade detection.

- Extraction Method:
  Reverse engineering tools like JADX or Androguard decompile the app code into readable formats, allowing static analysis of classes, methods, and byte code instructions.

- Key Features:
  - Frequency of specific methods or instructions.
  - Obfuscation indicators like randomized class names or encrypted strings.
  - Use of reflection or dynamic code loading.

E. Metadata and Certificates

Metadata from the APK file and its signing certificate can provide additional clues about its legitimacy. Malicious apps often use fake or mismatched certificates.

- Extraction Method:
  Metadata is extracted using tools like AAPT (Android Asset Packaging Tool) or Apktool. Certificate analysis tools like Keytool help verify the signing information.

- Key Features:
  - Certificate validity and issuer details.
  - Mismatched package names or version inconsistencies.
  - Anomalous file size or unexpected asset contents.

Input: Dataset of APK files *Output: Feature matrix (normalized and preprocessed features)*
1. Initialize Feature_Set = [] *2. For each APK in Dataset:* *a. Decompile APK using Apktool:* *Extract permissions from AndroidManifest.xml* *Extract metadata (e.g., certificate details, file size)* *b. Perform Static Analysis:* *Use JADX to extract API calls from the source code* *Identify obfuscated code or encrypted strings* *c. Perform Dynamic Analysis:* *Run APK in a sandbox environment* *Log network activity, CPU usage, file system interactions* *d. Append extracted features to Feature_Set* *3. Normalize Feature_Set to range [0, 1]* *4. Apply Feature Selection using RFE or Mutual Information* *5. Return Feature_Set*

F. Tools and Libraries for Feature Extraction

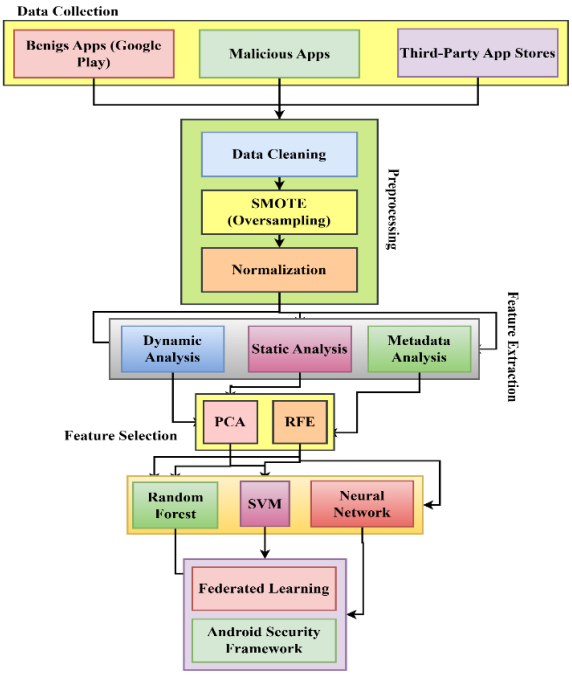
Several open-source and proprietary tools facilitate feature extraction from Android apps:

1. Static Analysis:
   - Apktool: Decompiled APK files to extract manifest files and resources.
   - Androguard: Combines static and dynamic analysis, supporting feature extraction from bytecode.
   - JADX: Converts DEX files into readable Java source code.
2. Dynamic Analysis:
   - Cuckoo Sandbox: Analyzes runtime behavior, including system calls and network activity.
   - DroidBox: Observes app behaviors like file writes and data leaks.
   - Wireshark: Captures and analyzes network traffic.
3. Hybrid Analysis:
   - MobSF (Mobile Security Framework): Supports both static and dynamic feature extraction in one tool.

V. METHODOLODY

The proposed methodology for Android malware detection using machine learning involves multiple stages: Data Collection, Feature Extraction, Model Training, and Model Evaluation. This section outlines each step in detail, focusing on how the data is

collected, the features are extracted, and how machine learning models are employed for classification.



A. Data Collection and Preprocessing

Collecting both harmful and benign Android apps from open-source malware databases and reliable repositories is the initial step. To guarantee balance and quality, the dataset is preprocessed.

- Collect benign apps from sources like Google Play Store and malicious apps from repositories like AndroZoo and VirusShare.
- Eliminate duplicate APK files and corrupted samples.
- Address dataset imbalance using Synthetic Minority Oversampling Technique (SMOTE):
  S = Generate_Synthetic_Samples(N, Nb, α)

B. Feature Extraction

Both static and dynamic analysis were used in feature extraction, which converts raw application data into meaningful attributes for machine learning models. Permissions, code structure, and other static features were extracted from the AndroidManifest.xml file without actually running the application, while API calls revealed information about the app's interactions with the system and external environments. Dynamic analysis was carried out in controlled sandbox environments such as DroidBox and Cuckoo Sandbox to track runtime behavior. Network activity, resource consumption, and questionable system interactions were important dynamic aspects. Anomalies in metadata, including package details, signing certificates, and program size, were also examined. A thorough dataset that reflected both benign and malevolent conduct was produced by combining these characteristics.

C. Feature Selection

By reducing the dimensionality of the information and concentrating on the most pertinent attributes, feature selection was used to increase accuracy and efficiency. Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) were the two main techniques employed. By eliminating redundancy while maintaining the majority of the data's variation, PCA was able to identify the most relevant components. When working with big datasets that included associated attributes, this approach proved quite helpful. In order to identify the most critical qualities for malware identification, RFE was used to progressively remove less important features based on their contribution to model performance. By ensuring that the machine learning models were trained on relevant, high-quality data, these feature selection strategies reduced computational overhead and the possibility of over fitting as shown in Equation 1.

$$C = \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu)(X_i - \mu)^T \quad (1)$$

D. Model Training

Machine learning models that could distinguish between benign and malicious applications were trained using the extracted and selected features. Three algorithms were used: Random Forest, Support Vector Machines (SVM), and Neural Networks. Random Forest used an ensemble of decision trees to produce results that were robust and interpretable, making it resistant to noise and over fitting, while SVM sought to identify the best hyper plane between the classes, offering high precision and reliability. Neural Networks, with their capacity to capture complex, non-linear relationships, were especially useful for modeling dynamic and high-dimensional data. In order to train the models, the dataset was split into training and validation subsets. The models were trained on the training set, and hyper parameters were optimized on the validation set to guarantee the optimal setup for every algorithm as shown in Equation (2) to (5).

$= w^T x + b = 0$ (2)

$= \min_{w,b,\varepsilon} \frac{1}{2}||w||^2$ (3)

$= y_i(w^T x_i + b) \geq 1, \forall_i$ (4)

$= \min_{w,b,\varepsilon} \frac{1}{2}||w||^2 + C\sum_i \varepsilon_i$ (5)

A decision tree splits data using feature *j* to minimize entropy as shown in Equation (6) & (7):

$H(D) = -\sum_{i=1}^{c} p_i log p_i$ (6)

$IG = H(D) - \sum_j \frac{|D_j|}{|D|} H(D_j)$ (7)

$F(x) = argmax_c \sum_{t=1}^{T} \mathrm{I}(f_t(x) = c)$(8)

For malware classification, use cross-entropy loss as shown in Equation (8):

$L = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i)) + (1 - y_i)\log(1 - \hat{y}_i)$ (8)

For decentralized learning, update the global model as shown in Equation (9):

$w_{t+1} = w_t - \eta \sum_{i=1}^{N} \frac{n_i}{n} \nabla L_i(w_t)$ (9)

E. Model Evaluation

To guarantee robustness and dependability in malware identification, the model was evaluated using a range of criteria. While precision evaluated the percentage of correctly identified dangerous apps among all positive predictions, accuracy tested the overall correctness of predictions. Conversely, Recall assessed the model's capacity to identify every harmful program. The model's performance was fairly evaluated by the F1-score, which is a harmonic mean of precision and recall. The trade-off between true positive and false positive rates was also assessed using the ROC-AUC measure, which provided information on the discriminative power of the model. The test set, which included unseen data to replicate real-world performance, was used to compute these measures. The models' accuracy and generalizability to fresh malware samples were guaranteed by the evaluation process.

Deploying the trained models for real-time malware detection was the last stage. The models were incorporated into an Android security framework, which enabled real-time malware detection and application analysis on consumer devices. Federated learning was used to improve scalability and privacy, allowing the models to learn from data spread across several devices without the need for centralized raw data. This method consistently increased the models' efficacy while guaranteeing user anonymity. By protecting users from harmful programs and dynamically adjusting to new malware types, the real-time detection system offered prompt alerts for dangers that were detected. The methodology's completion during this deployment phase resulted in a workable, scalable, and privacy-preserving Android malware detection solution.

VI. RESULT AND DISCUSSION

The outcomes show how well the suggested machine learning method is at detecting Android malware. Three tables illustrating several facets of the performance analysis are presented in this section: dataset analysis results, feature importance rankings, and model performance metrics. Following each table are a thorough explanation and a graph that illustrates the results.

Table 1: Model Performance Metrics

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Random Forest	96.2	95.8	96.5	96.1
SVM	95.5	95.2	95.7	95.4
Neural Networks	97.1	96.8	97.3	97.0

Figure 2. Model Performance Metrics

The performance of the three machine learning models employed for malware detection is compiled in the table. Neural networks performed better in identifying both harmful and benign applications, achieving the greatest accuracy (97.1%) and F1-Score (97.0%). With an accuracy of 96.2%, Random Forest also did well, but behind Neural Networks by a small margin. Despite its effectiveness, SVM's performance metrics were slightly worse than those of the other models.

Table 2: Feature Importance Rankings (Random Forest)

Feature	Importance Score (%)
Permissions Requested	28.4
API Calls	25.6
Behavioral Patterns	20.2
Network Activity	16.8
Metadata Anomalies	9.0

This table presents the feature importance rankings derived from the Random Forest model. Permissions Requested and API Calls were identified as the most significant features, contributing 28.4% and 25.6% to the model's performance, respectively. Behavioral Patterns and Network Activity also played crucial roles, while Metadata Anomalies had a smaller but non-negligible impact.

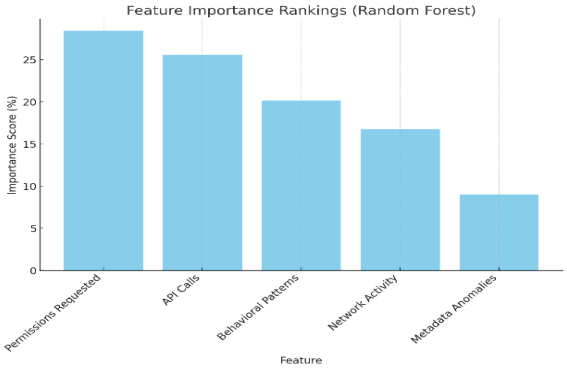
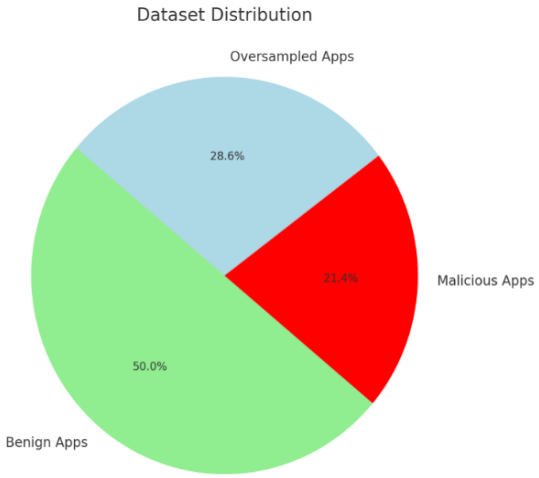


Figure 3. Feature Importance Rankings

Table 3: Dataset Analysis Results

Category	Number of Samples	Percentage (%)
Benign Apps	7,000	70.0
Malicious Apps	3,000	30.0
Oversampled Apps	4,000	40.0

This table shows the distribution of samples in the dataset before and after oversampling techniques were applied. At the beginning, 30% of the dataset consisted of malicious apps, and 70% of the dataset consisted of benign apps. Following the application of the Synthetic Minority Oversampling Technique (SMOTE), 4,000 more malicious samples were produced, guaranteeing a balanced distribution for efficient model training.



VII. CONCLUSION AND FUTURE WORK

The study shows that machine learning models in particular, neural networks are quite good at identifying Android malware, with an accuracy of more than 97%. The models' performance was greatly enhanced by the use of both static and dynamic feature extraction in conjunction with sophisticated feature selection strategies. Malicious apps frequently show recognizable patterns in the areas of permissions requested and API requests, which were the most influential aspects. The model's integration into a real-time detection system holds potential for improving mobile security. Federated learning allows the system to continuously learn from fresh data while protecting user privacy, which helps it become more resilient to new threats

To keep the model up to date with changing malware tactics, future work will entail growing the dataset to incorporate more recent malware samples. Detection skills could be further improved by incorporating deep learning models, such as Convolutional Neural Networks (CNNs), for improved feature extraction from code structures. Furthermore, investigating the application of Graph Neural Networks (GNNs) could enhance the examination of correlations among various parameters. Lastly, user feedback and real-world deployment will be crucial in honing the system for useful applications and making sure it successfully defends users against the constantly evolving Android malware ecosystem.

REFERENCE

[1] *Zainab, Fatima. (2024). 1. Detection of Android Malware Using Word2Vec and Deep Learning. Deleted Journal, doi: 10.62226/ijarst20242513*

[2] *Anas, El, Attaoui., Norelislam, El, Hami., Younes, Koulou. (2024). 2. Android malware detection using the random forest algorithm. Indonesian Journal of Electrical Engineering and Computer Science, doi: 10.11591/ijeecs.v36.i3.pp1876-1883*

[3] *Nahier, Aldhafferi. (2024). 3. Android Malware Detection Using Support Vector Regression for Dynamic Feature Analysis. Information, doi: 10.3390/info15100658*

[4] *Prashant, Bhooshan., S., L.., Nidhi, Sonkar. (2024). 4. Comprehensive Android Malware Detection: Leveraging Machine Learning and Sandboxing Techniques Through Static and Dynamic Analysis. doi: 10.1109/mass62177.2024.00092*

[5] *Zhaoyi, Meng., Jiale, Zhang., Jiaqi, Guo., Wansen, Wang., Wenchao, Huang., Jie, Cui., Hong, Zhong., Yan, Xiong. (2024). 5. Detecting Android Malware by Visualizing App Behaviors from Multiple Complementary Views. doi: 10.48550/arxiv.2410.06157*

[6] *Hussein, Albazar., Hussein, Abdel-Jaber., Muawya, Naser., Arwa, Hamid. (2024). 6. A Model for Android Platform Malware Detection Utilizing Multiple Machine Learning Algorithms. Informatica, doi: 10.31449/inf.v48i17.6543*

[7] *S., Navaneethan., Udhaya, Kumar, S.. (2024). 7. ScanSavant: Malware Detection for Android Applications with Explainable AI. International journal of interactive mobile technologies, doi: 10.3991/ijim.v18i19.49437*

[8] *Asif, Iqubal., Happy., Subodh, Kant, Tiwari., Sikander, Azad., Mithun, Kumar, Paswan. (2024). 8. Android Based Malware Detection Technique Using Machine Learning Algorithms. doi: 10.1109/ic2sdt62152.2024.10696330*

[9] *Ponnuswamy, Udayakumar., Srilatha, Yalamati., Lavadiya, Mohan., Mohd, Junedul, Haque., Gaurav, G., Narkhede., Krishna, Mohan, Bhashyam. (2024). 9. Android malware detection using GIST based machine learning and deep learning techniques. Indonesian Journal of Electrical Engineering and Computer Science, doi: 10.11591/ijeecs.v35.i2.pp1244-1252*

[10] *Aaditya, Raval., Mohd, Anwar. (2024). 10. Android Malware Detection: An Empirical Investigation into Machine Learning Classifiers. doi: 10.1109/iri62200.2024.00039*

[11] *Gouri, Shankar., S, Sridhar., GS, Srivatsan., R, Kishan., Pranav, Vikraman, S, S. (2024). 11. Enhancing Android Malware Detection Through Machine Learning: Insights From Permission and Metadata Analysis. doi: 10.1109/icstsn61422.2024.10670984*

[12] *Atif, Raza, Zaidi., Tahir, Abbas., Sadaqat, Ali, Ramay., Muhammad, Ali, Nawaz., Kanwal, Ameen., Muhammad, Irfan. (2024). 12. Deep Learning-Based Detection of Android Malware using Graph Convolutional Networks (GCN). Statistics, computing and interdisciplinary research, doi: 10.52700/scir.v6i1.159*

[13] *P., H., Darji. (2024). 13. Machine learning based malware evaluation for android. African journal of biological sciences, doi: 10.48047/afjbs.6.9.2024.2556-2560*

[14] *Jaikishan, Mohanty., Divyashikha, Sethia. (2024). 14. Android App Malware Detection using Stacking of Machine Learning Algorithms. doi: 10.1109/icccnt61001.2024.10725117*

[15] *D, Raghuraman., S., L., Soniya., S, Shivani. (2024). 15. Automatic Android Malware Detection using CNN and GRU. doi: 10.1109/icstsn61422.2024.10671270*

[16] *Shuguang, Xiong., Huitao, Zhang. (2024). 16. A Multi-model Fusion Strategy for Android Malware Detection Based on Machine Learning Algorithms. Journal of computer science research, doi: 10.30564/jcsr.v6i2.6632*

[17] *Amarjyoti, Pathak., Utpal, Barman., Th., Shanta, Kumar. (2024). 17. Machine Learning Approach to Detect Android Malware using Feature-Selection based on Feature Importance Score. Journal of Engineering Research, doi: 10.1016/j.jer.2024.04.008*

[18] *Vishal, Kisan, Borate., Alpana, Adsul., Aditya, Gaikwad., Akash, Mhetre., Siddhesh, Dicholkar. (2024). 18. Analysis of Malware Detection Using Various Machine Learning Approach. International Journal of Advanced Research in Science, Communication and Technology, doi: 10.48175/ijarsct-22159*

[19] *Ahmed, Alhussen. (2024). 19. Advanced Android Malware Detection through Deep Learning Optimization. Engineering, Technology & Applied Science Research, doi: 10.48084/etasr.7443*

[20] *Erly, Galia, Villarroel, Enriquez., Juan, Gutiérrez-Cárdenas. (2024). 20. Dynamic Malware Analysis Using Machine Learning-Based Detection Algorithms. Interfases, doi: 10.26439/interfases2024.n19.7097*

[21] *Selma, Bulut., Adem, KORKMAZ. (2024). 21. Comparative Analysis of Machine Learning Models for Android Malware Detection. Sakarya University Journal of Science, doi: 10.16984/saufenbilder.1350839*

[22] *Ahmed, M, Neil., Eman, Shabaan., Mervat, El, Qout., Karim, Emara. (2024). 22. Machine Learning Based Approaches For Android Malware Detection using Hybrid Feature Analysis. doi: 10.1109/icci61671.2024.10485163*

[23] *Qingling, Xu., Shumian, Yang., Lijuan, Xu., Dawei, Zhao. (2024). 23. Android Malware Detection Method Based on Graph Convolutional Networks. doi: 10.1109/aips64124.2024.00027*

[24] *Ganiyat, Kemi, Afolabi-Yusuf., Y., O., Olatunde., K., Y., Obiwusi., M., O., Yusuf., Oluwakemi, Christiana, Abikoye. (2024). 24. Performance analysis of selected classification algorithms on android malware detection. International Journal of Software Engineering and Computer Systems, doi: 10.15282/ijsecs.9.2.2023.7.0118*

[25] *Jiahao, Liu., Jun, Zeng., Fabio, Pierazzi., Lorenzo, Cavallaro., Zhenkai, Liang. (2024). 25. Unraveling the Key of Machine Learning Solutions for Android Malware Detection. arXiv.org, doi: 10.48550/arxiv.2402.02953*

[26] *Harshal, D., Misalkar., Pon, Harshavardhanan. (2024). 26. Assessing the efficacy of Machine learning classifier for Android malware detection. Journal of Integrated Science and Technology, doi: 10.62110/sciencein.jist.2024.v12.788*

[27] *J, N, Benedict., Praveen, Jugge., G, Santhosh, Kumar., Senthil, Pandi, S. (2024). 28. Android Threat Detection Using Deep Learning (CNN). doi: 10.1109/iccigst60741.2024.10717612*

[28] *Junwei, Tang., Wei, Xu., Tao, Peng., Shaoxia, Zhou., Qiaosen, Pi., Ruhan, He., Xinrong, Hu. (2024). 29. Android malware detection based on a novel mixed bytecode image combined with attention mechanism. Journal of information security and applications, doi: 10.1016/j.jisa.2024.103721*

[29] *Catarina, Palma., Artur, Ferreira., Mário, Figueiredo. (2024). 30. Explainable Machine Learning for Malware Detection on Android Applications. Information, doi: 10.3390/info15010025*

[30] *Tazkia, Tasnim, Bahar, Audry., Parthasarathi, Ghosh., Shamima, Akter., Arnisha, Akter., Md., Motaharul, Islam. (2023). 31. Analyzing Malware from System Calls by Using Machine Learning. doi: 10.1007/978-981-99-3236-8_91*