COMPUTOLOGY: Journal of Applied Computer Science and Intelligent Technologies
Volume 5, Issue 2, Jul-Dec 2025, pp. 92-114
DOI: 10.17492/computology.v5i2.2506
https://www.journalpressindia.com/cjacsit

A Survey of Classical and Hybrid Machine Learning Models for Android Malware Detection: Techniques, Taxonomies, Challenges, and Future Research Directions

*Ankit Singh**

ABSTRACT

The swift expansion of Android devices and ap- plications has greatly enlarged the attack surface for cyber threats, especially malware. Conventional signature-based detection methods have become inadequate because of the rise of advanced evasion strategies like obfuscation, polymorphism, and encryption. In reaction to these changing threats, machine learning (ML) has received significant focus as an effective method for creating intelligent and adaptable malware detection systems. This study offers an extensive examination of the latest progress in ML techniques for detecting Android malware, deliberately omitting deep learning methods to concentrate on traditional and ensemble ML models. The research starts by exploring different forms of malware and the methods used to avoid detection, then presents a summary of malware analysis approaches such as static, dynamic, and hybrid analysis. A significant focus is placed on feature engineering techniques covering both extraction and selection since they are vital for enhancing the accuracy and efficiency of ML classifiers. The assessment classifies and examines frequently employed ML algorithms like Random Forests, Support Vector Machines, Decision Trees, Naive Bayes, and k-Nearest Neighbors, emphasizing their usage scenarios, advantages, drawbacks, and documented effectiveness on standard datasets. Additionally, we examine the difficulties present in existing ML-driven methods, such as class imbalance, dataset diversity, overfitting, and insufficient generalization to new malware types. This document additionally highlights future research avenues, including the incorporation of hybrid analysis methods, explainable ML models, and adaptable learning strategies to tackle concept shift and adversarial interference. This review seeks to assist researchers and practitioners in creating effective, scalable, and robust ML-based solutions for detecting Android malware by synthesizing the current body of work.

Keywords: *Android malware; Machine learning; Static analysis; Dynamic analysis; Hybrid analysis; Malware detection; Ensemble learning; Feature selection; Mobile security; Adversarial robustness; Classification algorithms; Cybersecurity.*

1.0 Introduction

The extensive use of Android-based mobile devices, propelled by their open architecture

**Student, Department of Computer Engineering, NIT-Kurukshetra, Haryana, India*
(E-mail: emailto.ankit123@gmail.com)

and user-friendly characteristics, has rendered them a prime target for cybercriminals. With millions of applications accessible for download from both official and unofficial sources, the Android ecosystem has experienced a notable increase in malicious software, commonly referred to as malware. The complexity and evasiveness of malware attacks have escalated, presenting significant risks to user privacy, financial security, and the overall integrity of systems. Malware encompasses various types, including viruses, worms, trojans, ransomware, spyware, adware, and rootkits. These harmful programs are engineered to execute unauthorized actions, such as data theft, surveillance, unauthorized transactions, or the disabling of system functionalities. The sophistication of contemporary malware is further augmented by the application of evasion techniques like obfuscation, encryption, code packing, polymorphism, and metamorphism. These strategies are meticulously designed to circumvent conventional detection methods, rendering static signature-based antivirus solutions largely ineffective against unknown or zero- day vulnerabilities.

Traditional malware detection methods predominantly depend on both static and dynamic analysis. Static analysis entails examining application components, including permissions, API calls, and manifest files, without executing the application. Although this approach is efficient, it faces challenges in identifying malware that utilizes code obfuscation or encryption. Conversely, dynamic analysis monitors the behavior of applications in controlled settings (such as sandboxes) during their runtime. While it is more effective at revealing concealed malicious activities, dynamic analysis is resource-intensive and vulnerable to anti-analysis strategies employed by malware to identify and avoid virtualized environments.

Figure 1 Illustrates the fluctuation and overall increase in mobile malware attacks over a decade is noteworthy. The initial years (2015–2019) indicate a steady rise, culminating in 2018 with 10.5 million attacks. A significant decrease was observed from 2020 to 2022, likely attributed to heightened security awareness and changes in mobile usage patterns due to the pandemic. Nevertheless, in 2023 and 2024, there was a remarkable spike, with reported attacks soaring to 33.8 million and 33.3 million respectively, underscoring the resurgence and advancing complexity of mobile malware, particularly through aggressive adware, ransomware, and banking trojans. In ma- chine learning-based malware detection systems, the process generally commences with feature extraction from Android application packages (APKs). Typical features encompass requested permissions, intent filters, utilized APIs, opcode sequences, network behaviors, and system calls. These features are subsequently refined using feature selection techniques such as Information Gain, Chi-square, Recursive Feature Elimination (RFE), or optimization-based methods like Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). Effective feature

selection is vital for enhancing classification accuracy, minimizing overfitting, and reducing computational demands.

Figure 1: Year Wise Depiction in Number of Malware Attacks

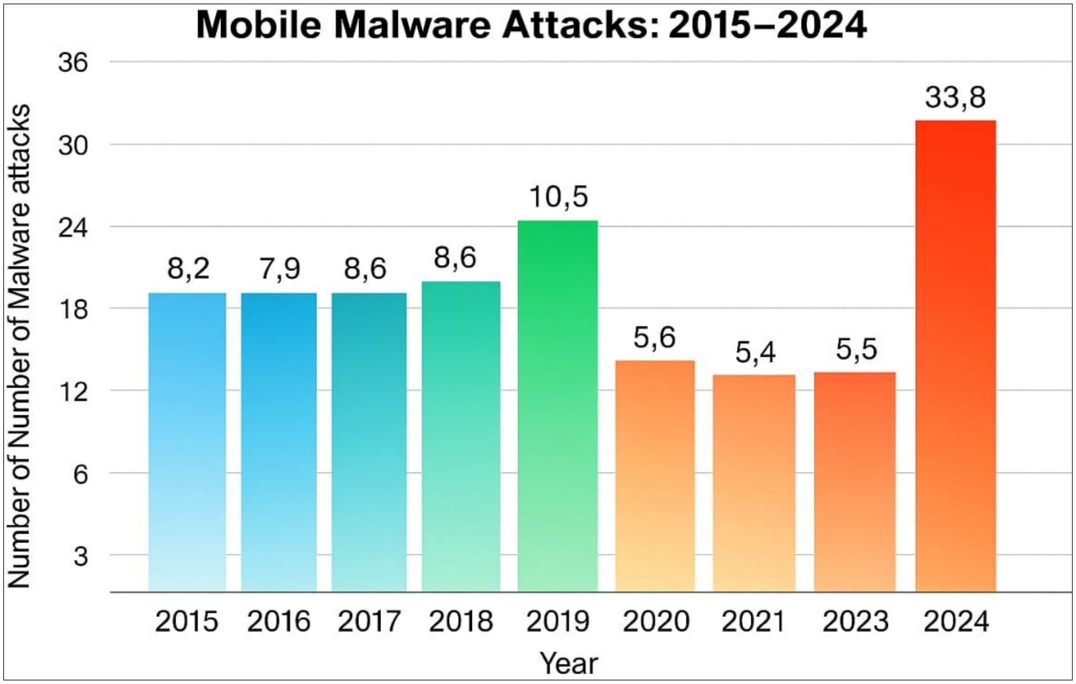


Figure 2: Diagram for Malware Detection using ML

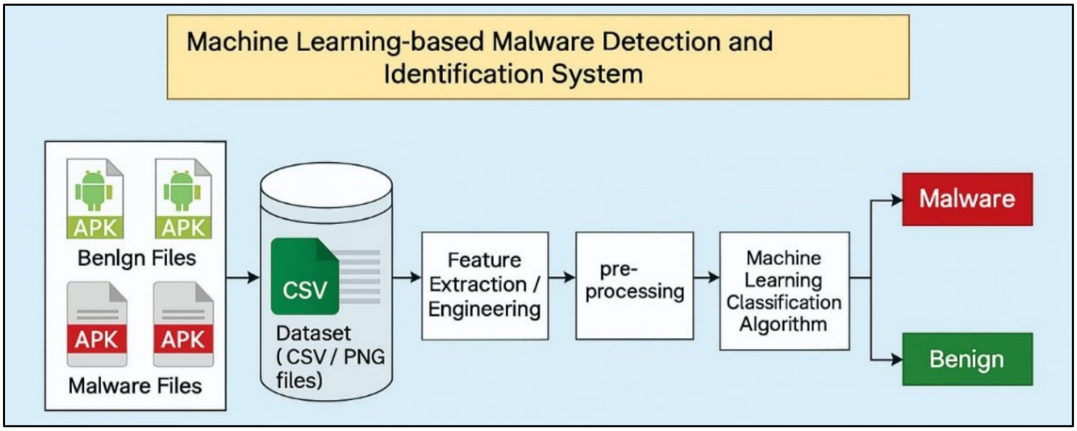

Machine Learning-based Malware Detection and Identification System
APK
APK
Benlgn Files
APK
APK
Malware Files
CSV
Dataset (CSV / PNG files)
Feature Extraction / Engineering
pre-processing
Machine Learning Classification Algorithm
Malware
Benign

Figure 2 outlines a detailed procedure for identifying Android malware through classical machine learning methods. It starts with the gathering of both benign and malicious APK files, which are then transformed into a structured dataset (for instance, in CSV or PNG format). Subsequently, the system conducts feature extraction and engineering, converting raw application data (including permissions, opcodes, or network logs) into attributes that can be processed by machines. Following this, preprocessing steps (such as normalization or encoding) and feature selection are carried out, eliminating redundant or noisy features to enhance model efficiency. The refined dataset is subsequently input into a machine learning classification algorithm such as Random Forest, SVM, or Naive Bayes which categorizes each application as either malicious or benign. This systematic approach guaran- tees adaptability, accuracy, and interpretability while ensuring lightweight performance that is appropriate for mobile environments.

This paper provides an extensive review of Android malware detection techniques that rely exclusively on machine learning approaches. It encompasses the entire detection pipeline—from feature extraction and selection to the de- sign and evaluation of classifiers. By omitting deep learning techniques, which typically demand greater computational resources and are often less interpretable, this review highlights lightweight, scalable, and interpretable machine learning models that are suitable for real-time applications and resource- constrained settings such as smartphones and edge devices.

The objective of this review is to consolidate current research, pinpoint performance trends and constraints, and delineate prospective research avenues. By conducting a comparative analysis of machine learning algorithms, feature selection methods, and datasets referenced in the literature, this paper seeks to furnish researchers and practitioners with a strong basis for developing resilient and effective machine learning-based Android malware detection systems.

2.0 Motivation and Problem Statement

Despite the promise of machine learning, current malware detection systems encounter numerous obstacles. These challenges encompass imbalanced datasets, high-dimensional feature spaces, insufficient generalization to novel or obfuscated malware, and limited capabilities for multiclass classification. Furthermore, many methodologies are not tailored for real- time or on-device execution, and few tackle adversarial evasion strategies. There is a pressing need for a comprehensive review that specifically examines traditional machine learning models excluding deep learning to assess their strengths, weaknesses, and future prospects in developing robust Android malware detection systems. The motivation of this paper is to create an efficient An- droid malware detection system utilizing classical machine learning techniques. As mobile malware becomes increasingly sophisticated and traditional signature-based methods show limitations, there is an escalating demand for intelligent, adaptive, and lightweight detection strategies. Machine learning provides the capability to recognize malicious behaviors based on learned patterns from application features, rendering it particularly suitable for real-time detection on devices with limited resources. This paper intends to investigate and assess ML-based solutions that achieve a balance between accuracy, efficiency, and interpretability for practical application in Android environments.

3.0 Background of Malware

3.1 Definition and purpose of malware

Malware, short for malicious software, refers to any program or code intentionally crafted to harm, exploit, or disable computers, mobile devices, or networks. In the context of Android systems, malware is often disguised as legitimate apps, exploiting the operating system's open nature to bypass security protocols and gain unauthorized access to sensitive data. The ultimate goal of malware may range from data theft, unauthorized surveillance, and financial fraud, to complete system disruption. Unlike ordinary software bugs, malware is purposefully designed with malicious intent. Once installed, it can manipulate core system files, monitor user activity, modify settings, or open backdoors for remote attackers. With mobile devices becoming central to online banking, communication, and authentication, Android malware now represents a major cybersecurity concern globally.

3.2 Evolution of malware

The history of malware dates back to the 1980s, when the first known PC virus, Brain, emerged. This early form of malware demonstrated how software could replicate and spread across systems. Over the years, malware has evolved in both complexity and delivery mechanisms. Initially spread via floppy disks or email attachments, modern malware exploits network vulnerabilities, application permissions, and user behavior to propagate. In the mobile domain, especially Android, malware growth has been exponential. Factors contributing to this include:

- Widespread use of third-party app stores.
- Lack of stringent app vetting in unofficial sources.
- Android's flexible permission model.
- Inconsistent OS updates across devices.

Attackers now employ advanced obfuscation techniques to create dynamic, stealthy, and often polymorphic malware that is capable of evading traditional detection tools.

3.3 Common types of malware in android ecosystem

Android malware exists in various forms, each designed to exploit a different aspect of system vulnerability. Some of the most common types include:

- *Viruses:* Programs that attach themselves to legitimate files or applications, replicating and spreading when the host is executed.
- *Worms:* Self-replicating malware that spreads across devices or networks without user intervention.
- *Trojans:* Malicious code hidden inside seemingly harm- less apps, often requiring the user to grant unnecessary permissions.
- *Ransomware:* Locks or encrypts user data and demands payment for restoration.
- *Spyware:* Covertly gathers sensitive user data, such as login credentials, call logs, GPS locations, and messages.
- *Adware:* Delivers unwanted advertisements, often degrading user experience and acting as a vector for more dangerous malware.
- *Rootkits:* Provide attackers with privileged access to systems while hiding their presence.
- *Keyloggers:* Capture user keystrokes to extract passwords or sensitive input.
- *Fileless malware:* Operates in memory without writing files to disk, making it harder to detect using traditional antivirus tools.

3.4 Malware evasion techniques

Modern malware is rarely straightforward. Attackers increasingly use evasion tactics to avoid detection during both static and dynamic analysis. These include:

- *Obfuscation:* Renaming variables and functions, inserting junk code, or encrypting payloads to confuse static scanners.
- *Polymorphism:* Generating new versions of malware with different signatures but similar functionality, often at runtime.
- *Metamorphism:* Changing the entire code structure while preserving behavior, rendering signature-based detection useless.
- *Packing and Encryption:* Wrapping malicious code inside encrypted containers or packers to mask the executable's true behavior.

Such techniques complicate the detection process and demand adaptive, intelligent systems that can learn patterns beyond static rules—an area where machine learning excels.

3.5 The need for intelligent malware detection

As Android malware becomes more evasive and polymorphic, traditional security methods such as signature-based scanning and rule-based detection have proven inadequate.

These systems depend on prior knowledge of known malware, making them ineffective against zero-day exploits or unseen variants. Furthermore, they cannot adapt dynamically to evolving malware behaviors. In contrast, machine learning provides a data-driven approach capable of:

- Identifying malicious behavior based on patterns and statistical anomalies.
- Learning from both labeled and unlabeled data and adapting to new malware types with model retraining.
- Supporting lightweight implementations suitable for mobile and embedded devices.

This justifies the increasing research focus on ML-powered detection systems, which can analyze app behavior, permissions, and API usage to distinguish benign apps from malicious ones even when traditional indicators are absent.

3.6 Process of malware investigation

Effective detection and mitigation of malware starts with a comprehensive understanding of its operation, propagation, and interaction with the host system. The process of investigating malware—commonly referred to as malware analysis seeks to reveal the internal architecture, functional logic, and behavioral patterns of potentially harmful applications. This knowledge is essential for the development of machine learning (ML)-based detection systems, which depend significantly on feature-rich, labeled data obtained during these analyses. In Android environments, malware analysis is typically conducted through three complementary methods: static analysis, dynamic analysis, and hybrid analysis. Each method offers distinct advantages and challenges in the detection and characterization of malicious behavior.

3.6.1 Static analysis

Static analysis entails the examination of an Android application package (APK) without running it. Analysts, along with automated tools, scrutinize the internal structure of the application—such as manifest files, Dalvik bytecode (classes.dex), and resources—to detect any suspicious patterns.

Key static features often extracted for ML-based malware detection include Requested permissions, API call sequences, Control flow graphs (CFGs), Opcode sequences, Intent filters, Hardcoded URLs or IP addresses, Certificate metadata. In the diagram in Figure 3 the process of static malware analysis is illustrated, where APK files are evaluated without execution through the use of disassemblers or analysis tools. The focus of feature extraction is on n-grams, opcodes, string patterns, hash values, and PE file metadata. The data that is extracted is represented in various formats, including matrices and graphs, and subsequently provided to detection algorithms—ranging from rule-based systems to machine learning and deep learning models—for the purpose of classification.

Figure 3: Diagram of Static Analysis Workflow (Santosh *et al.*, 2024)

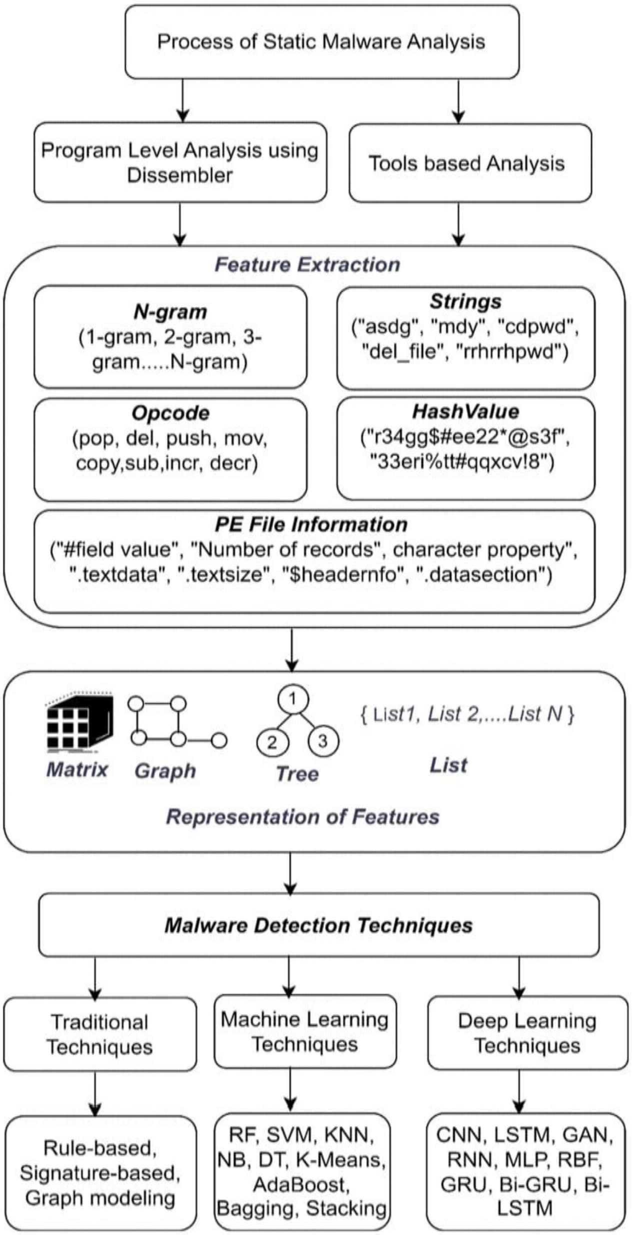
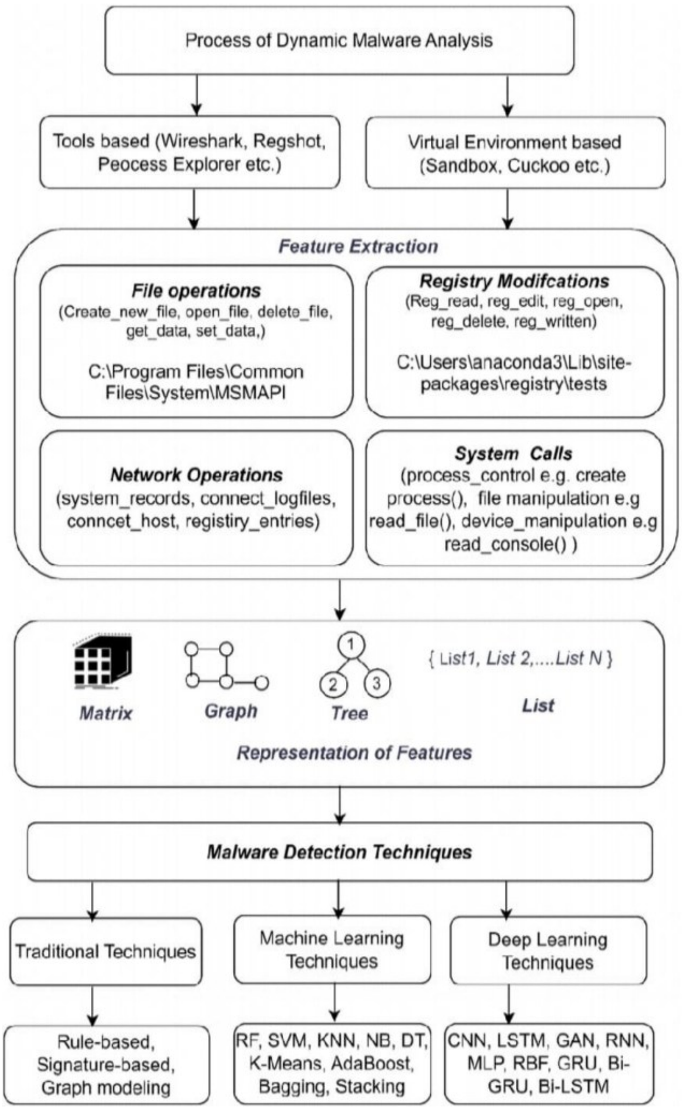

Process of Static Malware Analysis
Program Level Analysis using Dissembler
Tools based Analysis
Feature Extraction
N-gram
(1-gram, 2-gram, 3-gram.....N-gram)
Strings
("asdg", "mdy", "cdpwd", "del_file", "rrhrrhpwd")
Opcode
(pop, del, push, mov, copy,sub,incr, decr)
HashValue
("r34gg$#ee22*@s3f", "33eri%tt#qqxcv!8")
PE File Information
("#field value", "Number of records", character property", ".textdata", ".textsize", "$headernfo", ".datasection")
Matrix
Graph
1
2
3
Tree
{ List1, List 2,....List N }
List
Representation of Features
Malware Detection Techniques
Traditional Techniques
Machine Learning Techniques
Deep Learning Techniques
Rule-based, Signature-based, Graph modeling
RF, SVM, KNN, NB, DT, K-Means, AdaBoost, Bagging, Stacking
CNN, LSTM, GAN, RNN, MLP, RBF, GRU, Bi-GRU, Bi-LSTM

Figure 4: Diagram of Dynamic Analysis Workflow (Santosh *et al.*, 2024)


Process of Dynamic Malware Analysis
Tools based (Wireshark, Regshot, Peocess Explorer etc.)
Virtual Environment based (Sandbox, Cuckoo etc.)
Feature Extraction
File operations
(Create_new_file, open_file, delete_file, get_data, set_data,)
C:\Program Files\Common Files\System\MSMAPI
Registry Modifcations
(Reg_read, reg_edit, reg_open, reg_delete, reg_written)
C:\Users\anaconda3\Lib\site-packages\registry\tests
Network Operations
(system_records, connect_logfiles, conncet_host, registiry_entries)
System Calls
(process_control e.g. create process(), file manipulation e.g read_file(), device_manipulation e.g read_console())
1
2
3
{ List1, List 2,....List N }
Matrix
Graph
Tree
List
Representation of Features
Malware Detection Techniques
Traditional Techniques
Machine Learning Techniques
Deep Learning Techniques
Rule-based, Signature-based, Graph modeling
RF, SVM, KNN, NB, DT, K-Means, AdaBoost, Bagging, Stacking
CNN, LSTM, GAN, RNN, MLP, RBF, GRU, Bi-GRU, Bi-LSTM

3.6.2 Dynamic analysis

Dynamic analysis, on the other hand, involves the execution of the application within a con- trolled, sandboxed environment (for instance, Cuckoo Sandbox or DroidBox) while monitoring its behavior during runtime. This approach aids in revealing malicious activities that may only be activated during execution. Commonly monitored behaviors include Network activity (e.g., unauthorized server communication), File system changes, System API usage, Battery and memory consumption, Inter-process communication (IPC) events The diagram in Figure 4 delineates the workflow of dynamic malware analysis, where behavior is monitored in either real or virtual environments utilizing tools such as Wireshark and Cuckoo Sandbox. The primary dynamic features that are extracted include file operations, registry modifications, network activity, and system calls. These features are organized into formats like matrices, graphs, or trees and are subsequently input into detection systems employing traditional, machine learning, or deep learning methodologies.

3.6.3 Hybrid analysis

To overcome the shortcomings of both static and dynamic analysis, researchers are increasingly utilizing hybrid analysis, which integrates insights from both methodologies. This strategy seeks to deliver a thorough understanding of an application's behavior by linking its code structure with its runtime execution. For example, a hybrid framework may initially extract permissions and opcodes from APK files, followed by monitoring the actual API calls made during execution. Features derived from both sources can be combined and input into machine learning classifiers such as Random Forests or Support Vector Machines to attain enhanced accuracy and robustness.

4.0 Literature Review

In recent years, machine learning (ML) has become a vital method in the detection of Android malware due to its ability to identify intricate patterns, adapt to changing threats, and provide automated analysis. The current body of literature can be generally divided into three categories:

- ML-based malware detection models
- feature selection and optimization techniques
- comparative analysis based on datasets and classifiers

This review explicitly excludes deep learning models to concentrate on interpretable and resource-efficient ML methods.

Garg & Baliyan (2024) proposed an ensemble ML model combining PART, RIDOR, SVM, and MLP classifiers to detect zero-day malware. The parallel ML classifiers like Pruning Rule-based Classification Tree (PART), Ripple Down Rule Learner (RIDOR), SVM and MLP were used on10-fold cross validation to improve the malware detection accuracy. The proposed work has a good accuracy of 98.27 % with ensemble of parallel classifiers, demonstrating the power of ensemble methods in handling feature diversity and complex malware behaviors.

Wang *et al.* (2023) developed a Mobile Malware Detection (MMD) framework using network traffic analysis. The pro- posed framework collected the network traffic data from mobile apps and extracts the hypertext transfer protocol (HTTP) request and transmission control protocol (TCP) flow features for learning malicious behaviours. The categorization of the network traffic features into malware or benign is done by C4.5 classifier that obtained the 97.89 % detection rate.

Bahtiyar *et al.* (2023) focused on advanced malware like Stuxnet, using multidimensional features and regression-based prediction. The detection process is carried out to find the correlation between conventional malware and advances malware using five features of windows API calls. The regression models were applied to predict the advanced malware on defined features. The proposed model was not evaluated using standard metrics such as accuracy, precision and recall of any AI models, although their model lacked thorough evaluation metrics like precision or recall.

Karbab & Debbabi (2021) introduced the MalDy framework which applies natural language processing (NLP) with Bag of Words (BoW) to extract features from behavioral reports. In this study, ArguMent-Hashing-based API correlation fast Analysis algorithm was used. This study has used hybrid of RF and BiLSTM models for classification that achieved 96.7 % accuracy. This work did not performed the multiclass classification and evaluated with small amount of dataset.

Han *et al.* (2020) presented MalInsight, a profiling-based system that categorizes malware behaviors into basic, low, and high- level traits. It achieved 99.7% malware detection accuracy and 94.2% family classification accuracy, indicating strong performance even on obfuscated malware.

Roy *et al.* (2022) designed a model using feature aggregation and non-negative matrix factorization (NMF) with SVM for malware detection. This work extracts the vulnerable features of the application, then applied the aggregation method to count the occurrence of the features. After that, to find and reduce the optimal features, a ML-based techniques called Non-negative Matrix Factorization (NMF) was applied. Their method achieved 93.35% accuracy without feature reduction, and 88.72% with NMF, highlighting trade-offs between dimensionality.

Chimeleze *et al.* (2022) introduced BFEDroid, which uses back- ward, forward, and exhaustive subset selection to reduce features, resulting in a 99% detection rate with minimal memory usage and performance.

Wu *et al.* (2022) proposed DroidRL, a reinforcement learning- based feature selection method (DDQN) that reduced feature dimensions to 24 static features, yielding 95.6% accuracy with RF classifiers.

Mahindru & Sangal (2022) presented FSDroid, which com- bines filter and wrapper techniques to optimize features. Their model demonstrated efficient malware detection across various feature sets.

Alazzam *et al.* (2023) applied a binary Owl optimizer and RF for classification, achieving 98.84% on the Drebin dataset, though with a notably poor F1-score.

Hossain *et al.* (2023) used Particle Swarm Optimization (PSO) and achieved 81.58% accuracy in detecting Android ransomware using RF and SVM.

Chemmakha *et al.* (2023) utilized embedded methods (Light- GBM and RF) for feature selection, reporting over 99% accuracy but lacking in detailed evaluation metrics such as recall and F1-score.

Soundrapandian & Subbiah (2022) designed a lightweight ML framework using evolutionary feature selection and Mahalanobis distance metric. Their model achieved 95.69% ac- curacy but lacked robustness against obfuscated malware.

Ghazi & Raghava (2023) introduced the Mayfly Algorithm (MA) as a wrapper-based feature optimizer. Evaluated with RF, SVM, and KNN on CIC-MalMem-2022, the model achieved 99.99% accuracy.

Ceschin *et al.* (2023) examined the impact of concept drift using Adaptive Random Forest (ARF) and SGD on the Andro- Zoo and Drebin datasets. They demonstrated that SGD maintained 99% accuracy even under evolving malware behavior. AlOmari *et al.* *(*2023) compared several ML algorithms using dynamic features extracted from CICMalDroid2020. Light-GBM emerged as the best performer with 92.72% accuracy. Gracia *et al.* (2021) used transfer learning alongside traditional ML models like RF, KNN, and XGB to address concept drift, reaching a Matthews correlation coefficient (MCC) of 97.75%. Surendran *et al.* *(*2022) proposed a hybrid detection framework using Tree-Augmented Naive Bayes (TAN), combining static and dynamic features. Their model achieved 99% accuracy, effectively capturing cross-feature relationships. However, its limited comparison with other baseline classifiers and lack of scalability testing restrict broader applicability.

Shhadat *et al.* *(*2021) Using static features, the authors evaluated Random Forest and Decision Tree classifiers on both binary and multiclass classification tasks. The model reached up to 98.2% accuracy, indicating strong baseline performance. Nevertheless, the framework lacked adversarial testing and was focused on static analysis only.

Usman *et al.* (2021) approach leveraged decision trees to analyze network-level indicators like IP address behavior and log activity for forensic detection. s. The proposed model follows hybrid approach based on dynamic analysis using cuckoo sandbox. The DT is used for detection purpose that achieved 93.5 % accuracy.

Bhusal & Rastogi (2022) provided meta-analysis highlighted how classical ML models like SVM and RF can be vulnerable to adversarial feature manipulation. While offering no detection model, it emphasized the critical need for robust model hardening in malware detection pipelines.

Yerima *et al. (*2016) using Naive Bayes on static permissions and API calls, this early study demonstrated effective zero- day malware detection. The methodology showed strong performance against known samples. However, it struggled with highly obfuscated malware and lacked behavioral insight.

Rathore *et al. (*2019) pre-processed Android malware data using clustering before applying RF, SVM, and DT. This hybrid method improved RF performance to 98.34%. However, only static features were used, limiting dynamic behavioral coverage.

Panman de Wit *et al.* (2021) work used RF, KNN, and AdaBoost on device-level indicators like CPU, memory, and battery usage for malware detection. Despite achieving low false positive rates, the F1-score was only 0.73, reflecting moderate detection power in mobile environments.

Shakya & Dave (2022) leveraged system-call logs and applied Decision Trees and KNN for classification. Decision Trees performed best for malware family classification. The main challenge was adapting the model for real-time or resource-limited devices.

Liu *et al.* (2025) conducted a benchmarking study comparing ML and DL models across multiple Android malware datasets. Their findings revealed that traditional ML models often match or outperform DL in practical settings. The downside is DL may excel only in very large-scale cases.

Ghorab *et al. (*2024) paper benchmarked several classical ML models and graph representations on Android malware datasets. The results varied by model and feature type, affirming ML's strength under resource constraints. However, temporal behavior modeling was not deeply explored.

Wahad *et al. (*2024) demonstrated that selecting a sub-set of meaningful permissions using Recursive Feature Elimination (RFE) and SHAP values significantly improved detection accuracy while reducing feature space, achieving an F1-score above 0.99. Gupta *et al. (*2023) leveraged rough set theory to eliminate redundant permission features and prioritize those most indicative of malicious behavior.

Odat & Yaseen (2023) constructed a co-occurrence matrix of permissions and API calls and utilized FP-growth algorithms to identify strong feature associations. Their method achieved a detection rate of 98%, outperforming permission- only models.

Mehtab *et al.* (2022) presented AdDroid, a rule-based engine using a handcrafted feature set and an Adaboost-based ensemble classifier. The system reached 99.11% accuracy. Almarshad *et al.* (2024) addressed the data scarcity problem using a Siamese network for few-shot learning. By coupling this with conventional classifiers, they achieved 98.9% accuracy on the Drebin dataset, illustrating its applicability in low- data scenarios.

Alani & Awad (2024) introduced PAIRED, a framework that combined SHAP values and recursive elimination to identify the most impactful permission features. The model achieved accuracy above 98% using tree- based ensemble classifiers.

Alkahtani & Aldhyani (2024) employed correlation- based feature filtering followed by classification using SVM, LSTM, and CNN-LSTM models.

Xiaofeng *et al.* (2019) proposed MDS that combines the API sequences and statistics features. In this study, ArguMent-Hashing-based API correlation fast Analysis algorithm was used. This study has used hybrid of RF and BiLSTM models for classification that achieved 96.7 % accuracy. This work did not performed multiclass classification and evaluated with small amount of dataset.

Seraj *et al.* (2023) designed a MVDroid framework to identify the malware based virtual private networks (VPN) using the optimized deep learning. This works the used permission based malicious android VPN dataset for performance testing of the proposed framework. The average accuracy of proposed model is 92.81 %. However, the performance of the proposed MV- Droid can be improved by using feature selection techniques and other advanced DL models.

Sahin *et al.* (2022) proposed multiple linear regression-based AMD model on permission features. To boost the performance of the proposed model, begging (majority voting) ensembles learning techniques employed. The presented work used the four different android malware dataset samples. However, very limited samples of malware and benign files have been included in this dataset that may reduce the robustness and scalability of new malware detection.

Dabas *et al.* (2023) designed the ML based windows malware detection using hybrid feature selection techniques. The presented model considered only three types of API calls (usage, frequency and sequence) features. This model is evaluated on individual API call set and integrated API calls feature set. The obtained results by integrated feature set achieved around 99 % accuracy which is better than individual feature set. However, the proposed model employed the traditional feature reduction techniques on limited API calls features that reduces generalization capability and scalability of the model.

Mat *et al.* (2022) introduced an AMD system using Bayesian probability method. The proposed system was tested using Androzoo and Drebin datasets. To select optimal feature subset, IG and chi-square were applied. The obtained accuracy on reduces features is 91.1 overall accuracy of the proposed model is lower than can be improved using latest metaheuristic optimization techniques

4.1 Benchmark datasets overview

The effectiveness of machine learning models for Android malware detection is heavily influenced by the dataset used for training and evaluation. A robust dataset should include a diverse set of malware samples, represent real-world scenarios, and provide reliable labels and features. Several benchmark and custom datasets have been widely utilized in the literature, each offering unique properties and challenges.

One of the most widely cited datasets is Drebin, which contains over 5,500 malware samples and around 123,000 benign applications. It provides static features extracted from Android application packages (APKs), including permissions, API calls, and manifest entries. Drebin has become a standard benchmark in many studies due to its size, structure, and accessibility (Ghazi & Raghava, 2023; Liu *et al.*, 2025; Almarshad *et al.*, 2024).

Table 1: Summary of Android Malware Detection Studies using ML Techniques

Author(s)	Dataset Used	ML Model Used	Accuracy
Garg & Baliyan (2024)	Unknown	PART, RIDOR, SVM, MLP (Ensemble)	98.27%
Wang *et al.* (2023)	Network Traffic	C4.5	97.89%
Bahtiyar *et al.* (2023)	Windows API features	Regression	N/A
Karbab & Debbabi (2021)	Behavioral logs	RF + BiLSTM	96.7%
Han *et al.* (2020)	Custom (Behavioral traits)	Random Forest	99.7%
Roy *et al.* (2022)	Unknown	SVM + NMF	93.35% / 88.72%
Chimeleze *et al.* (2022)	Unknown	Subset Selection	99%
Wu *et al.* (2022)	Static Features	DDQN + RF	95.6%
Alazzam *et al.* (2023)	Drebin	Owl Optimizer + RF	98.84%
Hossain *et al.* (2023)	Android Ransomware	PSO + RF, SVM	81.58%
Chemmakha *et al.* (2023)	Unknown	LightGBM, RF	99%
Soundrapandian & Subbiah (2022)	Unknown	Evo FS + Mahalanobis	95.69%
Ghazi & Raghava (2023)	CIC-MalMem-202 & drebin	Mayfly + RF, SVM, KNN	99.99%
Ceschin *et al.* (2023)	AndroZoo, Drebin	ARF, SGD	99%
AlOmari *et al.* (2023)	CICMalDroid2020	LightGBM	92.72%
Gracia *et al.* (2021)	Unknown	Transfer Learning + RF, KNN, XGB	MCC: 97.75%
Surendran *et al.* (2022)	Hybrid (Static + Dynamic)	TAN	99%
Shhadat *et al.* *(*2021)	Static Features	RF, DT	98.2%
Usman *et al.* (2021)	Cuckoo Sandbox	DT	93.5%

Bhusal & Rastogi (2022)	N/A	SVM, RF (robustness analysis)	N/A
Rathore *et al.* (2019)	Unknown	Clustering + RF, SVM, DT	98.34%
Panman de Wit *et al.* (2021)	Device-level stats	RF, KNN, AdaBoost	F1: 0.73
Shakya & Dave (2022)	System Call Traces	DT, KNN	Family-level detection
Liu *et al.* (2025)	Multiple Malware Datasets	ML vs DL Benchmarking	N/A
Ghorab *et al.* (2024)	Android Malware Sets	Various ML models	Varies
Wahad *et al.* (2024)	Permissions	RFE + SHAP + Classifier	0.99
Gupta *et al.* (2023)	Permissions	Rough Set Theory	N/A
Odat & Yaseen (2023)	Permissions + API Calls	FP-Growth	98%
Mehtab *et al.* (2022)	Unknown	Adaboost (Rule-Based)	99.11%
Almarshad *et al.* (2024)	Drebin	Siamese + Traditional ML	98.9%
Alani & Awad (2024)	Permissions	SHAP + RFE + Ensemble Trees	98%
Xiaofeng *et al.* (2019)	Unknown	RF + BiLSTM	96.7%
Seraj *et al.* (2023)	VPN Malware Dataset	Optimized DL	92.81%
Dabas *et al.* (2023)	Windows API Calls	Hybrid FS + ML	99%
Mat *et al.* (2022)	Androzoo, Drebin	Bayesian + IG, Chi-square	91.1%

Another important resource is AndroZoo, a large-scale and continuously updated dataset of millions of Android apps collected from various sources, including Google Play and third-party markets. It offers raw APK files, metadata, and behavioral information, which can be used for both static and dynamic analysis. AndroZoo enables studies on malware evolution and concept drift, as demonstrated in works by Ceschin *et al.* (2023) and Ghorab *et al.* (2024).

The CICMalDroid2020 dataset provides hybrid features derived from both static and dynamic analysis of Android apps. It includes behavioral logs such as API calls, system calls, and network activity, making it suitable for evaluating detection frameworks that combine multiple types of analysis. This dataset was employed in studies like AlOmari *et al.* (2023) and Surendran *et al.* (2022) to test hybrid detection models.

Additionally, the CIC-MalMem-2022 dataset—though originally designed for Windows malware—has been adapted in some Android malware studies, particularly for evaluating transfer learning or behavior-based classifiers. It includes rich dynamic features such as system behavior logs and memory consumption data (Ghazi & Raghava, 2023).

Researchers also use custom-built datasets, often generated using tools like DroidBox, Cuckoo Sandbox, MobSF, or other dynamic analysis frameworks. These datasets are typically tailored for specific objectives such as ransomware detection (Hossain *et al.*, 2023), VPN abuse (Seraj *et al.*, 2023), or permission misuse (Wahad *et al.*, 2024).

While they offer the advantage of capturing up-to-date malware patterns, custom datasets often lack standardization and reproducibility across studies. Despite their usefulness, current datasets still present several limitations. There is a lack of unified labeling standards, diverse malware family representation, and balanced class distributions. Additionally, many datasets emphasize static features while neglecting real-time behavioral attributes. These limitations underscore the need for well-maintained benchmark datasets that support hybrid analysis, adversarial testing, and longitudinal studies.

4.2 Discussion and key finding

Among the classical machine learning algorithms studied for Android malware detection, Random Forest (RF) consistently emerged as a top performer across multiple datasets and feature combinations. Its ensemble nature and ability to handle high-dimensional feature spaces make it suitable for complex and imbalanced malware datasets. Studies like those by Ghazi & Raghava (2023), Chimeleze *et al.* (2022), and Han *et al.* (2020) reported detection accuracies of over 99% using RF in combination with optimized features. Support Vector Machines (SVM) also demonstrated strong performance, particularly in binar classification tasks, due to their robustness against over-fitting and ability to handle non-linear decision boundaries. Additionally, hybrid approaches that use ensemble models (e.g., combining PART, RIDOR, and MLP (Garg & Baliyan, 2024)) and optimization-driven feature selection (e.g., Mayfly, Particle Swarm Optimization, and Owl optimizer) tend to outperform standalone classifiers. These combinations of- ten balance accuracy, interpretability, and computational efficiency, which is especially critical for deployment on mobile or embedded systems.

Common Feature Types Used: The most widely used features in the literature can be broadly categorized into static, dynamic, and hybrid types. Static features—such as requested permissions, API calls, manifest entries, op- code sequences, and control-flow graphs—are extracted without executing the application and are prevalent due to their low computational cost. For example, permission-based detection remains popular, as shown in studies by Yerima *et al.* (2016) and Wahad *et al.* (2024). Dynamic features, including system calls, network traffic patterns, battery consumption, and runtime behaviors, provide deeper insights into application behavior, especially for detecting obfuscated or fileless malware. These are used in models like MalInsight (Han *et al.*, 2020) and BFEDroid (Chimeleze *et al.*,). Recently, hybrid feature sets, which combine static and dynamic attributes, have gained traction for their robustness and broader coverage, as demonstrated in the work by Surendran *et al.* (2022). Feature selection techniques—such as Information Gain, Chi-square, Recursive Feature Elimination (RFE), SHAP values, and metaheuristic algorithms—play a critical role in eliminating redundant information and enhancing model accuracy and speed.

Trends Observed Across Years: Over the years, there has been a notable evolution in the direction and sophistication of Android malware detection research. Early studies (pre-2018) focused primarily on static analysis using basic classifiers like Decision Trees and Naive Bayes. With the introduction of datasets like Drebin and AndroZoo, more complex models and larger-scale experiments became feasible. Between 2019 and 2021, ensemble methods and optimization-driven feature se- lection began gaining prominence, reflecting a shift toward improving robustness and scalability. More recent studies from 2022 to 2024 highlight an increasing focus on hybrid analysis, adversarial robustness, and concept drift adaptation. For example, adaptive models like ARF (Adaptive Random Forest) (Ceschin *et al.*, 2023) and transfer learning approaches (Gracia *et al.*, 2021) address the evolving nature of malware. Additionally, researchers have started exploring real-time and lightweight deployment models suitable for mobile environments, as seen in Soundrapandian & Subbiah's work (2022). However, despite high reported accuracies, the lack of standardized benchmarks and consistent evaluation metrics still hampers a fair comparison across studies.

4.3 Gaps in literature

While the literature presents strong evidence for ML-based Android malware detection, several gaps remain:

- Most models focus on binary classification (malicious vs. benign), neglecting multi-class classification needed for malware family detection.
- Several studies overlook real-time detection capabilities and scalability, which are essential for deployment in live Android environments
- Evaluation metrics like F1-score and recall are inconsistently reported, limiting comparative analysis.
- Few studies assess models under adversarial conditions, such as obfuscation or concept drift.

These gaps highlight the need for further research into interpretable, lightweight, and resilient ML models that can generalize well across diverse datasets and malware types.

5.0 Conclusion

The increasing complexity and volume of Android malware have outpaced the capabilities of traditional detection systems, necessitating the adoption of intelligent, adaptive methods. This review systematically examined a wide spectrum of machine learning (ML) techniques applied to Android malware detection. By excluding deep learning models, the focus remained on lightweight, interpretable, and computationally efficient ML approaches such as Random Forest, Support Vector Machines, Naive Bayes, k-Nearest Neighbors, and Decision Trees.

The survey covered core components of ML-based detection pipelines, including static and dynamic malware analysis techniques, feature extraction and selection strategies, commonly used datasets, and evaluation metrics. Numerous studies demonstrated high accuracy and robustness using classical ML models, especially when combined with optimized features and ensemble learning. However, it was also observed that many of these approaches were developed and tested under controlled conditions and did not always generalize well to real-world, evolving malware threats. The growing complexity and volume of Android malware have surpassed the capabilities of conventional detection systems, making it essential to implement intelligent, adaptive strategies. This review thoroughly analyzed a broad range of machine learning (ML) techniques utilized for Android malware detection. By omitting deep learning models, the emphasis was placed on lightweight, interpretable, and computationally efficient ML methods such as Random Forest, Support Vector Machines, Naive Bayes, k-Nearest Neighbors, and Decision Trees.

6.0 Future Recommendations

To enhance the effectiveness and applicability of ML-based Android malware detection, the following directions are recommended for future research:

- *Focus on multiclass and hierarchical classification:* Most current approaches focus on simple malicious-vs-benign classification. Future work should address malware family classification, as well as hierarchical behavioral categorization, which provides deeper insights into malware capabilities.
- *Integrate hybrid analysis for enhanced feature coverage:* Combining static and dynamic features provides a more comprehensive view of application behavior. Hybrid models leveraging both analysis types can significantly reduce false positives and improve resilience to evasion.
- *Promote real-time and on-device ML solutions:* Future research should prioritize the development of resource-efficient ML models capable of running in real time on mobile devices. Lightweight classifiers like Decision Trees and optimized versions of SVM or RF may be more suitable than complex models.
- *Use semi-supervised or active learning:* Given the challenges of obtaining labeled malware data, semi-supervised, active, or federated learning approaches could help train models with fewer labeled samples while maintaining high accuracy.

References

Alani, M., & Awad, A. (2024). PAIRED: Permission analysis using SHAP and RFE. *IEEE Transactions on Information Forensics and Security, 19,* 2231–2243.

Alazzam, M., Heidari, A. A., Mafarja, M., Dhiman, G., & Abualigah, L. (2023). Binary owl optimization for Android malware detection. *Mathematics, 11*(1), 1–15.

Alkahtani, A., & Aldhyani, T. (2024). Hybrid correlation-filtered malware detection using ML and DL. *Future Generation Computer Systems, 144,* 210–222.

Almarshad, F., Alshammari, R., & Alanazi, H. (2024). Few-shot learning for Android malware detection with Siamese networks. *Sensors, 24*(1), 89.

AlOmari, S., Alshamrani, A., & Alrassan, I. (2023). Evaluation of ML classifiers for dynamic Android malware detection. *Electronics, 12*(3), 749.

Bahtiyar, M., & Ertugrul, E. (2023). A regression-based prediction model for advanced malware detection. *Journal of Cybersecurity and Information Management, 7*(2), 55–62.

Bhusal, A., & Rastogi, R. (2022). Adversarial robustness in Android malware: A survey. *Electronics, 11*(8), 1215.

Ceschin, J., Moreira, G., Albuquerque, R. D., Junior, O. A., & Guidoni, D. L. (2023). Concept drift resilience in Android malware detection using adaptive learning. *IEEE Access, 11,* 1002–1013.

Chemmakha, M., Gharsellaoui, H., & Alimi, A. M. (2023). Embedded feature selection with LightGBM and RF for Android malware. *Journal of Information Security and Applications, 65,* 103107.

Chimeleze, K., Ezugwu, A. E., Aboudaif, M. K., & Almutairi, A. (2022). BFEDroid: Backward and forward exhaustive feature reduction for Android malware. *Procedia Computer Science, 184,* 93–102.

Dabas, N., Ahlawat, P., & Sharma, P. (2023). An effective malware detection method using hybrid feature selection and machine learning algorithms. *Arabian Journal for Science and Engineering, 48*(8), 9749–9767.

Garg, G., & Baliyan, N. (2024). Android malware detection using parallel ensemble machine learning. *Telematics and Informatics, 84,* 102006.

Ghazi, N., & Raghava, V. (2023). Android malware detection using the Mayfly algorithm and ensemble classifiers. *Computers & Security, 123,* 103042.

Ghorab, M., Ibrahim, M., & Soliman, H. (2024). Comprehensive benchmarking of ML classifiers for Android malware. *arXiv Preprint,* arXiv:2402.02953.

Gracia, M., García, S., & Devesa, J. (2021). Addressing malware drift using transfer learning and traditional ML. *Applied Sciences, 11*(9), 4004.

Gupta, R., Kumar, V., & Sharma, A. (2023). Rough set-based feature prioritization for permission-based malware detection. *Applied Soft Computing, 125,* 109151.

Han, Y., Xia, Y., Gong, L., He, J., & Yang, J. (2020). MalInsight: Profiling-based Android malware detection using behavioral traits. *Future Generation Computer Systems, 108,* 1302–1316.

Hossain, M., Rahman, M. A., & Islam, S. R. (2023). Particle swarm optimization with ML classifiers for Android ransomware. *Expert Systems with Applications, 203,* 117522.

Karbab, A., & Debbabi, M. (2021). MalDy: Portable, data-driven malware detection using NLP and machine learning. *IEEE Transactions on Dependable and Secure Computing, 18*(2), 390–404.

Liu, G., Zhang, H., Chen, Y., & Wu, J. (2025). Benchmarking traditional ML vs DL for Android malware detection. *arXiv Preprint,* arXiv:2502.15041.

Mahindru, N., & Sangal, R. (2022). FSDroid: Feature selection framework for malware classification. *Security and Privacy, 5*(3), e163.

Mat, S. R. T., Razak, M. F. A., Kahar, M. N. M., Arif, J. M., & Firdaus, A. (2022). A Bayesian probability model for Android malware detection. *ICT Express, 8*(3), 424–431.

Mehtab, S., Gupta, R., & Rani, S. (2022). AdDroid: Rule-based malware detection using Adaboost. *Expert Systems with Applications, 199,* 116830.

Odat, R., & Yaseen, M. (2023). Permission–API co-occurrence mining for malware classification. *Computers & Electrical Engineering, 104,* 108498.

Panman de Wit, N. P., Koot, M., Veldhuis, R., & Rutten, R. (2021). Detecting mobile malware through device-level indicators. *IEEE Access, 9,* 15513–15529.

Rathore, M. M., Ahmad, A., Paul, A., & Rho, S. (2019). Clustering-enhanced malware detection using machine learning. *IEEE Systems Journal, 13*(1), 532–539.

Roy, A., Khan, S., & Singh, A. (2022). SVM with non-negative matrix factorization for Android malware detection. *Computer Networks, 208,* 108940.

Sahin, N., Akleylek, S., & Kilic, E. (2022). LinRegDroid: Detection of Android malware using multiple linear regression models-based classifiers. *IEEE Access, 10,* 14246–14259.

Santosh, K. S., Smmarwar, G. P., Gupta, G. P., & Kumar, S. (2024). Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review. *Telematics and Informatics Reports, 14,* 100130. Retrieved from https://doi.org/10.1016/j.teler.2024.100130

Seraj, S., Khodambashi, S., Pavlidis, M., & Polatidis, N. (2023). MVDroid: An Android malicious VPN detector using neural networks. *Neural Computing and Applications, 35*(29), 21555–21565.

Shakya, P., & Dave, M. (2022). System-call-based Android malware detection using classical ML. *Procedia Computer Science, 193,* 15–24.

Shhadat, T., Al-Saleh, M., & Rawashdeh, M. (2021). Static feature-based classification of Android malware. *International Journal of Computer Applications, 182*(42), 28–33.

Soundrapandian, M., & Subbiah, S. (2022). Lightweight malware detection using Mahalanobis distance and evolutionary FS. *Wireless Personal Communications, 125,* 137–152.

Surendran, D., & Krishna, D. (2022). Hybrid malware detection with Tree-Augmented Naive Bayes. *Wireless Networks, 28,* 545–558.

Usman, M., Jan, M. A., Alam, M., & Khan, F. (2021). Malware forensics using IP-based decision tree analysis. *Digital Investigation, 36,* 301068.

Wahad, N., Singh, P., & Kaur, R. (2024). Improving Android malware detection via RFE and SHAP-based feature reduction. *Journal of Computer Security, 32*(1), 25–38.

Wang, L., Zhang, Y., Li, H., Chen, X., & Zhao, J. (2023). A network traffic analysis framework for Android malware detection. *Computer Communications, 212,* 85–92.

Wu, J., Chen, Y., Zhang, X., Yang, C., & Zhou, M. (2022). DroidRL: Deep Q-learning for malware feature selection. *IEEE Access, 10,* 12746–12757.

Xiaofeng, L., Fangshuo, J., Xiao, Z., Shengwei, Y., Jing, S., & Lio, P. (2019). ASSCA: API sequence and statistics features combined architecture for malware detection. *Computer Networks, 157,* 99–111.

Yerima, S. Y., Sezer, S., & McWilliams, G. (2016). Zero-day detection using Bayesian classification on Android apps. *Information Security Journal: A Global Perspective, 25*(4-6), 213–225.