*International Journal for Modern Trends in Science and Technology*
*Volume 12, Issue 01*, pages 20-29.
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol12issue01.html
DOI: https://doi.org/10.5281/zenodo.18362290

# A Novel Approach for Detecting Android Malware Through Co-Occurrence using Machine Learning

**Bandela Charan[1] | Dr. K. Naga Sujatha[2]**

[1]*Department of Electrical and Electronics Engineering JNTUH University College of Engineering Science and Technology Hyderabad, T.S, India-500085*
*charanbandela123@gmail.com*
[2]*Professor, Department of Electrical and Electronics Engineering JNTUH University College of Engineering Science and Technology Hyderabad, T.S, India-500085*
*kns@jntuh.ac.in*

**To Cite this Article**
Bandela Charan & Dr. K. Naga Sujatha (2026). A Novel Approach for Detecting Android Malware Through Co-Occurrence using Machine Learning. International Journal for Modern Trends in Science and Technology, 12(01), 20-29. https://doi.org/10.5281/zenodo.18362290

**Article Info**
Received: 28 December 2025; Revised: 16 January 2026; Accepted: 20 January 2026.



***KEYWORDS***

*Co-existence,*
*FP-Growth,*
*Machine learning algorithms,*
*Malware".*

**ABSTRACT**

*Traditional approaches to Android malware detection primarily depend on signature-based methods. While these methods have been widely adopted, they fall short when it comes to identifying newly emerging malware variants that are not yet present in known databases. To address this limitation, this project proposes a novel machine learning-based technique emphasizing static features, specifically permissions and API calls, within Android applications. Unlike conventional detection techniques, which typically involve static analysis or runtime behavior monitoring, our approach emphasizes the extraction of meaningful patterns from the static features of APK files. Static analysis alone often lacks comprehensive coverage, leaving security gaps. Therefore, this method is designed to overcome these gaps by investigating the co- existence patterns of permissions and APIs, which are often indicative of malicious behavior when compared to legitimate applications. To implement this, a new dataset will be constructed by analyzing Android APK samples from existing reputable datasets. This dataset will capture various combinations of permissions and API calls along with their frequencies. The Frequent Pattern Growth (FP-Growth) will be employed to identify the most significant co-occurrence patterns among these features, which act as reliable indicators for differentiating between harmful and benign applications. Subsequently, these extracted features will be fed into various conventional machine learning algorithms to evaluate their effectiveness in malware detection. The primary objective is to enhance the accuracy, robustness, and generalizability*

*of malware classification models. Through leveraging the power of feature co-existence analysis and machine learning, this approach is expected to outperform several contemporary state-of-the-art methods. The entire implementation will be carried out using either Windows or Linux operating systems, with Python as the primary programming language for dataset creation, feature extraction, and model evaluation.*

## 1. INTRODUCTION

The smartphone industry has witnessed exponential growth over the past decade, with global sales anticipated to surpass 351 million units by 2024 [1]. Among mobile operating systems, Android leads the market with a user base exceeding 2.5 billion across more than 190 countries [2]. The platform's openness and flexibility have enabled rich user experiences but have also exposed it to significant security threats. Malware developers increasingly target Android, exploiting its open-source nature to create and distribute harmful applications [3]. These threats are often shared via trusted app marketplaces like the official Android marketplace and various external app repositories such as AppChina and AppBrain, where monitoring mechanisms are comparatively weak [6], [13]. Studies have revealed that nearly 50% of apps on AppChina and 22% on Google Play Store exhibited malicious behavior [6].

Conventional malware prevention techniques have depended on signature-based detection systems that match new applications against known malware patterns [7]. While effective against existing threats, these systems fail to identify novel, obfuscated, or repackaged malware [8]. For example, obfuscation techniques can effectively hide malware features, rendering traditional scanners ineffective [9]. As a response to these limitations, researchers have shifted focus toward machine learning (ML)-based solutions. These techniques can identify malicious applications based on patterns in static or dynamic features, even when traditional indicators are hidden or altered [10].

Machine learning-driven identification of malicious software may be broadly divided into static and dynamic approaches. Static analysis examines code without executing the app, focusing on features like requested permissions, API calls, and intents [11], [12].In Conversely, dynamic analysis entails executing and observing the software within a regulated environment to study its real-time activities and detect any malicious activities [13]. However, static features often overlap between benign and malicious apps, making it difficult to distinguish between them. This has led to the incorporation of feature selection techniques to extract the most relevant and non-redundant attributes [14], [15]. Among these, the co-occurrence of characteristics like access rights and API invocations has emerged as a novel indicator of malicious behavior [22].Several studies have proposed enhanced detection methods by integrating multiple feature vectors. Techniques that use combinations of static features, such as permissions and APIs, have achieved notable accuracy. For instance, Tiwari and Shukla used optimized permissions and APIs to achieve detection accuracies above 97% [10]. Similarly, Arp et al. introduced DREBIN, a static analysis tool capable of detecting malware directly on the device, identifying 94% of threats with minimal false positives [13]. Despite these successes in controlled environments, real-world performance remains inconsistent, highlighting the challenge of creating universally effective detection systems [21], [22].

To address these gaps, recent research has focused on developing models that evaluate the interdependence of features rather than treating them in isolation. This includes the use of frequent pattern growth (FP- Growth) algorithms for mining relevant co- occurrence patterns in large datasets [18]. Combining such mining techniques with ML classifiers, including ensemble models like stacking classifiers, offers promising improvements in detection performance.These ensemble approaches utilize diverse classifiers—such as Random Forest, LightGBM, and MLP—and leverage them for enhancing generalization and reducing error rates [19], [20].

In this context, our study proposes a novel AI-driven framework designed to identify malicious software within the Android platform, leveraging the co-occurrence of static features.By integrating optimized feature selection and ensemble learning, the system enhances accuracy and robustness across multiple benchmark datasets including DREBIN, CIC-MALDROID2020, and Malgenome. The ultimate goal is to build a lightweight, accurate, and practical solution that addresses current gaps in Android malware detection systems.

## 2. METHODOLOGY

*A. Proposed Work:*

The model proposed in this research introduces a specialized computational framework for detecting malicious Android applications that emphasizes the importance of co-occurring permissions and APIs in accurately discriminating between trustworthy and malicious software. This model demonstrates superior accuracy compared to existing approaches when evaluated on well-known datasets such as Drebin, CIC_MALDROID2020, and Malgenome [9], [13], [21], with the core objective of enhancing Android security and safeguarding user privacy. To achieve this, a Stacking Classifier has been implemented, combining the predictive performance of Multi-Layer Perceptron (MLP),Random Forest and LightGBM classifiers, which collectively enhance feature extraction and improve overall prediction accuracy. Furthermore, a user-friendly system was developed utilizing the Flask framework in combination with SQLite, supporting secure user authentication through sign-up and sign-in functionalities while enabling users to submit applications for analysis and receive malware detection results efficiently. This comprehensive design makes the system practical, reliable, and well- suited for real-world Android security applications.

*B. System Architecture:*

The architecture of the proposed system begins with a dataset consisting of Android applications, where the attributes are derived based on various co-occurrence combinations of permissions and APIs [4], [5], [7].The dataset is systematically partitioned into two seperate divisions,,namely a training dataset and a testing dataset. The training dataset acts as the foundation for constructing a predictive modeling framework (KNN, SVM, RF, DT, LR and extension- stacking classifier). These models are specifically designed to accurately distinguish safe applications from malicious ones. Once the models are trained,the performance and effectiveness is assessed using the testing set, ensuring the models are capable of accurately classifying previously unseen applications based on the learned patterns of feature co-occurrence.

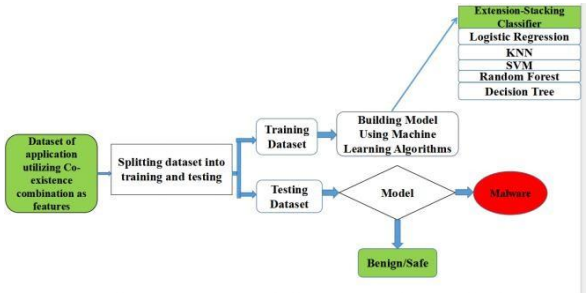


Fig 1 Proposed architecture

*C. Dataset collection:*

*(i)*DREBIN

The Drebin dataset is among the most well-known and extensively utilized datasets for Android malware detection and research, offering a substantial collection of both benign and malicious Android applications. Its broad size and diversity make it a popular standard for evaluating malware detection models, as it supports the development of more robust and generalizable machine learning solutions [9], [13], [25]. In this work, the Drebin dataset has been utilized along with various feature combinations to strengthen the malware detection process. To provide an initial understanding of the dataset, the top five records corresponding to each feature combination are displayed, clearly showing the structure and the number of columns involved in the analysis.

**Table 1: Drebin datasets**

| | SEND_SMS | READ_PHONE_STATE | GET_ACCOUNTS | RECEIVE_SMS | READ_SMS | USE_CREDENTIALS | MANAGE |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | |
| 2 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | |

5 rows × 182 columns

| | transact | onServiceConnected | bindService | attachInterface | ServiceConnection | android.os.Binder | Ljava.lang.Class.getCanonic |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | |

5 rows × 73 columns

| | SEND_SMS | READ_PHONE_STATE | GET_ACCOUNTS | RECEIVE_SMS | READ_SMS | USE_CREDENTIALS | MANAGE_ACCOUNTS |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

5 rows × 110 columns

(ii)MALGENOME

The Malgenome dataset is especially chosen to contain a broad spectrum of malicious software specimens, thereby serving as a significant resource for focused malware analysis and research. Unlike broader datasets, Malgenome offers specialized samples that complement datasets like Drebin by providing targeted examples for the development and assessment of malware identification models.Within this research, the Malgenome dataset has been utilized with various feature combinations to enhance the accuracy of malware detection [9], [17]. To illustrate the structure of the dataset, the top five rows for each feature combination are presented, allowing a clear view of the dataset's attributes and the number of columns involved in the analysis.

Table 2: Malgenome datasets

| | READ_SMS | WRITE_SMS | READ_PHONE_STATE | GET_ACCOUNTS | SEND_SMS | WRITE_APN_SETTINGS | RECEIVE_SMS | USE_ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | |
| 3 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | |
| 4 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | |

5 rows × 182 columns

| | transact | bindService | onServiceConnected | ServiceConnection | android.os.Binder | attachInterface | TelephonyManager.getSubs |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | |

5 rows × 73 columns

| | READ_SMS | WRITE_SMS | READ_PHONE_STATE | GET_ACCOUNTS | SEND_SMS | WRITE_APN_SETTINGS | RECEIVE_SMS |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

5 rows × 110 columns

(iii)CIC_MALDROID2020

CIC_MALDROID2020 is provided by the Canadian Institute for Cybersecurity (CIC) and is known for its size, recency, diversity, and comprehensiveness.

Similarly, we have used CIC_MALDROID2020 dataset with these feature combinations. We are displaying top 5 rows of the data with each feature combination here. and we can see the no. columns present.

Table 3: CIC_MALDROID2020 datasets

| | android.permission.RECORD_AUDIO | android.permission.MODIFY_AUDIO_SETTINGS | android.permission.WRITE_ |
|---|---|---|---|
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 2 | 0 | 0 | |
| 3 | 0 | 0 | |
| 4 | 0 | 0 | |

5 rows × 894 columns

| | transact | onServiceConnected | bindService | attachInterface | ServiceConnection | Ljava.lang.Class.getCanonicalName | android.os |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 1 | 1 | 0 | 1 | 0 | |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | |

5 rows × 70 columns

| | android.permission.RECORD_AUDIO | android.permission.MODIFY_AUDIO_SETTINGS | android.permission.WRITE_SETTINGS | andı |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 0 | |
| 3 | 0 | 0 | 0 | |
| 4 | 0 | 0 | 0 | |

5 rows × 1634 columns

*D. Data Processing:*

Data processing includes converting unprocessed data into valuable information for businesses. Generally,data scientists often handle data by gathering, systematizing, processing, confirming, scrutinizing and transforming it into easily understandable presentations like texts or graphs.Data processing can be accomplished through three primary approaches: manual, mechanical, and electronic. The objective of this process is to enhance the usefulness of information and support more effective strategic choices.This allows organizations to refine their operations and take prompt, evidence- based actions that shape their long-term direction. Computerized information handling systems, supported by digital applications, are instrumental in enhancing efficiency.This capability enables the transformation of extensive datasets, including big data, into actionable insights that support decision- making and process quality control.

*E. Feature selection:*

It refers to the technique for recognizing and retaining the key attributes within a dataset with the aim of boosting the effectiveness of machine learning algorithms.As datasets steadily increase in volume and structural complexity, systematically minimizing the number of features they contain becomes crucial.The main of feature selection is to strengthen the predictive capability of machine learning models while also minimizing the computing resources required for training and evaluation.

Feature selection is an essential stage in the wider field of feature engineering. It focuses on eliminating irrelevant or redundant features, narrowing down the dataset to include only those variables that contribute meaningfully to the model's predictive capability.By performing attribute selection before training, we can ensure that the machine learning model works more efficiently and effectively, rather than depending on the model itself to identify which features matter most.

F. *Algorithms:*

Logistic Regression is a commonly used classification approach designed to forecast the probability of a given input being part of a specific category.This algorithm utilizes a sigmoid (logistic) mapping to transform input features to a likelihood value bounded between 0 and 1.By applying a predefined threshold, the algorithm assigns the data point into classes determined by this likelihood, selecting one of two or more possible outcomes.Throughout the training phase, the learning system optimizes its parameters to represent the data effectively, enabling accurate and reliable classifications.

A Support Vector Classifier (SVC) represents supervised learning method that determines the optimal separating plane to classify data. By using key data points known as support vectors, it maximizes the distance between classes, allowing it to handle both binary and multi-class classification effectively

K-Nearest Neighbors (KNN) is a fundamental supervised learning method employed in both regression and classification tasks. It forecasts results through the identification of K closest instances and using majority voting or averaging. While easy to implement, its accuracy depends on the choice of K and it may face difficulties when handling high dimensional data lacking proper preprocessing [30].

Random Forest represents an ensemble-based supervised learning approach that builds a collection of decision trees to produce outcomes. It operates by training every tree with randomly selected subsets of the dataset and then combining their outputs through averaging or voting. This approach enhances the reliability of results, reduces overfitting, and ensures reliable results for both classification and regression applications.

A Decision Tree is a machine learning algorithm which forecasts results by repeatedly dividing dataset into smaller subsets according to the most important attributes. It forms a hierarchical structure where nodes represent features and branches indicate possible decisions.This clear and interpretable structure renders it effective across diverse classification and prediction tasks.

A Stacking Classifier represents an ensemble-based learning approach which integrates the outputs from several base classifiers, such as Multi-Layer Perceptron (MLP), Random Forest (RF),and LightGBM, to generate a more precise overall outcome.This method takes advantage of the distinct capabilities of individual models to enhance overall performance.In this approach, the base models undergo training on the dataset, after which their predicted outputs are provided as inputs to the meta- learner, which learns the optimal way to merge

these outputs to produce the final result.Stacking is a powerful technique used for improving prediction reliability and is commonly employed in machine learning for various applications.

## 3. RESULTS ANALYSIS

*A. Accuracy:* The accuracy refers to the capability of a test to correctly distinguish between diseased and healthy cases.To determine the accuracy, it is essential to calculate the ratio of accurately identified positive and negative cases in comparison with the overall number of instances assessed. Mathematically, this can be expressed as follows:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (1)$$

*B. Precision:* Precision expresses the share of actual positive outcomes within the set of samples the model designates as positive.Hence, the formula used to compute precision is given by:

$$Precision = \frac{\text{True Positive}}{\text{True Positive +False Positive}}, \quad (2)$$

*C. Recall:* Recall serves as an evaluation measure in machine learning that assesses how effectively a model identifies every applicable instance within a specific class. It is calculated as the proportion between correctly predicted positive cases and the overall number of true positives, reflecting how effectively the model identifies every relevant instance belonging to that category.

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

*D. F1-Score:* F1 Score is a performance measure in machine learning that integrates precision and recall to evaluate a model's effectiveness. It merges precision and recall into one unified value, offering a trade-off between the two. In contrast, the accuracy metric only considers the overall count of correct outcomes generated by a model across the entire dataset

$$F1\ Score = 2 * \frac{Recall\ X\ Precision}{Recall+Precision} * 100 \quad (4)$$

Table (4 to 6) assess how well each algorithm performs based on the metrics of Accuracy, Precision, Recall, and F1-Score.Across all evaluation criteria , the API+Permission: Stacking Classifier (CIC_MALDROID2020) & API: Stacking Classifier consistently outperforms all remaining algorithms. The tables also provide a comparative evaluation of metrics for the other algorithms.

**Table.4 Performance Evaluation Table - CIC_MALDROID2020**

| ML Model | Accuracy | Precision | F1_score | Recall |
|---|---|---|---|---|
| API+Permission : LR | 0.99 | 0.98 | 0.98 | 0.99 |
| API+Permission : SVC | 0.99 | 0.98 | 0.99 | 1.00 |
| API+Permission : KNN | 0.97 | 0.94 | 0.97 | 0.99 |
| API+Permission : RF | 0.97 | 0.97 | 0.97 | 0.97 |
| API+Permission : DT | 0.95 | 0.97 | 0.95 | 0.94 |
| **API+Permission : Stacking Classifier** | **1.00** | **1.00** | **1.00** | **1.00** |
| API : LR | 0.95 | 0.94 | 0.95 | 0.97 |
| API : SVC | 0.96 | 0.94 | 0.96 | 0.97 |
| API : KNN | 0.94 | 0.93 | 0.94 | 0.95 |
| API : RF | 0.94 | 0.94 | 0.94 | 0.94 |
| API : DT | 0.95 | 0.95 | 0.94 | 0.94 |
| API : Stacking Classifier | 0.97 | 0.98 | 0.97 | 0.96 |
| Permission : LR | 0.95 | 0.93 | 0.95 | 0.97 |
| Permission : SVC | 0.95 | 0.92 | 0.95 | 0.97 |
| Permission : KNN | 0.94 | 0.90 | 0.94 | 0.99 |
| Permission : RF | 0.95 | 0.93 | 0.95 | 0.97 |
| Permission : DT | 0.95 | 0.94 | 0.95 | 0.95 |
| Permission : Stacking Classifier | 0.99 | 0.98 | 0.99 | 1.00 |

**Table.5 Performance Evaluation Table - Drebin Dataset**

| ML Model | Accuracy | Precision | F1_score | Recall |
|---|---|---|---|---|
| API+Permission : LR | 0.93 | 0.89 | 0.91 | 0.92 |
| API+Permission : SVC | 0.94 | 0.91 | 0.92 | 0.93 |
| API+Permission : KNN | 0.94 | 0.90 | 0.92 | 0.93 |
| API+Permission : RF | 0.94 | 0.91 | 0.92 | 0.93 |
| API+Permission : DT | 0.94 | 0.91 | 0.92 | 0.93 |

| | | | | |
|---|---|---|---|---|
| API+Premision : Stacking Classifier | 0.95 | 0.94 | 0.93 | 0.93 |
| API : LR | 0.95 | 0.94 | 0.94 | 0.93 |
| API : SVC | 0.96 | 0.96 | 0.95 | 0.94 |
| API : KNN | 0.96 | 0.96 | 0.95 | 0.95 |
| API : RF | 0.97 | 0.97 | 0.95 | 0.94 |
| API : DT | 0.96 | 0.95 | 0.95 | 0.94 |
| **API : Stacking Classifier** | **0.98** | **0.98** | **0.97** | **0.96** |
| Permission : LR | 0.91 | 0.96 | 0.87 | 0.79 |
| Permission : SVC | 0.91 | 0.95 | 0.87 | 0.80 |
| Permission : KNN | 0.90 | 0.92 | 0.86 | 0.80 |
| Permission : RF | 0.92 | 0.96 | 0.87 | 0.80 |
| Permission : DT | 0.92 | 0.96 | 0.87 | 0.80 |
| Permission : Stacking Classifier | 0.91 | 0.97 | 0.86 | 0.78 |

**Table.6 Performance Evaluation Table - Malgenome Dataset**

| ML Model | Accuracy | Precision | F1_score | Recall |
|---|---|---|---|---|
| API+Permission : LR | 0.98 | 0.97 | 0.97 | 0.96 |
| API+Permission : SVC | 0.98 | 0.97 | 0.97 | 0.97 |
| API+Permission : KNN | 0.97 | 0.97 | 0.95 | 0.94 |
| API+Permission : RF | 0.98 | 0.96 | 0.96 | 0.97 |
| API+Permission : DT | 0.97 | 0.94 | 0.95 | 0.96 |
| API+Permission : Stacking Classifier | 0.99 | 0.98 | 0.98 | 0.99 |
| API : LR | 0.98 | 0.97 | 0.98 | 0.98 |
| API : SVC | 0.99 | 0.98 | 0.98 | 0.98 |
| API : KNN | 0.98 | 0.97 | 0.97 | 0.97 |
| API : RF | 0.98 | 0.98 | 0.98 | 0.97 |
| API : DT | 0.98 | 0.97 | 0.97 | 0.97 |
| **API : Stacking Classifier** | **0.99** | **1.00** | **0.99** | **0.98** |
| Permission : LR | 0.93 | 0.91 | 0.90 | 0.90 |
| Permission : SVC | 0.94 | 0.95 | 0.91 | 0.88 |
| Permission : KNN | 0.94 | 0.93 | 0.91 | 0.89 |
| Permission : RF | 0.96 | 0.95 | 0.94 | 0.93 |
| Permission : DT | 0.96 | 0.95 | 0.94 | 0.93 |
| Permission : Stacking Classifier | 0.96 | 0.94 | 0.95 | 0.95 |

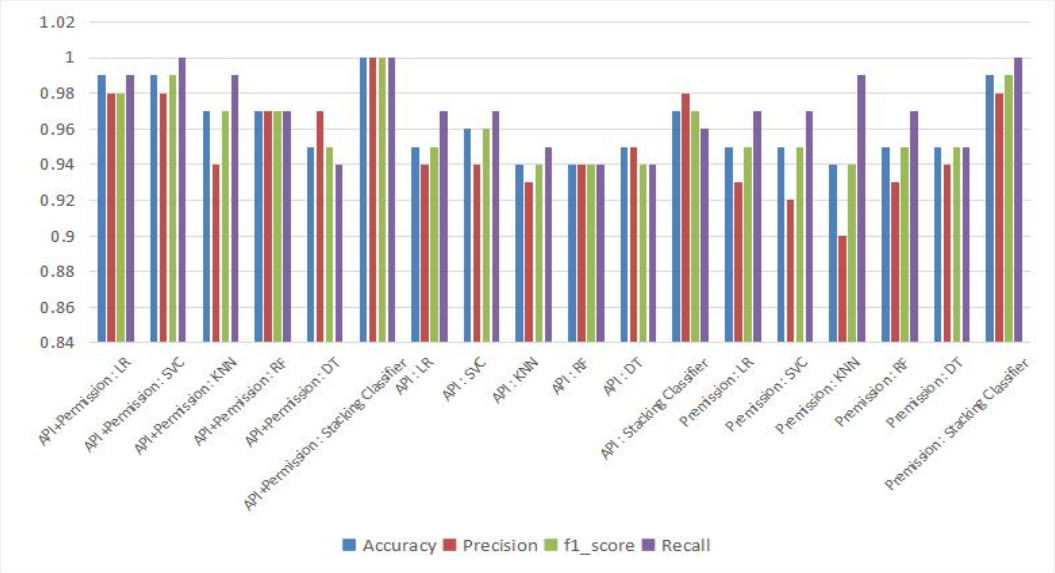


Fig.2 Graphical Comparison of Various Machine Learning Algorithms on CIC_MALDROID2020 Dataset Using the Evaluation Metrics: F1-Score, Recall, Accuracy, and Precision.

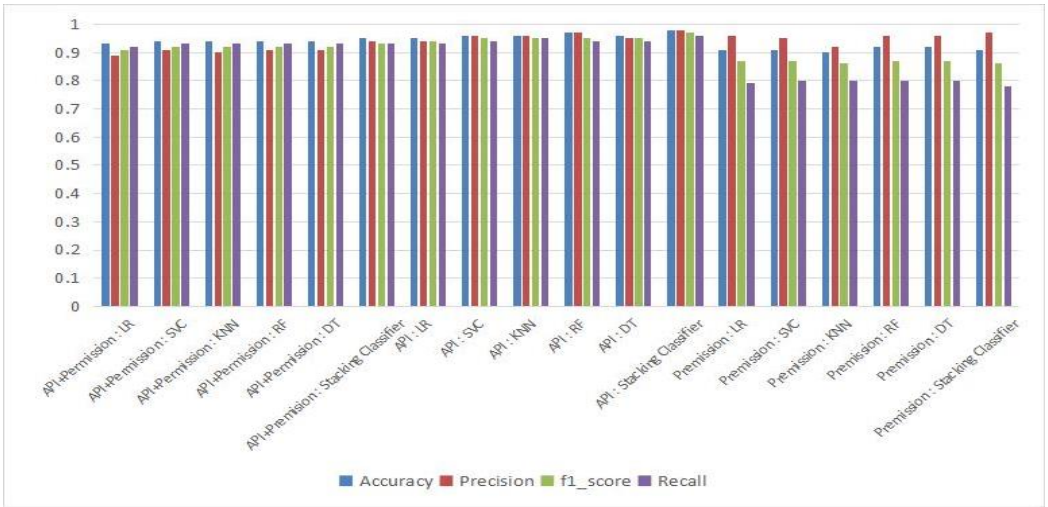


Fig.3 Comparative Graph of Various Machine Learning Algorithms on Drebin Dataset Using the Performance Measures: F1-Score, Recall, Accuracy, and Precision.

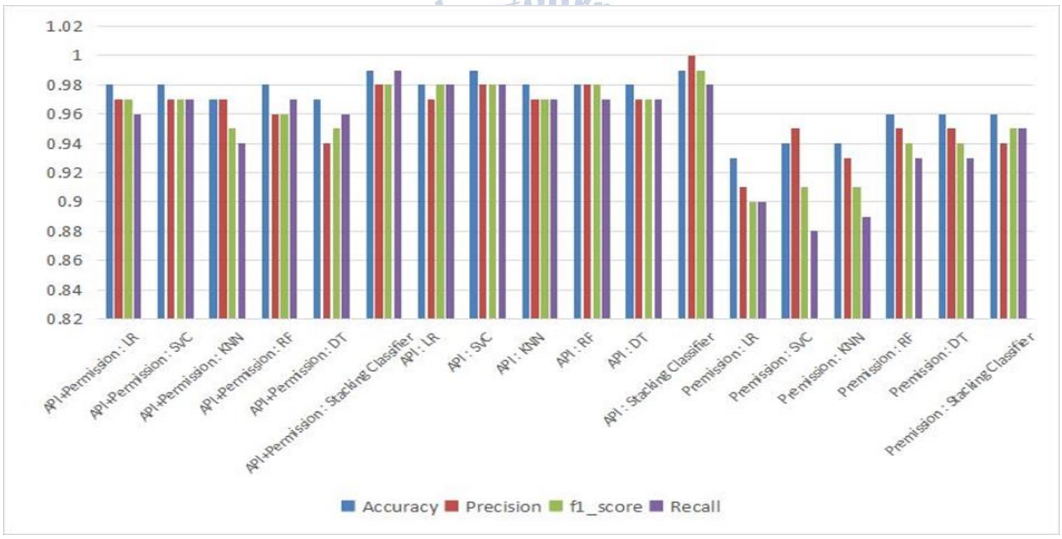


Fig. 4 Visual representation comparing different machine learning algorithms on the Malgenome dataset based on evaluation metrics such as F1-Score, Recall, Accuracy, and Precision

Accuracy is represented by blue, precision by red, recall by green, and F1-Score by purple Fig (2to4). In comparison to the other models, the API+Permission: Stacking Classifier (CIC_MALDROID2020) & API: Stacking Classifier shows superior results across all evaluation metrics, attaining the highest scores. The graphs above provide a clear visual representation of these outcomes.

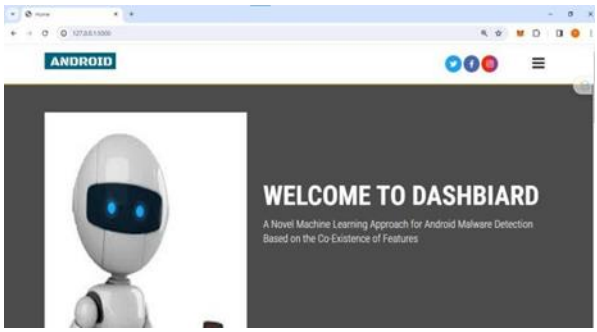


Fig 5: Home page for Android Malware Detection Interfac

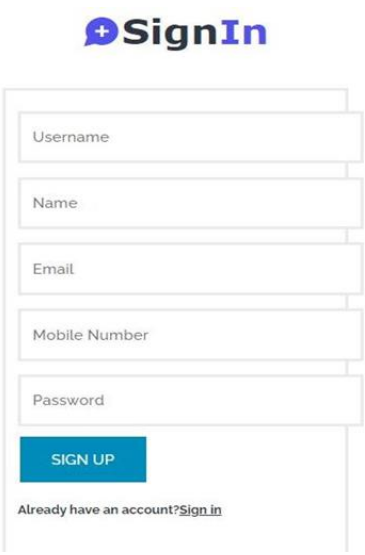


Fig 6: User Sign-in and Registration page


Fig 7: Login page for Android Malware Detection System

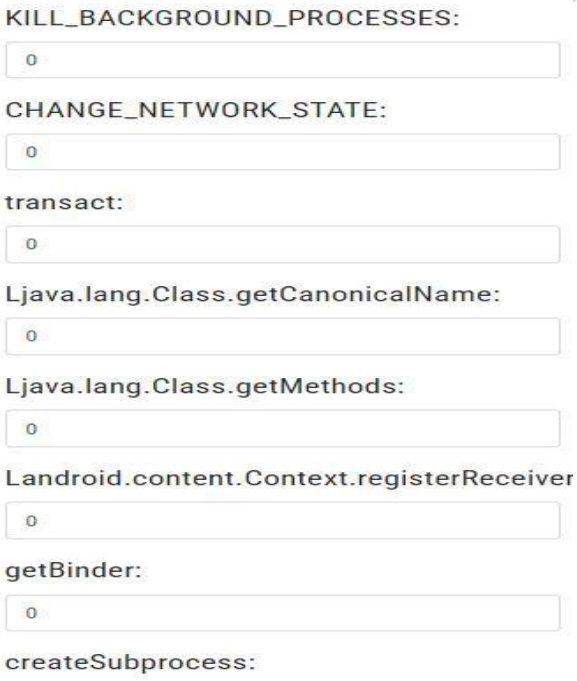


Fig 8: Application Feature-Based User input page


Fig 9 : Predict result indicating Malware Detection

## 4. CONCLUSION

The efficiency of predictive algorithms for detecting Android malware is demonstrated in this analysis. Android Malware is demonstrated in this analysis. These models showed strong capabilities in identifying malicious apps, which is crucial for protecting Android users. The stacking classifier, proved to be more effective than individual algorithms. The approach of combining multiple models enhanced the overall detection accuracy, showcasing the power of ensemble learning.Flask and SQLite are used to build a simple, easy-to-use interface, enabling wider reach and usability. The design supports user testing, validates inputs, and enables smooth model predictions, thereby improving practical usability and promoting adoption. The project highlighted the significance of combining both API and Permission features, whose combination was found to be critical for improving malware detection, emphasizing the importance of considering multiple static features in analysis.The performance of machine learning algorithms varied across different datasets, including Drebin, Malgenome, and CIC_MALDROID2020 [36].This underscores the significance of selecting datasets carefully and understanding for developing accurate detection models.The models maintained an equilibrium between achieving accuracy and reducing false positives. This is essential as it ensures that while detecting malware, legitimate apps are not mistakenly flagged as threats, reducing inconvenience for users. The outcomes of this project have broad implications. Security professionals can use these improved detection techniques to enhance cybersecurity. App developers can better safeguard their apps against potential threats, and end users benefit from increased protection against Android malware, ultimately leading to a safer mobile experience.

## 5. FUTURE SCOPE

Further research could focus on improving the instantaneous detection capabilities of the developed framework system by continuously monitoring and analyzing dynamic features. This would enable the system to respond more effectively to evolving Android malicious software attacks. Exploring techniques to identify the key dynamic attributes for malware detection can lead toward creating faster and more precise models.Approaches to attribute selection, such as mutual information or recursive feature elimination, could be investigated. Extending the system to detect behavioral anomalies in Android applications can provide an additional layer of security. This involves identifying deviations from expected behavior, which could be indicative of malware. [7] As new types of Android malicious software emerge, the framework could be developed to adapt and update its models and detection strategies. Regularly incorporating new threat intelligence and updating the system is essential for long-term effectiveness. Expanding the capabilities of the system to encompass cross-platform malware detection, including iOS and other mobile operating systems, can provide a more holistic solution for mobile security.

### Conflict of interest statement

Authors declare that they do not have any conflict of interest.

### REFERENCES


[1] H. Menear, IDC Predicts Used Smartphone Market Will Grow 11.2% by 2024, IDC, 2021. [Online]. Available. Accessed: Oct. 30, 2022.

[2] D. Curry, Android Statistics, 2022. [Online]. Available. Accessed: Oct. 30, 2022.

[3] O. Abendan, "Fake Apps Affect Android OS Users," Trend Micro Threat Encyclopedia, 2011. [Online]. Available. Accessed: Oct. 30, 2022.

[4] C. D. Vijayanand and K. S. Arunlal, ''Impact of malware in modern society,'' J. Sci. Res. Develop., vol. 2, pp. 593–600, Jun. 2019.

[5] M. Iqbal, App Download Data, 2022. [Online]. Available. Accessed: Oct. 30, 2022.

[6] K. Allix, T. Bissyand, Q. Jarome, J. Klein, R. State, and Y. L. Traon, "Empirical assessment of machine learning-based malware detectors for Android," Empirical Softw. Eng., vol. 21, pp. 183–211, Jun. 2016.

[7] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in Proc. IEEE Symp. Secur. Privacy, May 2012, pp. 95–109.

[8] J. Scott, Signature Based Malware Detection is Dead, 2017. [Online]. Available. Accessed: Oct. 30, 2022.

[9] Q. M. Y. E. Odat, A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence, [Online]. Available. Accessed: Dec. 27, 2022.

[10] U. Shukla, ''An Android malware detection technique based on optimized permissions and API,'' in Proc. Int. Conf. Inventive Res. Comput. Appl. (ICIRCA), Jul. 2018, pp. 258– 263.

[11] Dex2jar—Tools to Work With Android .dex & Java .class Files, 2018. [Online]. Available. Accessed: Oct. 30, 2022.

[12] AndroZoo, [Online]. Available. Accessed: Jul. 30, 2022

[13] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in Proc. NDSS, Feb. 2014, pp. 23–26.

[14] VirusShare, [Online]. Available. Accessed: Jul. 30, 2022.

[15] H. Cheng, X. Yan, J. Han, and C.-W. Hsu, "Discriminative frequent pattern analysis for effective classification," in Proc. IEEE 23rd Int. Conf. Data Eng., Apr. 2007, pp. 716–725.

[16] M. Parkour, Contagio Mini-Dump, [Online]. Available. Accessed: Jul. 30, 2022.

[17] Malgenome Project, [Online]. Available. Accessed: Jul. 30, 2022.

[18] C.-FC.-F. Tsai, Y.-C. Lin, and C.-P. Chen, "A new fast algorithm for mining association rules in large databases," in Proc. IEEE Int. Conf. Syst., Man Cybern., Oct. 1994, pp. 487–499.

[19] A. Lab, AMD Dataset, 2017. [Online]. Available. Accessed: Oct. 30, 2022.

[20] V. Avdiienko, "Mining apps for abnormal usage of sensitive data," in Proc. IEEE/ACM 37th Int. Conf. Softw. Eng., vol. 1, May 2015, pp. 426–436.

[21] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in Security and Privacy in Communication Networks, T. Zia, A. Zomaya, V. Varadharajan, and M. Mao, Eds. Cham, Switzerland: Springer, 2013, pp. 86–103.

[22] E. Odat and Q. M. Yaseen, "A novel machine learning approach for Android malware detection based on the co-existence of features," IEEE Access, vol. 11, pp. 15471–15484, Feb. 2023, doi: 10.1109/ACCESS.2023.3244656.