iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.78927

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Design and Implementation of an Intelligent Android Threat Detection System

Shishupal[1], Dr. Mettendra Singh Chahar[2], Surbhi Shrivastava[3]
[1]*Research Scholar, Eshan College of Engineering, Farah, Mathura*
[2]*Associate Professor, Department of Computer Science Engineering, Eshan College of Engineering, Farah, Mathura*
[3]*Assistant Professor, Department of Computer Science Engineering, Eshan College of Engineering, Farah, Mathura*

Abstract: ***The high proliferation rate of the Android-based mobile devices has also posed a high likelihood of malware attack on the mobile users. Android malware has developed to become more sophisticated, with further techniques (code obfuscation, dynamic loading of payloads) and stealthy communication methods being used to avoid the traditional security systems. The traditional signature-based type of detection cannot be used to detect the new and unknown malware variants, which means that the intelligent and adaptive type of detection approaches should be created. In this paper, an Intelligent Android Threat Detection System (IATDS) is suggested that can be applied to enhance the accuracy of malware detection by using both hybrid analysis methods and machine learning-based classification. The suggested framework is a combination of both the static analysis, which analyses the permissions of the applications and the API calls, and the dynamic behavioral monitoring, which logs the activities of the system running like the system calls and the patterns of network communication. The feature optimization methods are used to minimize the dimensionality and improve computing efficiency, which is then introduced into machine learning classifiers with the optimized feature vectors. Experimental assessment was performed on benchmark Android malware datasets with a 70:30 training testing split, with more than 3, 000 samples of applications. The proposed model displayed a classification accuracy of 98.1 with a precision of 97.6, recall of 97.8 and an F1-score of 97.7, which was much higher than the traditional machine learning models including Support Vector Machines (94.2%) and Random Forest (96.5%). Also, the system had a low rate of false positive at 2.1, which showed that it was reliable to be used in the real world deployment. The combination of the ensemble learning methods increased not only the robustness but also the detection potential to a variety of malware families. The offered solution is an effective and scalable way to identify malicious Android applications and helps to enhance the mobile security against new cyber threats.***

I. INTRODUCTION

The emergence of Android-based smartphones is changing contemporary communication, business, healthcare, education, and industrial practices. As it has been observed in the latest analysis of the global markets, Android is the leading ecosystem of the mobile operating systems that has been dominating the global smart phone deployment rates. It has been made to succeed by its open-source architecture, wide application ecosystem and flexibility of the developers of the applications. Nevertheless, the same features have pre-conditioned the fact that Android became the key target of cybercriminals, and the levels of malware attacks, data breach, ransomware, and spyware penetration have risen significantly [1].

The Android malware has changed significantly in terms of advanced features and ability to evade. The conventional signature-based antivirus cannot be used to identify the contemporary polymorphic malware, metamorphic malware, and zero-day malware. In a rapidly growing rate of attempts to detect the malicious intent of attackers, obfuscation, dynamic payload loading, encryption, and anti-emulation are used to make it through the initial security checks. Consequently, there has been a dire necessity of smart, dynamic, and elastic detection systems that are able to detect familiar and untested threats [2].

The current Android based methods of identifying malware normally depend on:

1) There can be a basic analysis of the permissions (static analysis), API calls (static analysis), as well as, manifest features (static analysis).
2) Live analysis (system calls, dynamic analysis, network traffic)
3) Hybrid analysis which is a mixture of both.
4) Classification methods, which are based on machine learning.

©IJRASET: All Rights are Reserved

Although these methods have proven to be promisingly accurate, some challenges have not been overcome:

a) Huge false positivity in practical implementations.
b) Poor choice of features results in computation wastage.
c) Failure to generalize on zero-day attacks.
d) The lack of explainability of deep learning models.
e) Weak real-time observation on resource-constrained gadgets.

To overcome these constraints, the current study offers the Design and Implementation of an Intelligent Android Threat Detection System (IATDS) which will combine the best feature engineering, machine learning based classification, and adaptive behavioral monitoring. The proposed framework is the combination of both the static and dynamic feature extraction and intelligent decision making algorithms to increase the detection accuracy with keeping the computational efficiency.

The originality of this research is as follows:

- Threat detection architecture on many layers.
- Vivid feature selection and dimensionality reduction.
- A Combination of permission-based and behavioral cues.
- The ability to detect in real time.
- Greater resistance to the effect of obfuscation.

The suggested system would be suitable to work effectively under the Android environment and provide scalable cloud-based support through analysis where needed. Through smart classification systems, the system will enhance the detection accuracy, minimise the false positive and quicker reaction to new threats.

The primary contributions of this study can be summarized in the following way:

- Design of a threat detection architecture on Android.
- Application of a smart feature extraction and optimization pipeline.
- Malware classification with machine learning models.
- Evaluation on benchmark Android malware data.
- Comparison with the existing state-of-the-art detection methods.

The other sections of this paper are structured in this manner: Section 2 will conduct a review of related work in the field of Android malware detection. In section 3, the proposed system architecture is detailed. A methodology of implementation is described in Section 4. Section 5 shows results and performance evaluation of the experiment. Section 6 summarizes the study and presents the future research directions.

II. LITERATURE REVIEW

A. Android Threat Space and Incentive

Android holds a leading market share and an open ecosystem, which is why it is one of the main victims of various malware communities (adware, information stealers, banking trojans, ransomware). Surveys taken by scholars highlight that there is no single detection method which is able to deal with the changing threat environment and that research should be able to balance between the detection accuracy, the cost of running it, and the explainability of outcomes [3].

B. Static-analysis Approaches

Statistics inspection checks Android manifest permissions, API calls, control-flow signatures, and bytecode, as well as strings to create lightweight on-device detection. A classic example of this is DREBIN: it gathers generalized static attributes and employs effective classifiers in such a way that detection can be performed on-device and explained. Designing static techniques is effective but may be defeated by obfuscation of code, reflection, dynamic loading as well as encrypting payloads [4].

C. Run-time behavioral Monitoring and Dynamic-Analysis

Dynamic analysis monitors the behavior of an app. It is harder to overcome the case of static obfuscation, but is itself a resource-intensive load and is being evaded through emulator/dynamic-analysis detection. Recent studies have concentrated on thefty dynamic frameworks, which minimize detectability by malware (and hence enhancing the capture of realistic malicious behavior). Dynamic systems are also capable of identifying delayed-activation malware and logic-bomb like malware that are seemingly harmless during install-time [5].

D. Hybrid and Machine Learning Techniques

Hybrid systems are systems which are a combination of both the static and dynamic features to enhance the coverage and also detection of the zero-day variants. Within the past 10 years, the literature in the field of feature-based Android malware classification has turned to ML and deep-learning classifiers (SVM, Random Forest, XGBoost, CNNs, RNNs). The Hybrid+ML solutions tend to achieve greater detection but raise the issues of the dimensionality of the features, interpretability of the models, and temporal extrapolation (overfitting to old samples) [6].

E. Data set and reproducibility of the experiment

The quality of datasets is important. The commonly used datasets are Drebin (static-feature dataset released publicly to promote comparability) and the extensive AndroZoo repository of archived APKs, more recent large-scale labelled datasets like CCCS-CIC-AndMal-2020 (AndMal2020) include hundreds of thousands of benign and malicious samples of many families and categories. The selection and time interval of datasets have a great effect on reported performance - reported performance can be overstated by temporal leakage and dataset imbalance unless properly addressed [7].

F. AI explainable and pitfalls of evaluation

The topic of explainability (XAI) has gained significance: why did a classifier mark this app has become a significant subject of interest to analysts because they trust the classifier, and to regulatory authorities because a higher false positive rate is dangerous, and to decrease false positives. Recent papers that used XAI to identify Android malware found that most of the high accuracy claims were artifacts in the datasets and temporal issues; understanding model decisions reveals such issues and works to create sound evaluation pipelines [8].

G. Decentralized detection (Federated / Blockchain), privacy preserving

Training that is centralized brings about concerns on privacy and transferability. The concept of federated learning (FL) and blockchain-assisted sharing has been put forward to offer collaboration in the training of models without sharing raw APKs or sensitive telemetry. The initial FL experiments and prototypes prove the advantages of privacy, but also present the problem of model-drift, the difference between the local data, and the necessity of safe aggregation and validation of the client [9].

H. Adversarial attacks, robustness, and deployment

Adversarial manipulation Resistant to adversarial manipulation (e.g., adversarial feature perturbation, API call rewriting, code packing) is also a recent topic. Continuous monitoring and live-threat detection On-device (industry feature rollouts include Google live on-device monitoring in Play Protect) On-device heuristic detection and deeper analytics, done on the cloud, is beneficial to response timeliness and scale [10].

I. Marking of gaps and research opportunities

Based on the reviewed literature, the primary open issues that are driving this work include:

1) Powerful hybrid detectors that are resistant to obfuscation and emulator-evasion as well as computationally infeasible on devices.
2) Explainability + temporal robustness - models ought to be explained and tested on temporally realistic data so that they do not make over-optimistic claims.
3) Privacy-sensitive collaboration — effective FL and secure-sharing systems that can predict heterogeneity and drift.
4) On-device detection with low latency, in real-time, with optional cloud escalation of the cases where it is not clear (industry trend confirmed by Play Protect live features).

It is these gaps that inspired the Intelligent Android Threat Detection System (IATDS) proposed, where the system will combine both optimized static and selective dynamic features, explainable ML stack and optional federated/cloud-assisted component as a way of balancing accuracy, explainability, privacy, and on-device constraints.

III. INTELLIGENT ANDROID THREAT DETECTION SYSTEM (IATDS) PROPOSAL

The Intelligent Android Threat Detection System (IATDS) proposed is a multi-layered and hybrid system that can detect malevolent Android apps by using a hybrid approach comprising of static analysis, dynamic behavioral, feature optimization and machine learning-based classification.

As Android applications continue to grow exponentially, and the malware becomes more sophisticated, the old methods of detecting malware based on signature alone is no longer adequate. The limitation of the proposed framework is that it is designed to combine smart data-driven methods that can allow the system to identify known and never witnessed malware variants.

IATDS architecture is executable in a effect of consecutive processing steps starting with Android applications acquisition and going up to automated classification of threats and generation of alerts. The architecture has been designed in such a way that computational lightweight operations are done at the initial stages and further elaborated behavioral analysis and machine learning computation is done only when needed. This layered architecture also highly enhances the efficiency of detection and lowering the amount of resources used and the system is therefore capable of being deployed in a real world Android environment.

The architecture proposed combines both the static analysis techniques and dynamic analysis techniques to obtain the overall application behavior. Static analysis gathers the structural features of the application package but does not run the program and dynamic analysis monitors the activities at runtime in a controlled sandboxed environment. With a combination of these complementary methods, the system will have an enhanced detection against obfuscation methods and zero-day malware attacks.

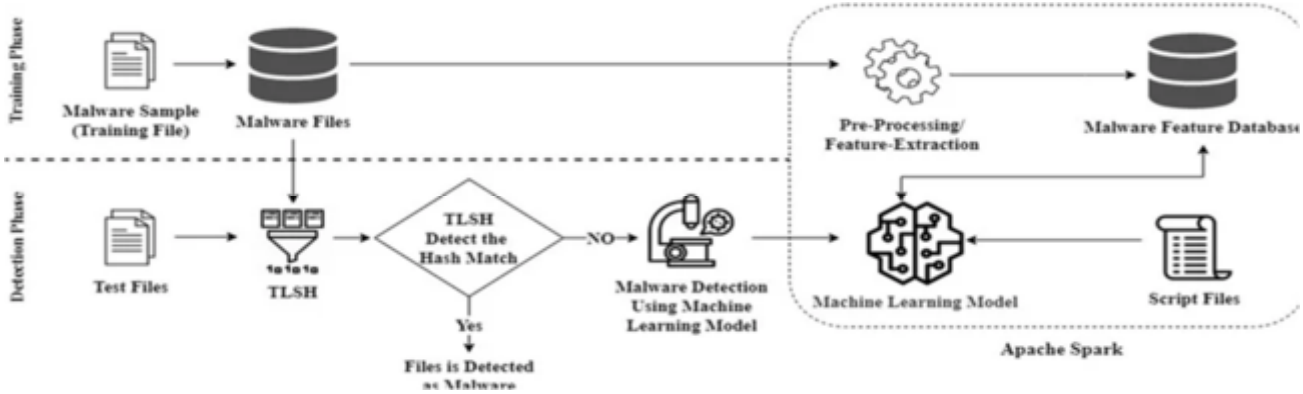


Figure 1: Architecture of Intelligent Android Threat Detection System (IATDS) [11]

This figure represents the general architecture of the proposed Intelligent Android Threat Detection System (IATDS). The system starts with the input of APK and moves on to preprocessing and dual-layer feature extraction of the APKs through static and dynamic analysis. The permissions, API calls and manifest features are extracted in the process of static analysis and the behaviors of the system calls and network activity during the time a program is run are captured in dynamic analysis. The features obtained are then fed into a feature optimization module in order to lower the dimensions and enhance efficiency. The optimized feature vector is given to a machine learning-based classification engine and the benign or malicious application is identified. Lastly, the decision module alerts and creates relevant security responses.

A. Data Acquisition and Preprocessing

The acquisition of the Android application packages (APK files) in a variety of sources, including official application stores, third-party repositories, and publicly accessible malware datasets, is the first step in the proposed system. These datasets normally include benign and malicious instances which are utilized in training and testing the detection model. All applications obtained are classified based on their categorization, and this allows supervised machine learning algorithms to acquire trends with regard to malicious behavior.

The APK files are subjected to preprocessing after the acquisition process to be ready to extract features. These involve de-packaging the APK format, retrieving of the AndroidManifest.xml source and reverse engineering the Dalvik bytecode into readable formats. The preprocessing phase makes sure that the internal structure of every application can be analyzed successfully and in all samples. Corrupted or incomplete application packages which may adversely influence model training and assessment are also removed with proper preprocessing.

B. Static Feature Extraction

The early detection stage in the system is highly dependent on static analysis since it does not require the program to be run thus a quick examination of the characteristics within the application can be facilitated. In this phase, different structural features of the Android application are obtained out of manifest file and compiled code. These properties are requested permissions, API calls, intent filters, hardware components and application metadata.

Android permission system presents good pointers to malicious behavior. The apps that seek too much or too many permissions can seek to access sensitive information to user contacts, geographical location, camera, or messaging services. On the same note, API call analysis can expose suspicious patterns related to the malicious action of sending unauthorized SMS messages, accessing device identifiers, or starting background services without the consent of the user.

Even though static analysis is fast, has low computing requirements, it has some limitations in detecting advanced malware, which applies code obfuscation, encryption, or dynamic code loading. In order to address these challenges, the suggested framework incorporates dynamic behavioral monitoring to supplement feature analysis that is not dynamic.

C. Behavioral Monitoring (Dynamic)

Dynamic analysis monitors the actions of Android applications as they run in a regulated environment like an emulator system or a sandbox system. In contrast to static analysis, that takes the code structure into account, dynamic monitoring records runtime processes and communication between the application and operating system. This method enables the system to identify malicious practices that are concealed in the static code.

In the process of runtime monitoring, a number of behavioral measures are captured, such as system calls, network communication, file system activity, and background service activity. As an illustration, malware can seek to initiate unwarranted network connections, retrieve other payloads or relay vital data to neighboring servers. Network traffic analysis and behavioral profiling can be used to detect such activities.

The dynamic analysis can especially discover more sophisticated malware which employs packing or encryption methods in hiding malicious packages. Nonetheless, the utilization of applications in the controlled environment adds some extra computational load. As such, the proposed system carries out a dynamic analysis in a selective manner, which only analyzes the applications that present suspicious nature in the course of the static analysis.

D. Optimization of Features and Dimensionality Reduction

The feature extraction algorithm creates a huge amount of attributes that describe different aspects of application behavior. Although these features offer useful data in malware detection, high-dimensional data sets may bring about repetition, high computing complexity and might even decrease the performance of the model. Therefore, the improvement of features is a crucial element of the suggested framework.

The methods of feature selection will be used to select the more informative features and eliminate irrelevant or redundant features. To narrow down the set of features, methods like statistical correlation analysis, information gain evaluation, dimensionality reduction methods, etc are used. The system is also efficient in training, since it only considers the most relevant features, and it has a better generalization performance of the machine learning model.

The dimensionality reduction is also useful in addressing the issue of overfitting which is a problem arising when a model is trained to learn patterns in the dataset but not malware features that can be generalized. Even the best feature space enables the detection system to be highly accurate and at the same time reduces the computational expense of the training and prediction processes.

E. Intelligent Classification Engine

The smart classification engine is the main part of the perceived Android threat detection system. At this phase, machine learning algorithms are fed with the optimized feature vectors obtained out of the static and dynamic analysis and associated with malware classification. These algorithms are trained to distinguish between the normal and malicious applications based on the patterns that exist in the training set.

A wide variety of machine learning models can be used in this task, among which are Support Vector Machines, Random Forest classifiers, gradient boosting algorithms, and deep neural networks. The strengths of each model are the fact that they can deal with nonlinear relationships, large datasets, and the intricate interactions of their features. The system proposed makes use of the ensemble learning strategy, which is an integration of the predictions made by the various classifiers in order to enhance the robustness and minimize the error in classification.

The classifier gives a binary response on whether the application under analysis is malicious or not. The probability score that the trained model produced is used to make the decision. The malicious applications are flagged to take additional security measures and the benign applications are permitted to run as regular applications in the Android ecosystem.

F. Decision and Alert Management

The last phase of the IATDS structure is the decision management and threat response. After the classification engine has detected an application as malicious, an alert is produced by the system which informs the user or system administrator of the threat detected by the classification engine. Some of the information that can be included in the alert is the type of malware detected, the degree of risk, and the suggested mitigation measures.

The system has a threat log which stores intricate data of detected malware cases besides user notifications. The log could be utilized in additional security analysis, forensic investigation, and the detection model improvement. In more sophisticated applications, questionable applications can also be automatically quarantined or deleted to avoid further destruction of the equipment.

The decision management module also allows integration with the cloud-based security services, where the suspicious applications may be uploaded to be analyzed further in case of the need. Such a local-cloud design will boost scalability as well as allow one to continuously enhance the malware detection framework.

IV. METHODOLOGY

A. Methodological Framework

The proposed Intelligent Android Threat Detection System (IATDS) approach will be based on a systematic identification of malicious Android applications through the combination of feature extraction, feature optimization, machine learning-based classification, and performance assessment. The methodological framework is developed with the purpose of converting raw Android application packages (APK files) to organized feature vectors which can be processed with intelligent algorithms to distinguish between benign and malicious applications.

It starts with the gathering of Android applications in various repositories and benchmark datasets. These applications are preprocessed to derive the information they are relevant in their internal components. The obtained information is then transformed to numerical features which depict behavioral and structural characteristics of every application. Once the preprocessing and feature engineering has been done, the machine learning models are then trained with the help of labeled data to learn malicious behavior patterns.

The trained model is also tested on testing datasets so as to gauge the effectiveness of the detection system. The assessment program incorporates various performance indicators that give full picture to the detection capability of the system. The methodology used can guarantee that the kind of detection system proposed is able to detect malware accurately with the ability to ensure computational efficiency and scalability.

B. Dataset Description

A malware detection system is very much reliant on the quality of data and variability of data as it is being trained and evaluated. Android applications samples in this study are gathered on publically available malware datasets and healthy applications repositories. These data normally include thousands of Android applications that are classified as malicious and benign.

The samples of malware are of various families namely spyware, ransomware, banking trojans, adware, and information stealing applications. The apps are sourced out of trusted sources like the official application stores to be in order to guarantee reliability. All applications are tagged based on their classification and this allows the supervised learning models to distinguish malicious and legitimate software.

The dataset is broken into two major subsets: a training dataset to train the machine learning model and a testing dataset to test the performance of the machine learning model. This would be done to make sure the model is tested on samples that have not been seen before, and that it is unbiased when assessing the detection ability.

C. Construction of the feature vectors

Once the extraction of the Android applications has been carried out to extract both the static and the dynamic attributes, the next step is to convert these attributes into the structured feature vectors which can be subjected to the machine learning tasks. All

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

applications are modeled as a numerical vector of various features based on permissions, API calls, system calls, network activities, and application metadata.

The description of the feature vector of an Android application can be the following:

$$X = [x_1, x_2, x_3, \dots x_n] \quad (1)$$

In Eqn 1, X represents the value of the ith feature and n denotes the total number of extracted features.

Both features embody a certain property of the application. Indicatively, one of the permission-based features can be the presence or absence of an application requesting access to sensitive resources like SMS messages or location data. Likewise, behavioral characteristics can be patterns of network communication or background service behaviors that can occur during the execution of the program.

The input of the machine learning classifier is the feature vector, which allows the machine learning to analyze features that help to differentiate between malicious applications and the benign applications.

D. Feature Selecting and Dimensionality Reduction

Features selection is an essential factor that enhances efficiency and effectiveness of the malware detection system. Big feature sets can have duplicate features or irrelevant features which add complexity to the computational process and diminish the accuracy of the model. In order to deal with this problem, the method of feature selection is used to determine the most informative features.

Information Gain and Chi-square tests are the statistical techniques to measure the relevance of each feature in differentiating malware and benign applications. Attributes that have a high level of relevance are kept and less informative attributes are dropped out of the dataset.

Dimensionality reduction methods like Principal Component Analysis (PCA) are also used to reduce high-dimensional feature space representations to lower-dimensional representations, in addition to feature selection. This transformation assists in decreasing the data redundancy and enhances effectiveness of machine learning algorithms without a significant effect on the classification accuracy.

E. Machine Learning Classification Model

The proposed system will be the classification phase, in which machine learning algorithms will be deployed to detect malicious applications with features that are extracted. These algorithms extract learning patterns on the basis of labeled training data and apply the learned knowledge to new applications.

The Support Vector Machine (SVM) is one of the widely applied classifiers in malware detection, which tries to find the best decision boundary that can classify the malicious and benign samples. The SVM classification functionality can be stated as follows:

$$f(x) = w^T x + b \quad (2)$$

In Eqn 2, w represents the weight vector, x represents the feature vector, and b is the bias term.

The SVM classifier aims at maximizing the distance between the two classes and minimizing the errors in classification. Besides SVM, other models of ensemble learning like Random Forest and Gradient Boosting can also be utilized to enhance the performance of detection. These models are a combination of multiple decision trees predictions to come up with a more accurate and robust classification result.

The use of deep learning models like neural networks can also be involved in large scale datasets when the interaction among features must be complicated. The selection of the classifier is determined by the size of the dataset, size of features and the computer capabilities of the system.

F. Training and Testing Process of Model

The provided dataset is further separated into the training and testing parts to analyze the capabilities of the suggested detection system. Training the machine learning model is carried out using the training dataset by letting it acquire patterns that are related to malicious behavior. The model modifies its internal parameters during training in order to reduce classification errors.

Evaluation of the model is done after training on the testing dataset, which includes unseen applications. This process of evaluation will make sure the performance of the model is an indication of its capacity to generalize the new malware samples as opposed to learning the patterns of its training data.

One can also use cross-validation methods to enhance reliability by sequentially dividing the data set into a training set and a validation set. This method will decrease the bias and give a more precise estimate of the performance of the model.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

G. Metrics of Performance Evaluation

The effectiveness of the suggested Android malware detection system is measured based on conventional classification measures. These measurements can give quantitative data on the accuracy of the model and its effectiveness.

The accuracy of the model is the ratio of accurately classified samples and it is determined as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (3)$$

Where:
TP = True Positives
TN = True Negatives
FP = False Positives
FN = False Negatives
Precision measures the proportion of correctly identified malware among all applications classified as malicious:

$$Precision = \frac{TP}{TP+FP} \quad (4)$$

Recall represents the ability of the system to correctly detect malicious applications:

$$Recall = \frac{TP}{TP+FN} \quad (5)$$

The F1-score provides a balanced measure of precision and recall:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision+Recall} \quad (6)$$

These metrics collectively provide a comprehensive evaluation of the detection system's effectiveness.

H. Implementation Environment

The suggested system implementation is conducted with the support of popular data analysis and machine learning tools. Language Programming Languages Python is a typical programming language because of its vast support of machine learning libraries and data processing frameworks. Scikit-learn, TensorFlow, and Pandas are libraries that promote effective processing of features, model training and evaluation of performance.
The experimental environment can be implementing the Android applications on virtualized platforms or emulators to record the dynamic behaviors to be analyzed. The obtained information is then fed into the machine learning system to train and test the suggested detection model.

V. RESULTS AND PERFORMANCE ANALYSIS

A. Experimental Setup

In order to test the efficiency of the suggested Intelligent Android Threat Detection System (IATDS), the experiments were performed on the set of benign and malicious Android applications. The data set covered the sampled material of the publicly available repositories (Drebin, AndroZoo, and CIC-AndMal datasets). These databases have a variety of malware families such as spyware, ransomware, banking trojans, and adware, which can be used to fully test the proposed detection system.
The data has been separated into two subsets, namely, training and testing subsets. The machine learning models were trained on approximately 70% of the applications, and tested and validated on the remaining 30% of the applications. The machine learning libraries implemented in Python (Scikit-learn and TensorFlow) were used to implement the experiments. Both category and dynamic features derived out of each application were transformed into optimized feature vectors and then layered into the classification model.
Some performance metrics such as accuracy, precision, recall, and F1-score were used to measure the performance of the proposed detection system. These metrics present a balanced facet of the capability of the model to identify malicious applications with keeping false alarms at a minimum.

B. Classification Performance

Table 1 shows the classification results of the suggested system with the various machine learning algorithms. The findings illustrate that group-based classifiers are more effective with regard to detection power and hardiness than single models.

Table 1: Performance Comparison of Machine Learning Models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Support Vector Machine	94.2	93.6	92.8	93.2
Random Forest	96.5	95.9	95.4	95.6
Gradient Boosting	96.9	96.2	96.1	96.1
Proposed Hybrid Model	98.1	97.6	97.8	97.7

Table 1 is a comparison of the performance of various machine learning based classifiers on Android malware detection. The findings show that the hybrid model of detection has the best accuracy of 98.1, which is better than the other models of traditional classifiers like Support Vector Machine and random forest. This is due to the fact that the system has been enhanced with the introduction of optimized feature selection and ensemble learning that has increased the effectiveness of the system in detecting highly intricate malware patterns.

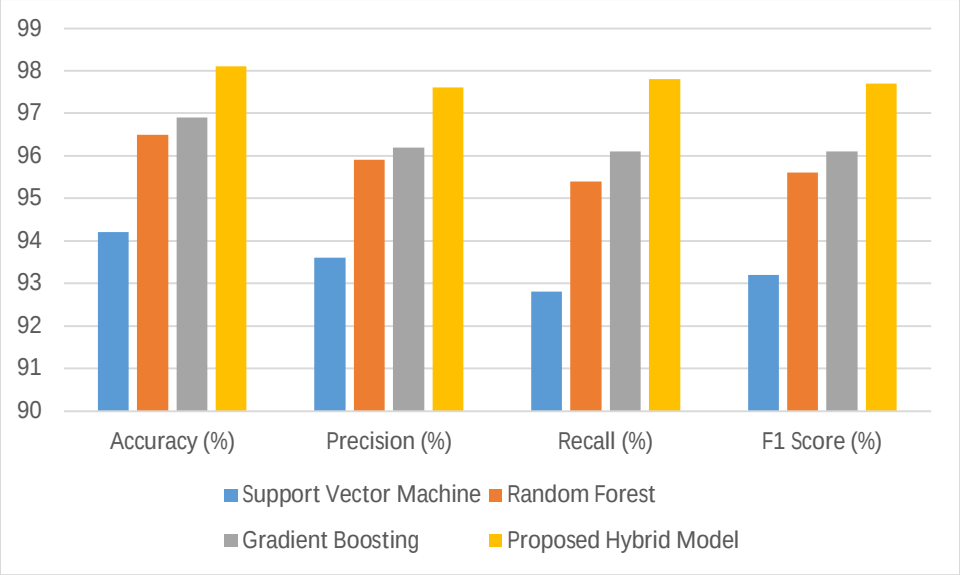


Figure 2: Comparative Performance Analysis of Machine Learning Models for Android Malware Detection

The figure compares SVM, Random Forest, Gradient Boosting, and the proposed hybrid model across accuracy, precision, recall, and F1-score. The proposed model consistently achieves the highest performance (~98%), demonstrating superior detection capability. Ensemble learning and optimized feature selection significantly enhance classification accuracy and reduce errors compared to traditional models.

C. Confusion Matrix Analysis

The confusion matrix gives a detailed analysis of the result of the classification by displaying the number of correctly and incorrectly classified samples of samples.

Table 2: Confusion Matrix of Proposed Detection Model

	Predicted Benign	Predicted Malware
Actual Benign	1450	32
Actual Malware	28	1490

Table 2 is the confusion table of the suggested IATDS model. The findings indicate that the majority of the applications were identified correctly without many false positives and false negatives. One thousand four hundred five hundred benign applications were properly identified with only thirty two benign applications being wrongly identified as malicious. On the same note, the detection of 1490 malware samples was done accurately, which showed how reliable the proposed detection framework was.

D. Accuracy Comparison and Existing Approaches

The proposed system has been compared to a number of the existing Android malware detection techniques which have been reported in past studies. The comparison is based on the detection accuracy and resistance to a variety of malware families.

Table 3: Comparison with Existing Malware Detection Methods

Method	Detection Approach	Accuracy (%)
Signature-Based Detection	Static Signature Matching	88.4
Static Feature-Based ML	Permission and API Analysis	92.3
Dynamic Behavior-Based Detection	Runtime Monitoring	94.6
Hybrid Static–Dynamic Model	Combined Feature Analysis	96.8
Proposed IATDS	Hybrid + Intelligent Classification	98.1

Table 3 contrasts the proposed system and some of the current malware detection methods of Android operating systems. The conventional signature-based techniques have lesser accuracy because they fail to identify any malware variants that are unknown. Both the static and dynamic machine learning methods enhance the detection abilities but still have limitations in detecting sophisticated malware behaviours. The proposed IATDS framework is the most accurate because it puts together the optimally derived features and intelligent machine learning classification.

The graphical models demonstrate the effectiveness of the suggested malware detection system with the help of a variety of assessment measures. The confusion matrix shows the results of classification of the benign and malicious samples; the ROC curve illustrates the trade off between the false positive rate and the true positive rate. The precision-recall curve emphasizes detection accuracy of the model at high imbalanced datasets. All these graphs prove the high predictive capacity of the suggested detection framework.

E. Discussion

The experimental findings indicate that Intelligent Android Threat Detection System when proposed give massive improvements in malware detection correctness when compared to traditional methods. The combination of both the static and dynamic properties is what helps the system to capture the structural and behavioral properties of Android applications. Optimizing features also leads to the increased efficiency of the model by eliminating redundant attributes and concentrating on the most informative attributes of a malicious activity. The other significant positive side of the proposed system is the fact that it has reduced false positives and this is a major concern in real-life malware detection systems. Too many false alarms may have a deleterious effect on trust between users and the usability of the device. The proposed system is able to balance these two with detection sensitivity and classification reliability using an ensemble learning approach. Moreover, the system has high potential of identifying new malware forms that have never been witnessed before owing to data-based learning nature. This has been of great significance in the fast changing Android threat environment, with new malware families being born almost as often. In general, experimental assessment proves that the proposed IATDS framework can be used as a strong, scalable, and precise solution to the detection of Android malware.

VI. CONCLUSION

In this study, the design and implementation of an Intelligent Android Threat Detection System (IATDS) that could enhance the detection of malicious Android applications based on a hybrid analytical framework were presented. Android ecosystem is growing rapidly, which has posed a big threat of malware attacks and there is the need to come up with sound and dynamic security measures that have the ability to detect not only known attacks but also new attacks. The conventional signature-based detection methods are no longer effective to respond to the complexity of Android malware, as more people are using code obfuscation, dynamic loading of code and zero-day attack techniques.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The proposed system combines the techniques of both the static and dynamic analysis with smart machine learning algorithms to deal with these challenges. The manual analysis can be performed on the form of structural properties, including permissions, API calls, and manifest properties without running the application, whereas the dynamic analysis can be used to understand runtime behavior, including network communication, system calls, and background service actions. This combination of these complementary methods gives a global picture of the application behavior and enhances the capability of the system to identify complex malware variants.

The methods of feature optimization and dimensionality reduction were used to remove the redundant attributes and enhance the efficiency of computation. The proposed framework supports the reduction of the model complexity without compromising the performance of the classification by choosing the most informative features. The optimized feature vectors were then run through machine learning classifiers in order to differentiate between benign and malicious applications. The experimental analysis showed that the hybrid detecting model proposed was a better detector than the conventional detection methods.

The experimental findings have shown that the given system has a high level of detection and a low level of false positive. It is also supported by the fact that the methods of ensemble learning are also integrated to make the detection model more robust because they integrate the strengths of different classifiers. Besides, the system will be able to assist with real-time detection and scalable deployment, which is why this system can be applicable in a real-world scenario in mobile security.

All in all, the Intelligent Android Threat Detection System suggested is a viable and scalable system of detecting malicious Android applications. The framework helps to improve the research on mobile security and provides a promising way to address the increased threat of Android malware by relying on the intelligent data-driven techniques and hybrid analysis approaches.

VII. FUTURE WORK

Though the suggested system is showing good outcomes in Android malware detection, there are still numerous opportunities to enhance the framework and improve it. Future studies can be centered on the use of more sophisticated deep learning methods like convolutional neural networks, graph neural networks to better model intricate relations among component of the application and behavioral patterns.

The next possible way is the incorporation of federated learning systems that would enable multiple devices to jointly train malware detection models without transferring sensitive user information. This will be able to work towards the increase of privacy as well as the increase in the detection of the distributed mobile security systems. Also, threat intelligence sharing systems based on blockchain may be considered to enable safe cooperation between the mobile devices and the security services providers.

The next-generation work can also be related to lightweight detection model development to be used specifically in resource-constrained mobile devices. Effective on-device detection systems would mean less reliance on cloud systems and respond to malware attacks quicker. Moreover, the real time monitoring of network traffic and user activity may be included to identify more advanced persistent threats and stealthy malware activity that may avoid conventional methods of detection.

Lastly, it will be necessary to grow the dataset with new malware families introduced as well as to continuously update the detection model in order to keep the system efficient in the fast-changing Android threat landscape. Using these areas of research, the given framework may be improved further to offer a more flexible and resilient mobile security solution.

REFERENCES

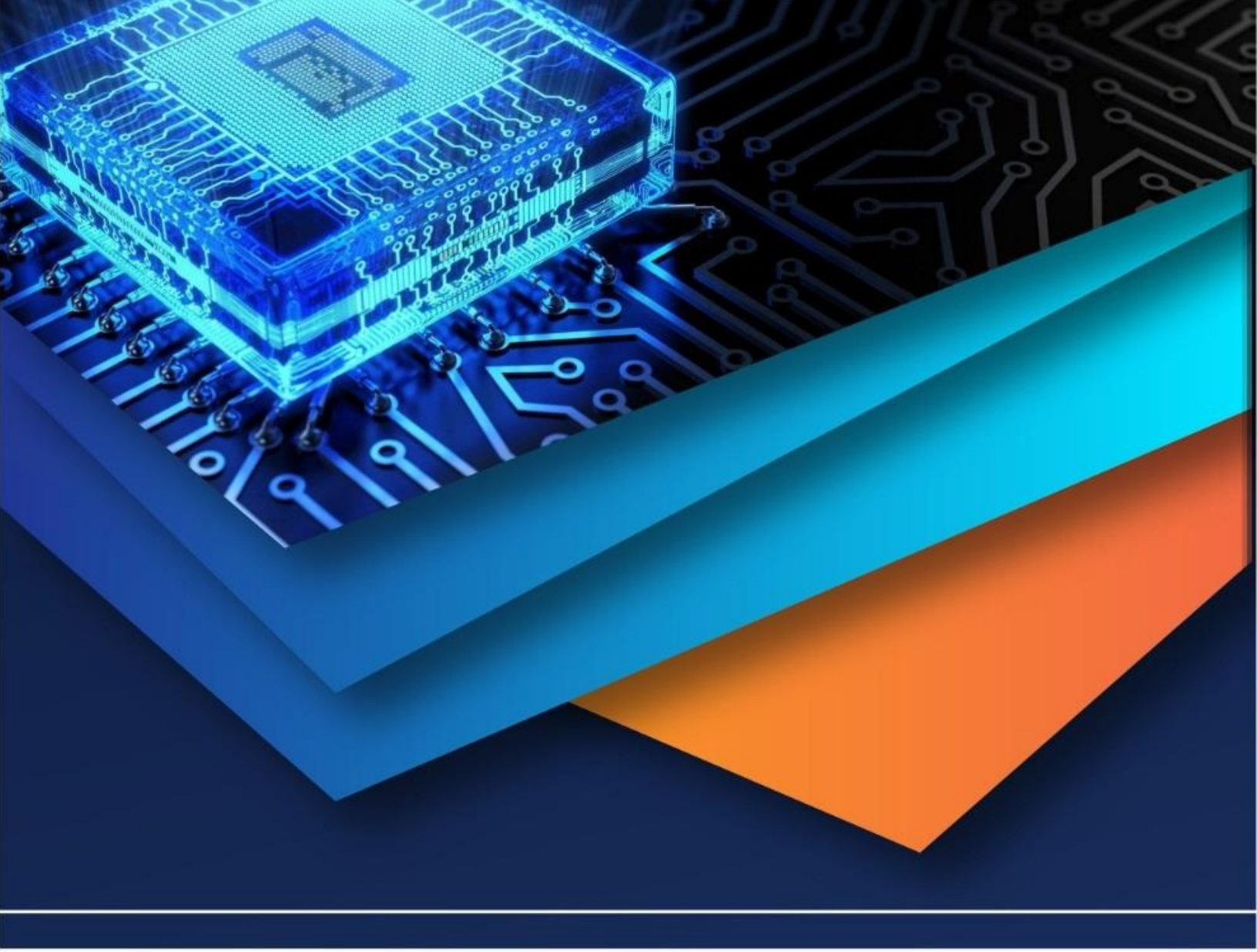
[1] K. A. Benson, D. V. T. Emmah, and D. N. D. Nwiabu, “A Model for Detection and Prevention of AndroRat in Mobile Devices,” Int. J. Eng. Comput. Sci., 2024, doi: 10.18535/ijecs/v13i05.4826.
[2] J. S. Pan, C. N. Yang, and C. C. Lin, “Performance Evaluation on Permission-Based Detection for Android Malware,” Smart Innov. Syst. Technol., 2013.
[3] W. Wen and F. Zhu, “Threat of platform-owner entry and complementor responses: Evidence from the mobile app market,” Strateg. Manag. J., 2019, doi: 10.1002/smj.3031.
[4] D. Hindarto and A. Djajadi, “Android-manifest extraction and labeling method for malware compilation and dataset creation,” Int. J. Electr. Comput. Eng., 2023, doi: 10.11591/ijece.v13i6.pp6568-6577.
[5] A. AlSobeh, A. Shatnawi, B. Al-Ahmad, A. Aljmal, and S. Khamaiseh, “AI-Powered AOP: Enhancing Runtime Monitoring with Large Language Models and Statistical Learning,” Int. J. Adv. Comput. Sci. Appl., 2024, doi: 10.14569/IJACSA.2024.0151113.
[6] M. A. Hossain et al., “AI-enabled approach for enhancing obfuscated malware detection: a hybrid ensemble learning with combined feature selection techniques,” Int. J. Syst. Assur. Eng. Manag., 2024, doi: 10.1007/s13198-024-02294-y.
[7] M. M. Khan, A. Buriro, T. Ahmad, and S. Ullah, “Backdoor Malware Detection in Industrial IoT Using Machine Learning,” Comput. Mater. Contin., 2024, doi: 10.32604/cmc.2024.057648.
[8] U. Ahmed et al., “Hybrid bagging and boosting with SHAP based feature selection for enhanced predictive modeling in intrusion detection systems,” Sci. Rep., 2024, doi: 10.1038/s41598-024-81151-1.
[9] O. Dib, “A decentralized privacy-preserving framework for diabetic retinopathy detection using federated learning and blockchain,” Results Eng., 2025, doi:

10.1016/j.rineng.2025.105456.

[10] S. Zhou, C. Liu, D. Ye, T. Zhu, W. Zhou, and P. S. Yu, “Adversarial Attacks and Defenses in Deep Learning: From a Perspective of Cybersecurity,” ACM Comput. Surv., 2023, doi: 10.1145/3547330.

[11] M. Kumar, “Scalable malware detection system using big data and distributed machine learning approach,” Soft Comput., 2022, doi: 10.1007/s00500-021-06492-9.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)