Available online at
https://jcrinn.com/
https://crinn.conferencehunter.com/

Journal of Computing Research and Innovation 11(2) 2026

Journal of Computing Research and Innovation

MalDet: Malware Detector in Mobile Phone using Multilayer Perceptron Integrated with Virus Total

Fakariah Hani Mohd Ali[1*], Izma Khairul Anuar[2], Siti Arpah Ahmad[3]

[1,3]*RIG Cybersecurity and Digital Forensics, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.*
[2]*Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.*

ARTICLE INFO

Article history:
Received 6 November 2025
Revised 1 January 2026
Accepted 15 March 2026
Online first
Published 1 September 2026

Keywords:
Android Malware
Machine Learning
Static Analysis
Permissions
Multilayer Perceptron
Virus Total

DOI:
10.24191/jcrinn.v11i2.633

ABSTRACT

The proliferation of Android applications has led to an increase in malicious software targeting mobile devices especially Android, posing significant security threats to users. This study presents a comprehensive approach to malware detector application in mobile applications using machine learning approach which is multilayer perceptron and virus total integration. By analysing static features extracted from various Android applications, the proposed system classifies applications as benign (safe) or malware (malicious). The methodology incorporates static analysis to extract critical features from the application package files (APKs). The feature focuses on permissions. Machine learning algorithms, Multilayer Perceptron, are employed for feature classification. Datasets chosen which is the combination of malware and benign apps are utilized to train and test the models, ensuring a robust evaluation of their performance. The integration of feature selection and optimization techniques further enhances detection accuracy, with results demonstrating high efficacy in identifying malicious applications. The integration with Virus Total providing an additional layer of verification for each scanned application. This research underscores the potential of machine learning in bolstering mobile security by providing an efficient and scalable solution for detecting Android malware.

1. INTRODUCTION

With the sudden rise in mobile usage, mobile devices, especially Android have grown more vulnerable to malicious applications, commonly referred as malware. Malware is described as software that is intended to harm or exploit any programmable device, service, or network, with the goal of stealing sensitive information, controlling device functions, or causing financial and privacy loss. Anti-virus tools cause their database to need updating from time to time (Kushwana & Gandotra, 2019). One of the promising approaches in dealing with these constraints is machine learning. Machine Learning approaches can identify trends and classify applications based on their behavioural features, permission usage, and network

*Corresponding author. *E-mail address*: fakariah_hani@uitm.edu.my
https://doi.org/10.24191/jcrinn.v11i2.633

activities. If trained on large amounts of data combining both benign and harmful applications, machine learning models can learn to detect malware more precisely and adaptively since they are more adapted to increasing in mobile threats. This research investigated how machine learning improves mobile malware detection through accuracy, adaptability, and real-time analysis. Urmila (2022) stated that every day, over 1 million malware variations are created, according to commercial and scientific reports. Mobile applications downloaded from both official sources, such as Google Play Store, and unofficial sources often lack proper security checks, making it difficult to detect and prevent malicious software that can compromise user privacy, data security, and financial safety. Cybercriminals are continually evolving their tactics, using advanced techniques such as credential theft, malicious advertising, and dynamic code execution, which allow them to bypass conventional detection systems. Limitation of malware detection nowadays is it struggle to adapt in zero-day threats that mimic benign apps until activated. Attacker can target the victim without victim's awareness (Guo, 2023). This creates an urgent need for a proactive, adaptive solution such as the proposed malware detection model using a Multilayer Perceptron (MLP) that can accurately analyze app permissions and detect malicious behavior with improved precision and adaptability to emerging threats. This approach also aims to prevent zero-day malware threats since it can detect threates by learning malicious behavior and pattern from known malicious applications. Since the model generalizes feature relationships rather than relying on fixed signatures, it can identify previously unseen (zero-day) malware that contain similar characteristics. Objectives of this project is to design and develop malware detector using machine learning model and integrated it with VirusTotal for classifying application and apk file as benign or malicious. The effectiveness of the proposed research is evaluated using standard performance metric to assess its accuracy and reliability in detecting Android malware.

2. LITERATURE REVIEW

Recent studies have explored different approaches to Android malware detection, combining static, dynamic, and hybrid analysis with various machine learning and deep learning techniques (Selamat & Ali et al., 2019). Urmila (2022) mentioned that Nivedita and Ananthan (2019) investigated anomaly-based detection in smart devices using machine learning methods such as HMM, SVM, Random Forest, DBN, and CNN. Their approach relied on static behavioral features, including time-series data and event characteristics, to determine whether a device exhibited malicious activity. However, their study did not specify the dataset used, limiting reproducibility. Feng et al. (2025) proposed MobiTive, a performance-sensitive malware detection system that applies deep learning techniques such as CNN and RNN on dynamic features. By analyzing manifest properties, API calls, and binary code, their system defends mobile devices against malware threats in real time, evaluated using the Dalvik dataset. Bakir and Bakir (2023) focused on the feature extraction stage in Android malware detection by employing hybrid methods combining Decision Trees, KNN, LightGBM, and CNN. Their approach analyzed both binary code and image-based features extracted from a self-created dataset. Ahmed et al. (2024) emphasized the use of detection methods and comparative evaluations of Android malware detection solutions. They applied classifiers such as k-Nearest Neighbor, Naïve Bayes, SVM, and Decision Trees to dynamic features, particularly API calls and permissions during execution, using datasets such as MalGenome and CICAndMal2017. Upadhayay et al. (2021) introduced RPNDroid, which integrates ranked permissions and network traffic features to detect Android malware. They applied hybrid approaches using Naïve Bayes, Random Forest, and SVM on datasets collected from Genome, Drebin, and Google Play. Their study demonstrated the effectiveness of combining permissions with traffic analysis in improving detection accuracy. Overall, existing works highlight the potential of machine learning and deep learning in Android malware detection. However, most studies either rely on static features or require extensive runtime monitoring. Few works have attempted to integrate collaborative threat intelligence, such as Virus Total, into detection systems. This gap motivates the proposed system, which combines a multilayer perceptron model with Virus Total integration to enhance detection reliability.

https://doi.org/10.24191/jcrinn.v11i2.633

3. METHOD

3.1 Dataset and preprocessing

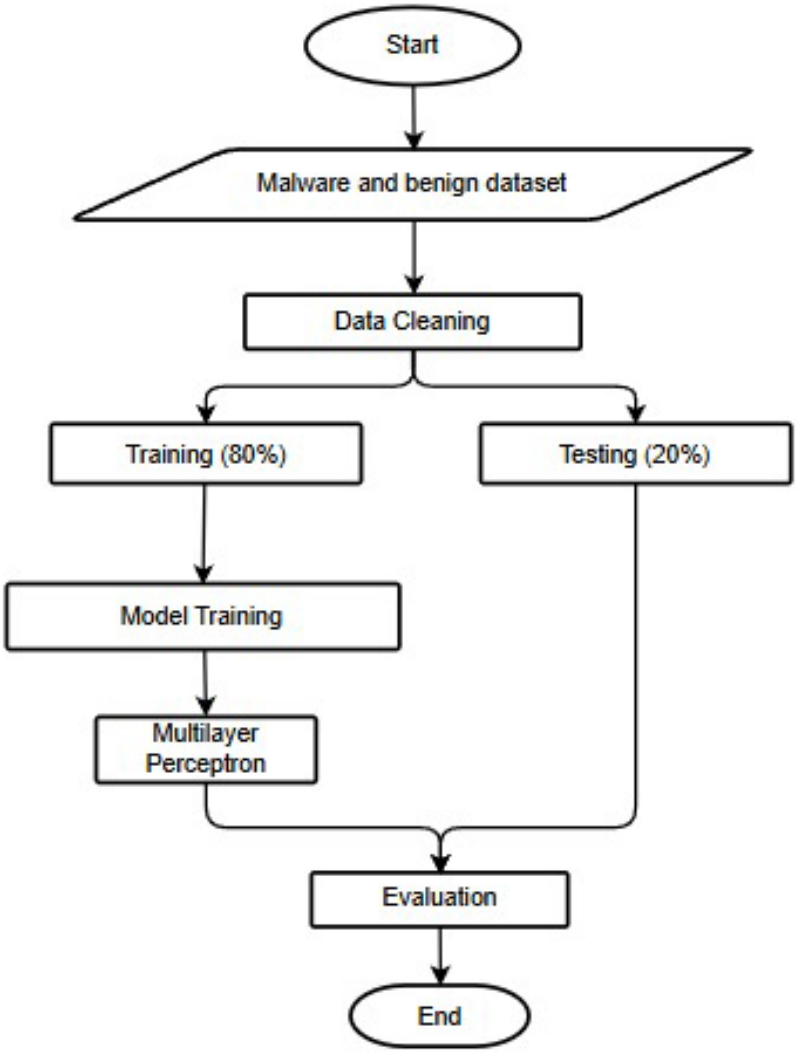

Fig. 1. Flowchart of preparing the Multilater Perceptron model

For this study, we utilized the Android Malware Dataset for Machine Learning 2 developed by Yerima (2018), which contains a total of 15,036 Android applications, including 5,560 malware samples and 9,476 benign applications. Each application is represented as a 215-dimensional feature vector derived through static code-level analysis. These features encompass a broad range of characteristics, including API calls, permission requests, hardware usage, and behavioral patterns, all of which serve as critical indicators for distinguishing malicious applications from benign ones. The feature used for this research can be extracted from each application's AndroidManifest.xml and DEX code. The feature used in this research is utilized as follows:

https://doi.org/10.24191/jcrinn.v11i2.633

(i) Permission

It indicates the level of access requested by the app, example attribute is READ_SMS, SEND_SMS, INTERNET, READ_CONTACTS, ACCESS_NETWORK_STATE.

(ii) Intent Filters

It defines system or user actions that trigger app behavior, it is often used for automatic execution, example attribute, BOOT_COMPLETED, SMS_RECEIVED, CALL, USER_PRESENT.

(iii) API Calls

This type of feature can show sensitive functions executed by the application and reveal attempts to execute commands, collect device info, or obfuscate payloads. Example, Runtime.exec(), getDeviceId(), getSubscriberId(), Base64.decode().

(iv) Components

This type of feature describes how the app interacts internally and externally such as Activities, Services, Broadcast Receivers.

(v) Other Metadata

This type of feature provides contextual information used during static analysis and feature mapping. Such as App package name, version, requested features.

For our machine learning model to work well, it needs numeric labels. So, Label Encoding is needed to encode 'S (malware)' to 1 and 'B (Benign)' to 0. Sum of duplicated value of this dataset is 6865. Although duplicate values usually degrade model quality, in this dataset, duplicate samples are retained. This is because they may represent different apps with the same behavior and removing them could distort the real-world distribution of malware or benign samples. Before training the model, it is important to standardize to ensure all features contribute equally to the model. This prevents features with larger values from dominating the learning process and prevents bias. Many machine learning models, MLP included perform better and faster when input is standardized.

3.2 Model architectures and training

To make sure that the machine learning model learns and generalizes appropriately, the data set is split into a training set and a test set. It is done by utilizing the train_test_split() method of the sklearn module.

(i) Training

80% of the entire data is employed to train the model in the training dataset. In this process, the model learns from the patterns and correlations between the static features and their respective class labels. By repeatedly adapting its internal weights according to this data, the model becomes good at separating malicious and benign programs.

(ii) Testing

In contrast, the test set comprising 20% of the data is an independent set used to project the performance of the model. The data in this subset of the dataset are never presented to the model with the intention of training so that the performance estimates the capability of the model to generalize to new, unseen applications. This is carried out to avoid overfitting, in which case the model can do well on the training data but completely fail on actual examples.

(iii) Model Training

After completing the preprocessing steps, the cleaned and standardized dataset is used to train a machine learning model. In this project, a Multilayer Perceptron (MLP) neural network is

https://doi.org/10.24191/jcrinn.v11i2.633

implemented using TensorFlow and Keras due to its effectiveness in binary classification tasks. Training is capped at 50 epochs at most, although it is possible to train early in the event of early stopping. A 32-batch size determines the number of samples being trained simultaneously when training. A 20% validation split is utilized to track model performance against new training data unseen. Early stopping is added as a callback to train stopping at the best time.

3.3 Evaluation

(i) Accuracy

Measures the overall correctness of the model the percentage of correctly predicted samples. High accuracy means the model correctly identifies both malware and benign apps.

(ii) Precision

Indicates how many predicted malware apps are malware. High precision means the model makes fewer false alarms (false positives).

(iii) Recall

Shows how many actual malware apps the model successfully identifies. High recall means fewer malware apps go undetected (false negatives).

(iv) F1-Score

A harmonic mean of precision and recall gives a balanced view of performance. It's useful when dealing with imbalanced datasets like Drebin.

After training and testing the model, it is then optimized from the basic Keras format to a TensorFlow Lite format with support by the TFLiteConverter. .tflite format is compatible for application on mobile phone. This is the last step of the machine learning pipeline before it is packaged into the mobile application. The Multilayer perceptron tflite model is now ready to use to be integrated in Android Studio.

3.4 Feature extraction

The malware scanning process in the proposed Android application follows a systematic pipeline that applies equally to both installed applications and APK files selected from device storage. Fig. 2 illustrates the flowchart of this process.

https://doi.org/10.24191/jcrinn.v11i2.633

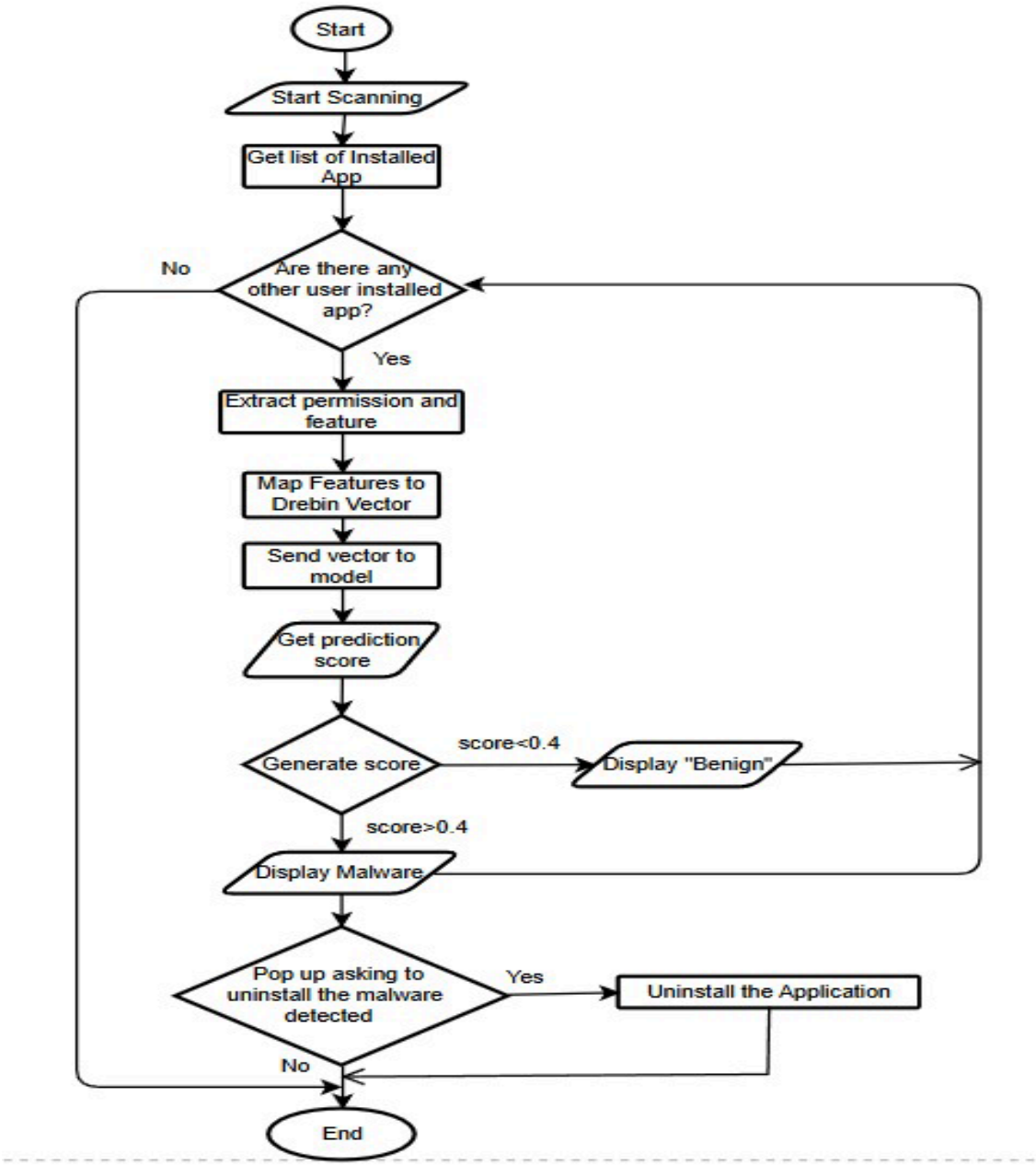

Fig. 2. Flowchart of scan installed app

https://doi.org/10.24191/jcrinn.v11i2.633

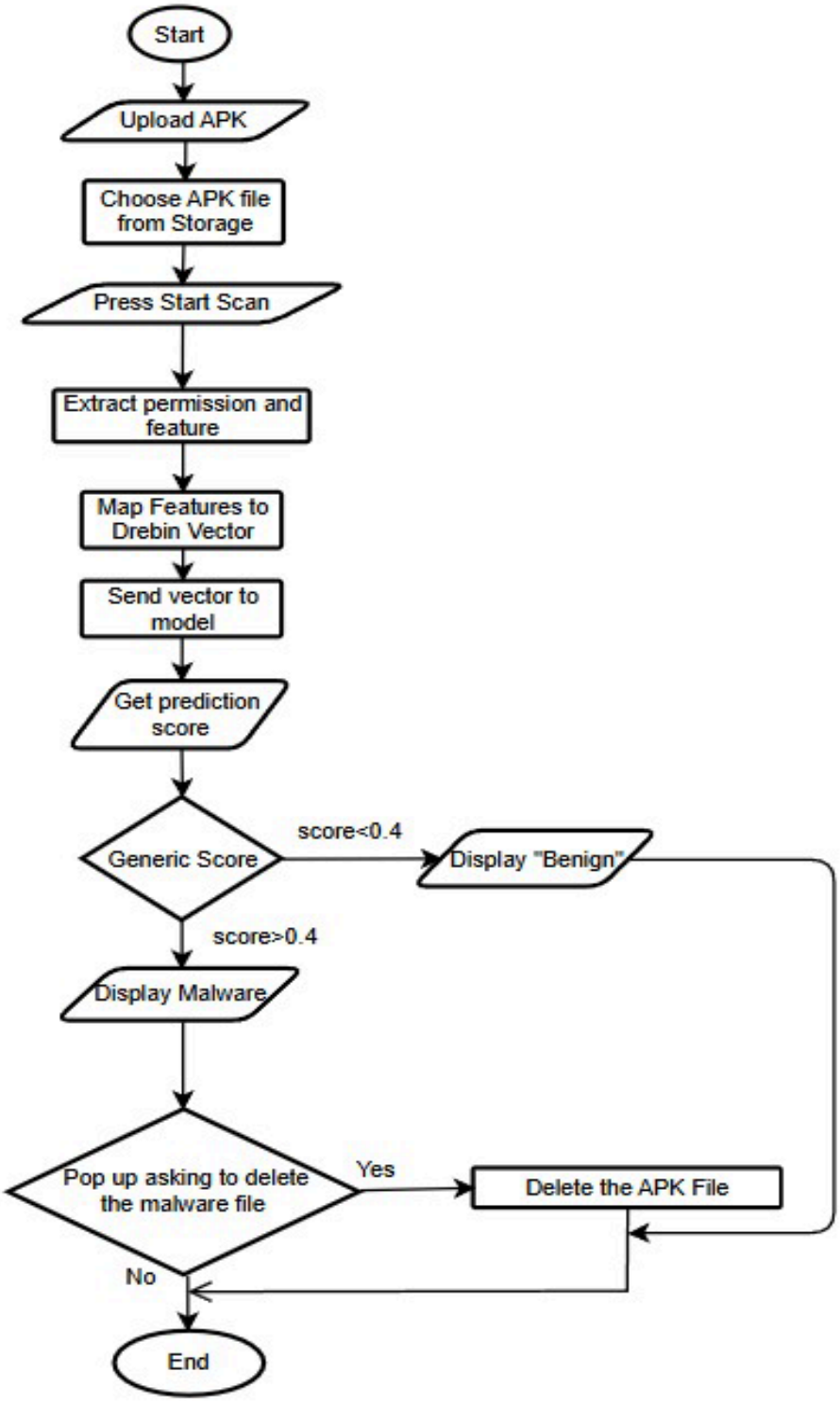

Fig. 3. .apk file

https://doi.org/10.24191/jcrinn.v11i2.633

The procedure begins when the user initiates a scan, either by opening the Scan Phone page to analyze all installed applications or by selecting an APK file through the Scan APK page as in Fig. 3. For installed applications, the system employs Android's PackageManager to retrieve a comprehensive list of apps on the device. For APK scanning, the application uses Android's built-in file picker (getContentLauncher) filtered for MIME type "application/vnd.android.package-archive", allowing the user to select an APK file from device storage. Once the target application is identified, the scanning process extracts relevant static features from two main sources, First, permissions declared in the AndroidManifest.xml, retrieved using PackageManager. GET_PERMISSIONS, and DEX code strings, obtained by unpacking the application's .dex files and applying regular expressions to capture suspicious method calls. These extracted attributes are subsequently mapped into a 215-dimensional feature vector, where each entry is represented in binary form 1 for malware, 0 for benign. The generated Drebin vector is then formatted into a TensorFlow Lite-compatible structure by converting it into a ByteBuffer and wrapping it as a fixed-size TensorBuffer. The vector is passed to the pre-trained Multilayer Perceptron (MLP) model, which processes the input and produces a prediction score between 0.0 and 1.0. A higher score indicates a greater likelihood of the application being malicious. To classify the result, a decision threshold of 0.4 is applied. Applications with scores ≥ 0.4 are labeled as Malware, while those with scores below this threshold are labeled as Benign. If malware is detected, the application prompts the user with an alert dialog containing details of the suspicious features and an alert to uninstall or delete the application. If benign, the result is displayed on the screen as safe. After all scans are completed, the system resets to an idle state, allowing the user to perform subsequent scans if desired. This unified scanning workflow ensures consistent processing for both installed applications and standalone APK files, combining static feature extraction,vectorization, and machine learning classification into a seamless malware detection system.

3.5 Virus Total integration

The scanning process begins when the user presses the Start Scanning button, which triggers the malware detection function to analyze the selected APK file or installed application. First, the system generates the SHA256 hash of the APK, providing a unique digital fingerprint. Using the VirusTotal API, the system checks whether this has already existed in the VirusTotal database. If a record is found, the result is retrieved directly without uploading, saving time and resources. However, if no record exists, the APK is uploaded to VirusTotal via the file scan endpoint, where it undergoes deep analysis by over 70 antivirus engines. Finally, the detection result is retrieved: if no engine flags the app, it is displayed as benign, whereas detection by one or more engines classifies it as malware.

4. RESULTS AND DISCUSSION

We have presented a multilayer perceptron malware detector for android operating system. The evaluation prediction is shown in Table 1.

Table 1. Evaluation result

Accuracy	Precision	Recall	F1-Score
0.987	0.982	0.983	0.982

The dataset was examined for missing values, and it was confirmed that there were zero null values across all features. This ensured that no data imputation or replacement was required, and the dataset was complete for model training. It was found that the dataset contained 6,865 duplicated rows. These duplicates

https://doi.org/10.24191/jcrinn.v11i2.633

were intentionally kept maintaining the natural distribution of the dataset. In malware datasets, repeated patterns are common, and retaining duplicates can help the model generalize better to recurring threats.

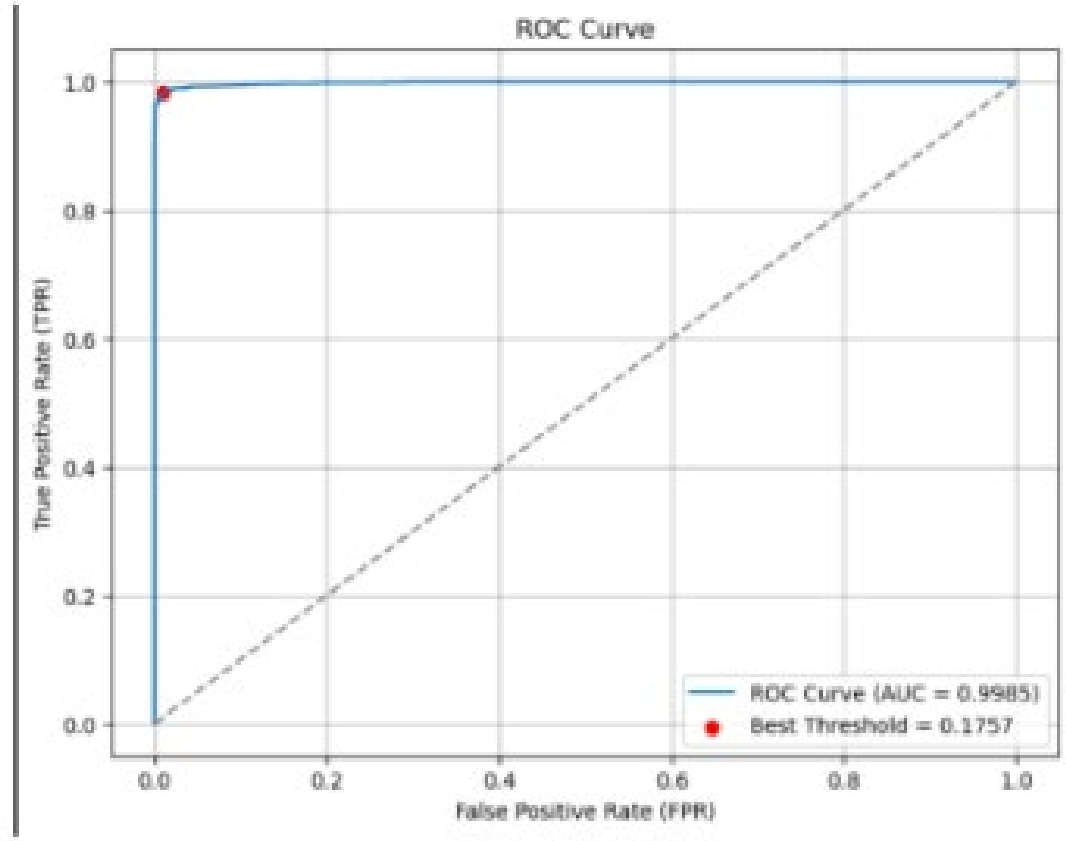

Fig. 4. Threshold value graph

The Drebin dataset consisted of 5,560 malware and 9,676 benign APK samples, leading to class imbalance. Such imbalance can bias the model toward the benign class, which has nearly twice the number of samples. To address this, a probability threshold was applied during prediction to balance detection accuracy and false positives. Fig. 4 shows the predictions using the best threshold of 0.1757, which was selected to optimize performance.

Fig. 5 and 6 present the MalDet application interface for scanning installed applications and external APK files.

https://doi.org/10.24191/jcrinn.v11i2.633

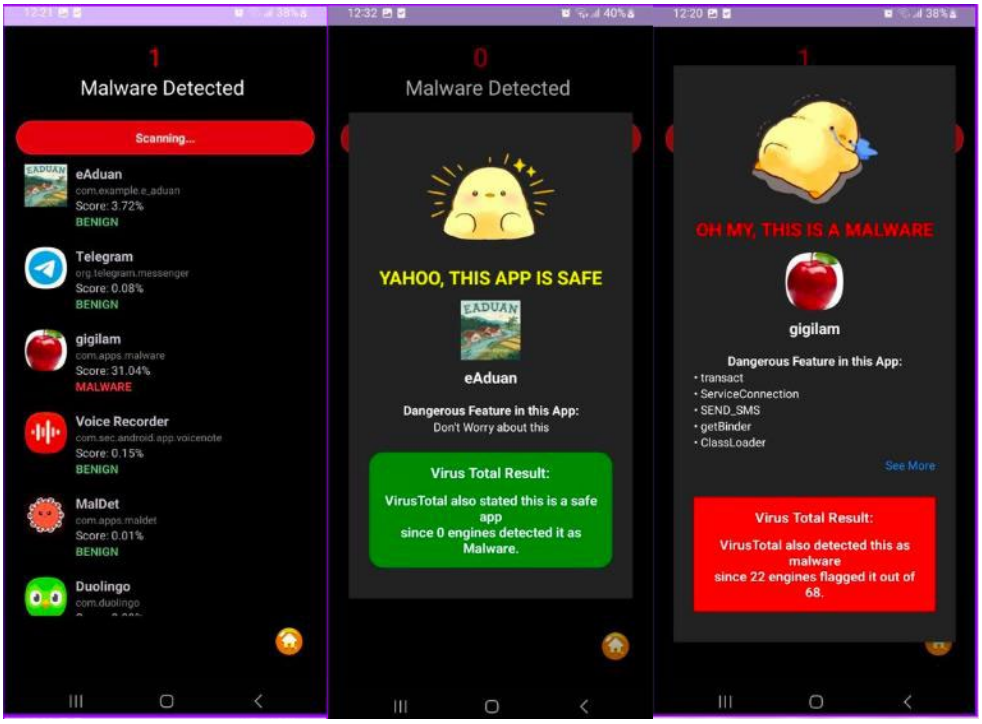

Fig. 5. MalDet interface of scan installed app in android

Fig. 6. MalDet interface of scan installed .apk file in android

https://doi.org/10.24191/jcrinn.v11i2.633

Once the scan is completed, the interface displays the detection results clearly, indicating whether the application is classified as benign or malware. In addition to the local machine learning model, the system also integrates with VirusTotal to strengthen reliability. If VirusTotal shows that zero antivirus engines detect the app, it is labeled as benign, while detection by one or more engines classifies it as malware. This integration ensures that users receive accurate and transparent results directly within the interface.

5. CONCLUSIONS

In conclusion, the Malware Detector in Mobile Phone Using Multilayer Perceptron Integrated with VirusTotalwas successfully implemented and completed all its objectives. The project led to the development of a powerful and precise machine learning model for malicious mobile app detection. The model was efficiently integrated into the MalDet application, which offers users a convenient tool for testing and identifying malware from their android phones. MalDet app is crafted to serve the needs of individuals and organizations by increasing the mobile security and limiting the malware infection options. The app keeps the users proactive in handling potential threats by providing them with real-time outcomes for malware classification and scanning. Proper use of the app increases the confidence of users in mobile security and enables cybersecurity awareness. The process for development included some necessary steps like static APK file analysis, feature extraction such as permissions and API calls, preprocessing to create Drebin-style vectors, and model training in the MLP technique. TensorFlow Lite was employed to execute the trained model on Android. This project provided practical experience in detecting mobile malware, using machine learning, and applying it with Android. It demonstrated that artificial intelligence could be used successfully for enhancing cybersecurity at the user end. Hence, the project was able to achieve all the intended objectives and added to the knowledge of how machine learning could actually be applied to detect mobile malware in the real world. In future work, the system can be extended to incorporate real time detection and dynamic analysis to enhance its applicability against evolving malware threats.

6. ACKNOWLEDGEMENT

This work was supported by Research Interest Group (RIG) Cybersecurity and Digital Forensics, and the Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA (UiTM).

7. REFERENCES

Ahmed, S. R., Mohamed, S. J., Aljanabi, M. S., Algburi, S., Majeed, D. A., Kurdi, N. A., Al-Sarem, M., & Tawfeq, J. F. (2024). A novel approach to malware detection using machine learning and image processing. In *Proceedings of the Cognitive Models and Artificial Intelligence Conference (AICCONF '24)* (pp. 298–302). ACM. https://doi.org/10.1145/3660853.3660931.

Ananthan, T. V., & Niveditha, V. R. (2019). Detection of malware attacks in smartphones. International *Journal of Innovative Technology and Exploring Engineering (IJITEE), 5*. https://doi.org/10.35940/ijitee.A5082.119119.

Bakır, H., & Bakır, R. (2023). DroidEncoder: Malware detection using auto-encoder based feature extractor and machine learning algorithms. *Computers & Electrical Engineering, 110*, 108804. https://doi.org/10.1016/j.compeleceng.2023.108804.

Feng, H., Li, P., Fu, Y., & Li, Q. (2025). Plane positioning error calibration with multilayer perceptron and Gaussian mutation genetic algorithm for visual guidance industrial Cartesian robot. *Measurement, 256*(Part B), 118269. https://doi.org/10.1016/j.measurement.2025.118269.

https://doi.org/10.24191/jcrinn.v11i2.633

Guo, Y. (2023). A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer Communications, 198*, 175–185. https://doi.org/10.1016/j.comcom.2022.11.001.

Kushwana, H., & Gandotra, E. (2019). Permission-based Android malicious application detection using machine learning. In *Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2019)* (pp. 103-108). IEEE. https://doi.org/10.1109/ICSC45622.2019.8938236.

Selamat, N., & Ali, F. (2019). Comparison of malware detection techniques using machine learning algorithm. Indones. *J. Electr. Eng. Comput. Sci, 16*(1), 435-440. https://doi.org/10.11591/ijeecs.v16.i1.pp435-440.

Upadhayay, M., Sharma, A., Garg, G., & Arora, A. (2021). RPNDroid: Android malware detection using ranked permissions and network traffic. In *Proceedings of the Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)* (pp. 19–24). IEEE. https://doi.org/10.1109/WorldS451998.2021.9513992.

Urmila, T. S. (2022). Machine learning-based malware detection on Android devices using behavioral features. *Materials Today: Proceedings, 62*(7, SI), 4659–4664. https://doi.org/10.1016/j.matpr.2022.03.121.

Yerima, S. (2018). Android malware dataset for machine learning 2 [Dataset]. figshare. https://doi.org/10.6084/m9.figshare.5854653.v1.