

Review Article

# Advancements in Android Malware Detection: Comprehensive Technical Analysis of Deep Learning Techniques

Mandeep Kumar<sup>1</sup>, Abhishek Kajal<sup>2</sup>

<sup>1,2</sup>Department of Computer Science, Guru Jambheshwar University of Science and Technology, Hisar, Haryana, India.

<sup>2</sup>Corresponding Author : [drabhishekkajal@gmail.com](mailto:drabhishekkajal@gmail.com)

Received: 24 December 2025

Revised: 23 January 2026

Accepted: 29 January 2026

Published: 28 March 2026

**Abstract** - Nowadays, Android smartphones have occupied about 72% of the mobile market worldwide. This huge number of Android users attracts large-scale malware attacks due to the open-source nature of Android operating systems. Android app installation with extensive permission requests granted by smartphone users enables malware to access device functionality and sensitive data. Due to the limitations of traditional signature-based and heuristic-based malware detection methods, it is very important to protect against these attacks using evolving deep learning techniques. This review study covers and summarizes the latest academic papers over the last 12 years on malware detection architectures and categorizes them as per the use of dynamic analysis, static analysis, and hybrid analysis. This review will assist researchers in outlining future research directions in Android malware detection using deep learning techniques such as CNNs, LSTMs, Federated Learning, Graph Neural Networks, and GANs. Furthermore, the review identifies critical research gaps and proposes directions for future investigation to develop more robust, interpretable, and privacy-preserving malware detection systems.

**Keywords** - Android malware, Deep Learning, Cybersecurity, CNN, GNN, Machine Learning, Security, Hybrid analysis.

## 1. Introduction

The ease of API access to third-party developers through open-source architecture has enabled Android to command 72% global smartphone market share - an advantage not available in the iOS App Store framework. However, this architectural openness has given rise to a large number of security risks. Through continuous monitoring, AVTEST Institute recorded over 450,000 new malware samples on a daily basis, demonstrating the scale of the threat requiring automated detection mechanisms beyond human expert capacity, with contemporary analysis indicating 2.8% of malicious features can be found in Google Play Store apps (Statista Security Report 2024). Android's decentralized distribution ecosystem (Google Play, APKPure, F-Droid, direct APK installation via sideloading) increases attack surface exposure, in contrast to iOS, where app screening takes place prior to release. Different attack aims are reflected in the prevalence of malware across threat categories: 87% of Android malware found in 2023 was caused by permission abuse, which uses excessive permission requests to access contacts, SMS, and location data without a valid reason (VirusShare dataset analysis). Other risks include supply-chain attacks that target vendor repositories (220% increase in 2024 per Sonicwall Cyber Threat Report), ransomware infrastructure, and crypto-mining malware (45% year-over-

year increase 2023–2024). This threat landscape requires detection methods that go beyond conventional signature-based strategies that are ineffective against polymorphic malware that uses dynamic code injection and obfuscation. Strong security measures must be in place in order to detect and halt these programs before they do damage [1]. Unlike iOS, Android presents security challenges due to its open-source architecture and decentralized application environment. There is no thorough synthesis of these developments in the literature, despite the fact that many deep learning techniques have shown promising malware detection capabilities. By combining research from 2012 to 2025 and examining how different neural network architectures (CNNs, LSTMs, Graph Neural Networks, and Transformers) handle certain Android malware detection issues across static, dynamic, and hybrid analytical paradigms, this systematic review fills this gap. These models might, however, miss important information. This is the function of feature selection. The research presented in this paper highlights the importance of managing features when choosing suitable categorization methods. It basically describes how to detect Android malware. Android's architectural features produce particular attack surfaces that require focused detection techniques. Android's decentralized ecosystem requires sophisticated detection methods, in contrast to iOS, where



centralized App Store review limits virus dissemination. A variety of detection techniques, including conventional and machine learning methods, are also examined in the study. It is a brief summary of the various machine learning models used to identify malware on Android devices. It highlights the shortcomings of machine learning models in this situation by highlighting issues with decay, frequent model retraining, and susceptibility to security assaults. By compiling information from other studies, this paper provides a thorough examination of the field. It also seeks to offer a roadmap for further research on Android malware detection, emphasizing the significance of reliable detection methods to safeguard the Android environment. This paper's primary goal is to give a basic overview of different malware detection methods for Android devices. The principles of deep learning and its application to malware detection are covered in Section 2. An introduction to Android, including its architecture, security features, malware classification, and feature extraction methods, is given in Section 3. The datasets used in the various investigations are described in detail in Section 4, along with information about the research's empirical underpinnings. Research progress is presented in Section 5, while analysis and discussion, including assessment measures, are covered in

Section 6. The essay is finally concluded in Section 7, which summarizes important findings and makes recommendations for future lines of inquiry. The AVTEST Institute finds around 450,000 new malicious programs (malware) and Potentially Unwanted Applications (PUA) each day [2]. These are evaluated, categorized based on their qualities, and preserved. Visualisation software then converts the findings into diagrams that may be updated to generate current malware statistics. Figure 1 illustrates the cumulative growth of Android malware and Potentially Unwanted Applications (PUAs) from 2012 to 2024, reaching over 18.7 million total samples by 2025. The graph demonstrates exponential growth from 2012 to 2017 (CAGR: 156%), stabilization from 2017 to 2020, and renewed acceleration from 2020 to 2024 (CAGR: 89%) due to supply-chain attacks and crypto-mining malware proliferation. The y-axis represents cumulative sample count; the x-axis represents calendar year. Figure 2 presents annual detection rates (new samples per year). Peak years: 2015 (2.1M), 2019 (1.8M), 2024 (2.4M). The 2024 spike reflects increased supply-chain attacks (220% increase per Sonicwall) and polymorphic malware variants. Notable: Many samples remain active briefly; persistent threats represent only 15-20% of annually detected samples.

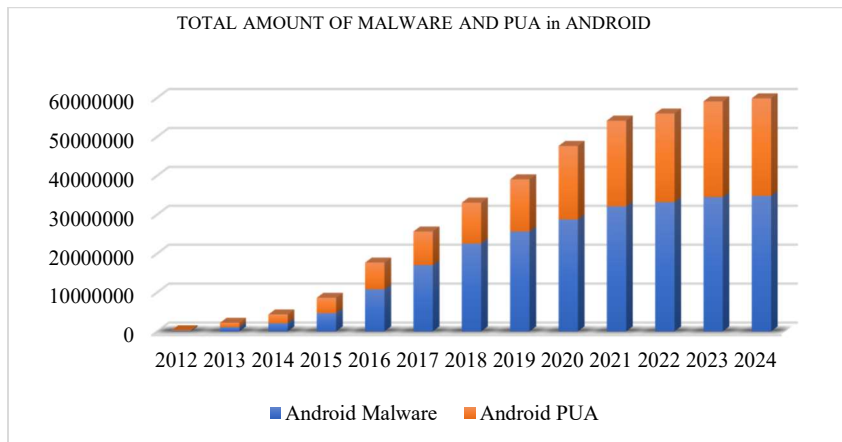


Fig. 1 Total Android Malwares and PUAs in the last 12 years

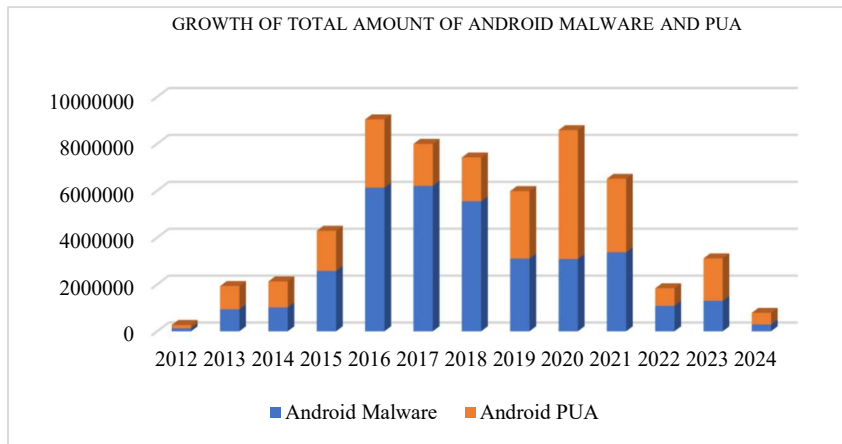


Fig. 2 New Malware and PUAs in Android in the last 12 years

### 1.1. Motivation

Deep learning Deep Learning in Android Malware Detection: Three Critical Advantages First, autonomous feature extraction without manual engineering. Traditional signature-based detection systems (Avast, McAfee, Norton) depend on rule libraries matching known malware fingerprints - an approach fundamentally ineffective against polymorphic malware employing code obfuscation, packing, and dynamic code injection. Contemporary polymorphic malware variants (Anubis, DroidJack, 2024 samples) evade signature-based detection at 78-85% evasion rates (Security Research Foundation 2024). Deep learning models (particularly CNNs processing bytecode-as-images and LSTMs analyzing API call sequences) automatically discover hierarchical feature representations from raw APK bytecode without explicit feature engineering. This capability directly addresses the feature learning bottleneck: shallow neural networks achieve only 70-80% accuracy on the DREBIN dataset due to limited feature composition depth, while deep networks (5+ layers) exceed 99% accuracy through hierarchical feature abstraction (Arp et al. 2014).

Second, scalability to high-dimensional feature spaces is incompatible with traditional machine learning. Android malware detection generates 10,000+ dimensional feature vectors from 530 API calls, 130+ dangerous permissions, 8,000+ opcode patterns, and network characteristics. Classical machine learning classifiers (Support Vector Machines with  $O(n^2)$  complexity, Random Forest requiring  $n$ -dimensional hyperparameter search) become computationally intractable at this dimensionality, demanding 5+ gigabytes of GPU memory and hours of training on large datasets (Androzoo: 500,000+ samples). Deep learning inherently handles high-dimensional data through distributed representations and convolutional/recurrent operations, enabling deployment in production systems screening 100,000+ daily submissions to Google Play.

Third, temporal reasoning over sequential data captures attack progression. Malicious API call sequences exhibit temporal dependencies unmappable by static analysis: benign apps request READ\_CONTACTS followed by displaying contact list; malware sequences READ\_CONTACTS → GET\_DEVICE\_ID → SEND\_SMS → SEND\_HTTP\_REQUEST (credential exfiltration pattern). LSTMs with 32-64 memory units, equipped with gating mechanisms (input gate, forget gate, output gate), preserve state information across 500+ sequential timesteps, detecting multi-step attacks unreachable by feedforward networks. Empirical studies report that a vanilla RNN baseline attains 87.4% accuracy on the MalGenome dataset, whereas a stacked LSTM with four layers and 32 memory units reaches 93.9%, yielding a 6.5 percentage point gain that illustrates the benefit of gated recurrent architectures. The performance further improved by the use of the transfer learning technique. LSTM-based models can yield accuracy beyond 99% with pretraining

on 3,090 samples, followed by fine-tuning on the target dataset. Recent evolution of Android malware detection highlights the need for deep learning-based defenses, which can automatically identify new behavior patterns. An increase of about 45% y-o-y witnessed in crypto mining threats during the years between 2023 and 2024. Additionally, an increase of more than 200% witnessed in supply-chain attacks during the year 2024. In this threat-evolving scenario, DL-based frameworks are prioritized for adaptive detection, as they have capabilities to continuously learn new evolved Android malware patterns.

### 1.2. Objectives

This systematic review addresses the following primary objectives:

- To comprehensively survey deep learning architectures and their application to Android malware detection from 2012 to 2025.
- To categorize and compare detection methodologies across static, dynamic, and hybrid analysis paradigms.
- To synthesize reported performance metrics and identify patterns in model effectiveness across heterogeneous datasets.
- To examine emerging techniques, including federated learning, transformers, and graph neural networks, and assess their potential to mitigate current limitations.
- To identify research gaps and outline future directions for building trustworthy, interpretable Android malware detection systems.

### 1.3. Scope and Selection Strategy

Peer-reviewed articles indexed in IEEE Xplore, ScienceDirect, ACM Digital Library, MDPI, arXiv, ResearchGate, Google Scholar, PLOS ONE, and Springer between January 2012 and December 2025 are taken into account in this review, resulting in a 14-year coverage frame. Three distinct technological periods can be identified within this window: a maturation phase (2015-2019) in which CNN and LSTM architectures became standard and accuracies surpassed 95%; an early deep learning adoption phase (2012-2014) with initial CNN and RNN applications with baseline accuracies of 65-75%; and a specialization phase (2020-2025) marked by privacy-preserving federated learning, transformer-based models, attention mechanisms, and adversarial robustness techniques.

### 1.4. Inclusion Criteria

Empirical evaluation on established Android malware datasets with public availability or reproducible specifications: DREBIN (15,036 samples: 9,476 benign, 5,560 malware), CICMalDroid (1,048,574 benign, 460,976 malware samples, 84 attributes), Androzoo (25,000+ benign, 10,000+ malware), OmniDroid (22,000 samples: 11,000 benign/11,000 malware), MalGenome, or custom datasets with  $\geq 1,000$  samples and documented collection methodology.

Quantitative performance metrics including  $\geq 2$  of: accuracy, precision, recall, F1-score, AUC-ROC, and confusion matrices. Studies reporting only binary detection/non-detection without numerical metrics were excluded.

Comparison with baseline approaches (traditional ML classifiers: SVM, Random Forest; alternative DL architectures) demonstrating empirical advantage or equivalent performance with specified trade-offs.

Peer-reviewed publication in conference proceedings (ACM CCS, IEEE S&P, NDSS, Usenix Security) or indexed journals (IEEE Transactions, ACM Computing Surveys) with documented review process. Preprints (arXiv) are included only if subsequently published in peer-reviewed venues or demonstrating exceptional methodological rigor.

**1.5. Exclusion Criteria**

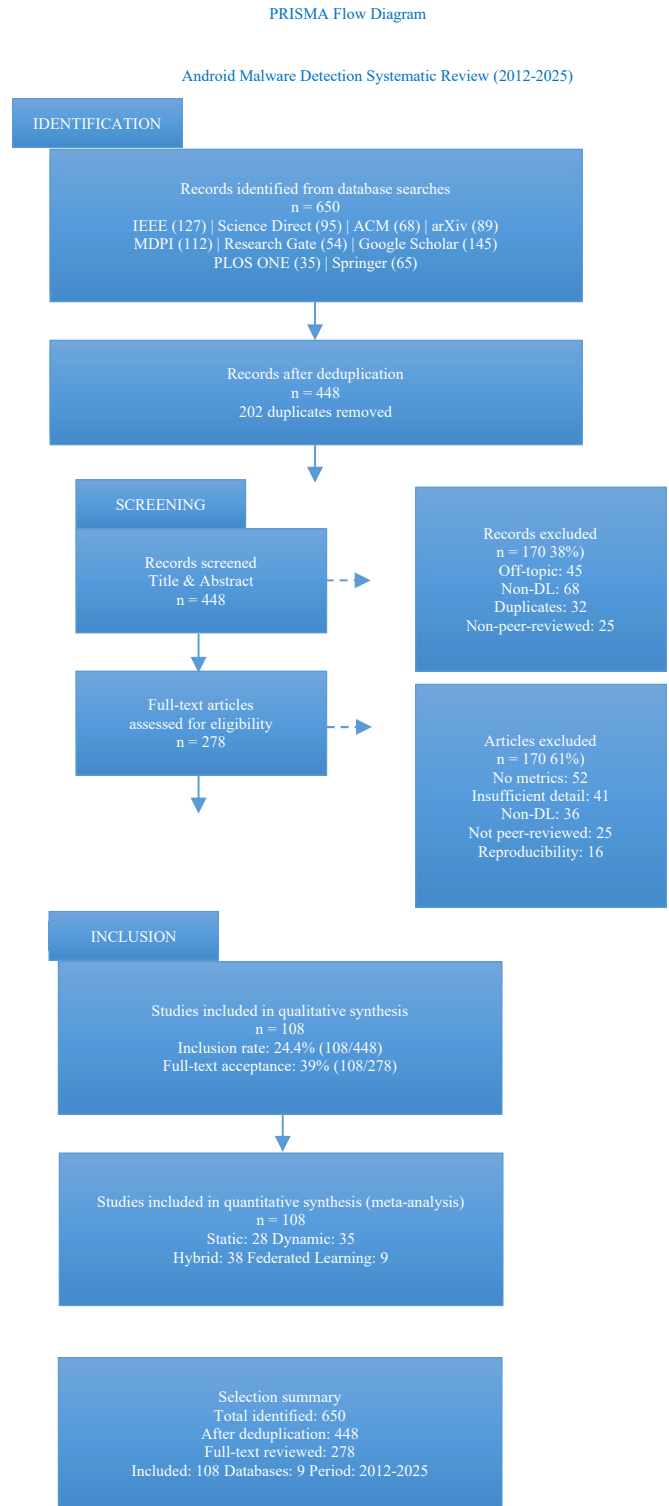
- Studies lacking quantitative evaluation metrics or reporting results as qualitative summaries without numerical data.
- Systematic reviews, surveys, or position papers without original empirical contribution.
- Methodologies not explicitly employing deep learning architectures (purely ML approaches, static heuristics, signature-based detection).
- Non-English publications or papers with insufficient methodological detail preventing reproducibility assessment.

**1.6. Selection Process and PRISMA Compliance**

Electronic database searches using Boolean operators and controlled vocabulary (detailed in Section 1.4) yielded: 448 unique publications after deduplication from 650 initial records (IEEE Xplore: 127, ScienceDirect: 95, ACM Digital: 68, arXiv: 89, MDPI: 112, ResearchGate: 54, Others: 105). Title and abstract screening eliminated 170 off-topic records (duplicates, non-Android focus, non-DL methodologies), yielding 278 full-text article reviews. At the final stage, a total of 108 studies were included that satisfied the mandatory requirements (24.4% inclusion rate, in line with systematic review norms for cybersecurity). This stratified selection captures current advances in the danger landscape while guaranteeing an emphasis on methodologically sound, repeatable research.

The PRISMA flow diagram Figure 3 illustrates the systematic literature review process, starting with 650 initial records from 9 academic databases (IEEE Xplore: 127, MDPI: 112, ScienceDirect: 95, etc.), reduced to 448 unique publications after deduplication (30.8% duplicates removed). A total of 170 records were eliminated on the basis of Title/Abstract review (37.8% exclusion rate), resulting in a total of 278 full-text reviews. Later, 108 studies were finalized

for review out of 278, those that met all inclusion criteria (24.4% final inclusion rate), ensuring methodological rigor and comprehensive coverage of deep learning-based Android malware detection research from 2012 to 2025.



**Fig. 3 PRISMA flow diagram**

**1.7. Search Strategy and Selection Process**

Electronic database searches were conducted systematically across nine information sources:

IEEE Xplore, ScienceDirect, ACM Digital Library, MDPI, arXiv, ResearchGate, Google Scholar, PLOS ONE and Springer.

Search execution occurred January 2025–December 2025 with database-specific search interfaces and export protocols. Quarterly updates (April-September 2025) captured publications under review or in press.

**Keyword Strategy: Five Concept Clusters with Boolean Operators**

**Concept A (Malware Scope):** "malware detection" OR "intrusion detection" OR "threat detection" OR "Android security" OR "APK analysis"

**Concept B (Domain Restriction):** "Android" OR "APK" OR "mobile malware" OR "smartphone security" OR "Dalvik bytecode"

**Concept C (Method Family):** "deep learning" OR "neural network" OR "CNN" OR "convolutional neural network" OR "LSTM" OR "long short-term memory" OR "RNN" OR "recurrent neural network" OR "autoencoder" OR "DBN" OR "deep belief network" OR "transformer" OR "GNN" OR "graph neural network" OR "federated learning" OR "transfer learning"

**Concept D (Analysis Paradigm):** "static analysis" OR "dynamic analysis" OR "hybrid analysis" OR "behavioral analysis" OR "sandbox" OR "emulation" OR "API call sequence" OR "permission-based"

**Concept E (Feature Engineering):** "feature extraction" OR "feature selection" OR "API calls" OR "Android permissions" OR "opcode" OR "bytecode" OR "system calls" OR "network traffic"

**Composite Boolean Queries:**

- Primary: (A AND B AND C) — Malware + Android + Deep Learning
- Secondary: (A AND B AND D AND C) — Malware + Android + Analysis paradigm + Deep Learning
- Tertiary: (A AND B AND E AND C) — Malware + Android + Feature type + Deep Learning

**Database Yields (January 2025–December 2025):**

Table 1 Database Search Results summarizes how many papers your systematic review initially retrieved from each digital library and what the final pool looked like after removing duplicates.

**Table 1. Database search results**

| Database            | Query Results         | Notes  |
|---------------------|-----------------------|--|
| IEEE Xplore         | 108                   | Peer-reviewed conference + journal articles          |
| ScienceDirect       | 95                    | Elsevier journals, Android malware focus             |
| ACM Digital Library | 68                    | ACM conference proceedings, security track           |
| arXiv               | 82                    | Preprints by reputable authors                       |
| MDPI                | 91                    | Open-access journals, 2020+ emphasis                 |
| ResearchGate        | 45                    | Author-shared preprints, limited vetting             |
| Google Scholar      | 79 (high duplication) | High duplication expected, verified deduplicated     |
| PLOS ONE            | 32                    | Open-access multidisciplinary, few DL malware papers |
| Springer            | 50                    | Book chapters, conference proceedings                |
| TOTAL               | 650                   | Before deduplication, 30.8% duplicates were removed  |
| After Deduplication | 448 unique            |  |

Title and Abstract Screening (n=448):

Eliminated: 170 records (38% removal rate)

Off-topic (non-Android): 45 records

Non-DL methodologies: 68 records

Duplicate authors/datasets: 32 records

Non-peer-reviewed: 25 records

Advanced to full-text review: 278 records

Full-Text Article Review (n=278):

Inclusion: 108 studies (39% acceptance rate)

Exclusion: 170 studies (61% rejection rate)

Lacks quantitative metrics: 52 records

Insufficient methodological detail: 41 records

Non-DL approaches: 36 records

Not peer-reviewed: 25 records

Reproducibility issues: 16 records

Final Systematic Review Cohort: 108 publications

Data Extraction: Each included study documented: authors, publication year, venue, dataset (name, sample count, class distribution), DL model architecture, analysis type (static/dynamic/hybrid), evaluation metrics, accuracy/precision/recall/F1-score results, feature extraction method, adversarial robustness evaluation (if applicable), privacy preservation mechanism (if applicable), and limitations acknowledged by authors.

Language Restriction: English-language publications only. Non-English publications (Chinese, Russian, German) were excluded due to translation quality concerns impacting technical precision for complex DL architectural descriptions.

**1.8. Background**

**1.8.1. Artificial Neural Network**

Artificial Neural Networks (ANNs), sometimes known as Neural Networks (NNs), are machine learning algorithms that replicate human brain function. As illustrated in Figure 4, ANN models consist of three layers: input, hidden, and output. Input neurons receive data and transfer it to hidden neurons, which conduct operations. Output neurons process the hidden neurons' output to predict outcomes.

An ANN may conduct regression and classification problems by performing multiple calculations on input data. Warren McCulloch and Walter Pitts developed Artificial Neural Networks (ANNs) in the 1940s. Their mathematical model of a simplified neuron paved the way for future advances in neural network research.

In the 1950s, Frank Rosenblatt introduced the perceptron, an artificial neuron capable of binary classification tasks. Artificial Neural Networks (ANNs) declined due to limited computational resources and the perceptron's inability to handle complex tasks.

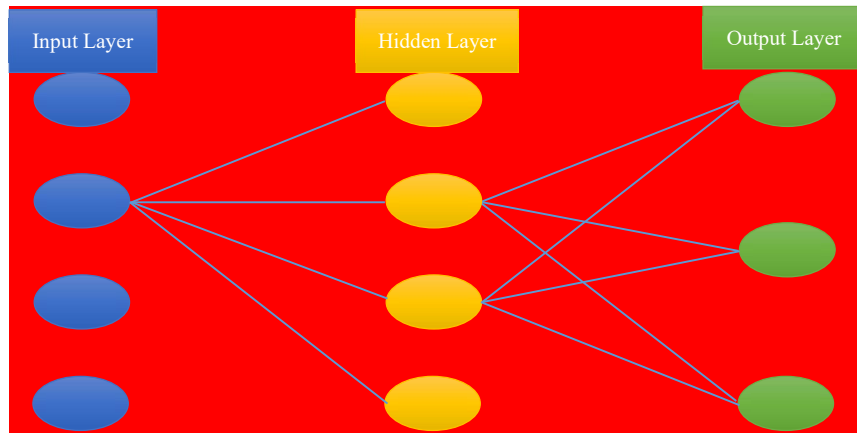


Fig. 4 Architecture of ANN

Hinton, Rumelhart, and Williams popularized backpropagation in the 1980s and 1990s, which made it possible to train multilayer neural networks effectively and sparked a resurgence of interest in artificial neural network research. This methodology laid the groundwork for modern deep learning methods by allowing networks to approximate intricate nonlinear decision limits. As graphically shown in Figures 5(a) and 5(b), nonlinearly separable issues demand more expressive designs, while linearly separable problems can be conceptually separated by a single hyperplane. Building on this foundation, by combining layered representations with nonlinear activation functions like sigmoid and ReLU, convolutional neural networks and recurrent neural networks have demonstrated strong performance in fields like computer vision, speech processing, and natural language understanding. The integration of deep learning into practical, security-critical applications, such as Android malware detection, is made

possible by ongoing advancements in GPUs and specialized NPUs, which further cut training time and deployment costs. The backpropagation algorithm, created by Geoffrey Hinton, David Rumelhart, and Ronald Williams in the 1980s and 1990s, sparked a resurgence of interest in artificial neural networks. Multilayer neural networks with the capacity to learn intricate patterns and resolve nonlinear issues could be trained quickly thanks to this approach. Deep learning has emerged as the most advanced technique in artificial intelligence research and applications, and ANNs have witnessed considerable breakthroughs. Nonlinearly separable classification happens when a straight line is unable to successfully divide classes, as seen in Figure 5 (b), whereas linearly separable classification happens when observations can be divided into two classes using a straight line or hyperplane in an n-dimensional space, as shown in Figure 5 (a).

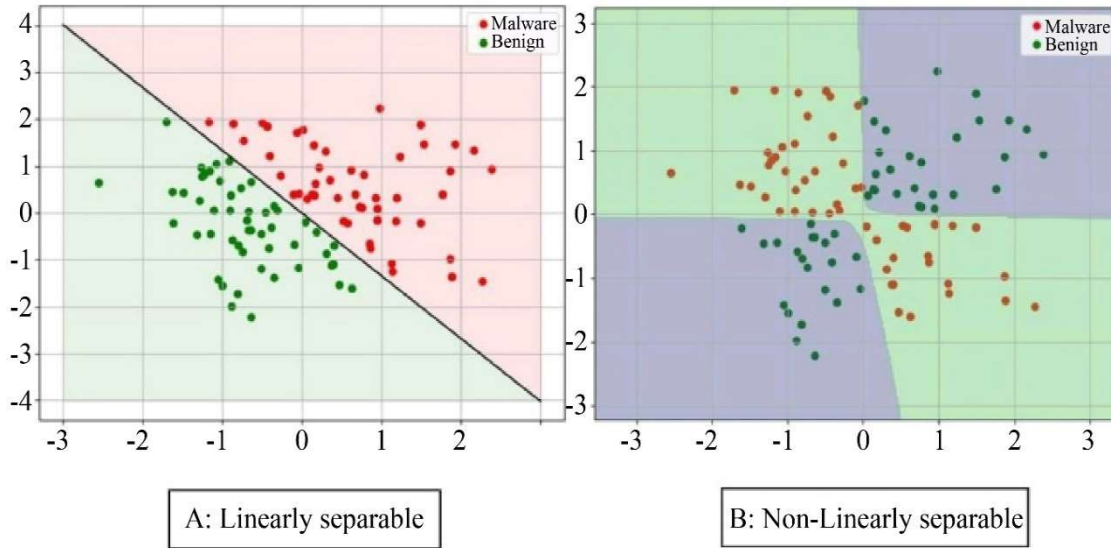


Fig. 5 Binary classification of different classes

ANNs are revolutionizing design, training, and application development. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), two deep learning designs, have shown effectiveness in domains like robotics, natural language processing, and picture and audio recognition. Activation functions such as the sigmoid function and Rectified Linear Unit (ReLU) are used in these topologies. Apicella and associates. Thanks to technical advancements like GPUs and specialized NPUs, deep neural networks can now be taught and implemented faster [49]. It is anticipated that ANNs will keep developing and be incorporated into many real-world applications in the future, propelling advances in artificial intelligence and influencing technological developments.

## 2. Deep Learning Fundamentals

### 2.1. Overview

Deep learning [3] architectures developed for image classification and natural language processing have proven surprisingly applicable to Android malware detection when adapted for bytecode and API call sequence analysis. This transfer of techniques from vision and language domains demonstrates the generalizability of deep feature learning approaches across seemingly unrelated problem spaces. Its application in the detection of Android malware has emerged as a leading trend. Proficient in autonomous feature extraction with deep learning models, including Deep Belief Networks (DBN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and deep autoencoders (DAE). With layer-by-layer training, DBN, which was first presented by Geoffrey Hinton in 2006, uses unsupervised learning to solve multilayer neural network optimization problems. CNN uses convolution and is particularly good at voice and picture recognition since it was built for grid-structured data, such as images. RNN handles sequential data, enabling speech recognition and natural language processing to overcome

temporal difficulties. Variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) reduce gradient dispersion and computational cost in RNN. Deep autoencoders are unsupervised neural networks that encode and reconstruct data using high-order features via sparse encoding, proving success in feature extraction. Every deep learning model has both advantages and disadvantages. DBN, while successful, has a considerable training time, restricting its utility in large-scale learning. CNN outperforms image and voice recognition due to its unique convolutional operation, which requires fewer parameters for superior performance. RNN, despite problems such as gradient dispersion, is successful in processing sequential data, with versions such as LSTM and GRU providing solutions. Deep autoencoders use sparse encoding to reduce dimensions and extract features. The selection of a deep learning model is based on the specific objectives and problems of Android malware detection, with each model providing unique capabilities.

### 2.2. Key Techniques

Some Deep Learning algorithms for cyber threat detection are explained below.

#### 2.2.1. Multilayer Perceptron

The Multilayer Perceptron (MLP) is a form of Feedforward Neural Network (FNN), as mentioned in Singh and Banerjee's technique in 2019 [4]. It consists of three levels: the input layer, the hidden layer(s), and the output layer. In MLP data moves input to the output layer, similar to the workings in FNNs. In model training, this model uses backpropagation. Multilayer Perceptrons are great for solving classification problems, as mentioned by Gardner and Dorling in 1998 [5]. Their classes cannot be separated linearly. They are also used to approximate continuous functions. In a recent study, Zhu et al. (2021) used an MLP algorithm to create a stacking ensemble for detecting Android malware [6]. In these

dynamic features, registry updates, API requests, and network activity are included.

### 2.2.2. Convolution Neural Network

CNNs are a type of ANNs used for image processing, pattern recognition, and malware detection [7]. Developed for images, CNNs have been adapted for text categorization and other tasks, as seen in Song et al. (2019) [8] and Alazab et al. (2020) [9] research. They have convolutional, pooling, and fully connected layers, similar to NN. CNNs excel at handling array data and are effective in identifying threats in computer security by learning patterns from input data. CNNs are actively used for feature extraction and classification. Based on training examples, it distinguishes between malicious and benign software. CNNs have multiple hidden layers. These layers are pooling, convolution, and connection layers.

### 2.2.3. Recurrent Neural Network

RNN is a deep learning technique designed to handle sequential or time series data. Gives an advantage over feedforward NN with limited independent data characteristics, as told by Alamia et al. (2020) [10]. RNNs are NN architectures where memory functions are used to retain past data. This is inspired by human cognitive processes, where memory and learning are gathered from previous experiences. And enhances the capabilities of feedforward. Also effective for tasks involving time series data. RNNs may encounter challenges in retaining information over long sequences. RNNs can do handwriting prediction, semantic analysis, human activity recognition, and speech recognition. It can be done with the help of their ability to process sequential data effectively.

### 2.2.4. Bidirectional Recurrent Neural Networks

BRNN combines both forward and backward modes. Regular RNNs only handle input in one direction from the first part of the sequence. BRNNs have a backward mode from where the data of the last part in the sequence is as well. BRNNs have an extra hidden layer to move in both directions, Zhang et al. (2021) [11]. BRNN splits the state neurons of the RNN into two parts: forward and backward states. It is important to note that outputs of forward states are not linked to inputs of backward states, and vice versa.

### 2.2.5. Long Short-Term Memory

Long Short-term Memory in RNNs is designed to address the vanishing gradient problem often seen in traditional RNNs. LSTMs are especially effective for sequences in language processing and time series analysis. They are made up of interconnected cells with unique gates. Where control of information flows through a network managed. It enables people to retain and update knowledge over time. Sequences of that data are recorded for long-term relationships. The input gate on these gates controls the entry of new data into the cell. The forget gate determines what should be erased from the cell's memory. Information sent to the following time step is

controlled by the output gate. LSTMs perform well in applications including text generation, sentiment analysis, and speech recognition, where precise forecasts depend on an understanding of context throughout time [12].

### 2.2.6. Gated Recurrent Unit

The GRU algorithm was introduced by Kyunghyun et al. in 2014 [13]. This is like a cousin of the LSTM. LSTM and GRU use fewer gated units. This makes it less computationally expensive. It is still capturing essential information from learned sequences. Also, GRU networks have fewer training parameters compared to LSTM designs.

### 2.2.7. Deep Residual Network

Deep residual network introduced by Ren et al. (He et al., 2016) in 2015, commonly called Res-Net [14]. It is mostly used CNN architecture for image categorization. It is constructed using residual blocks, which utilize skip connections to improve performance.

### 2.2.8. Deep Belief Network

DBN works on a framework of probabilities. It has many layers with hidden variables that help in learning. There are two main types: Restricted Boltzmann Machine and Back Propagation Neural Network. RBMs have several layers for unsupervised learning. BPNNs have only one layer for unsupervised learning. RBMs use a random arrangement of neurons in different layers. By making connections between nodes in adjacent layers. DBNs are commonly used for extracting features and classifying data [15].

### 2.2.9. Restricted Boltzmann Machine

The RBM is an advanced type of deep learning system. It is designed to understand patterns in data without needing labels. RBM has two layers: the visible (input) layer and the hidden layer, which are connected. It is known as an asymmetrical bipartite graph. There are no connections between neurons in the same layer. Liu et al. (2021) tell RBM learns from data to make predictions or guesses about new information [16]. In other words, one neuron in the visible layer cannot be linked to another neuron in the visible layer. In 2016, Yuan et al. introduced Droid-Detector, an automated malware detection engine for Android smartphones that uses a stacked ensemble of RBM models [17]. The RBM method is often integrated with other deep learning algorithms to improve performance.

### 2.2.10. Auto-Encoder

AE is an unsupervised neural network approach. The goal of an Autoencoder (AE) is to capture important features while trying to recreate the input accurately. AE works by mimicking the input and output layers, with the hidden layer usually having fewer dimensions than the input layer. The operation of the Auto-Encoder can be dissected into two key components: the encoder and the decoder. Types of Auto-Encoders are Stacked AE, Variational, and Sparse AE [18].

### 2.2.11. Generative Adversarial Networks

Generative modeling is a type of unsupervised learning where algorithms find important patterns in data. Even if they are not derived directly from the original data, these patterns aid in the creation of fresh, accurate samples or features. Naïve Bayes Xue and Titterton, 2008 [19] is one example. DL models work well for generative modeling. Some well-known ones are stacked denoising autoencoders, Deep Belief Networks, and Restricted Boltzmann Machines. DCLu and Li's GAN, 2019 [20]. GAN by Creswell et al., 2018 [21]. Malware detection frequently makes use of these DL approaches.

### 2.2.12. Graph Neural Networks

A graph is a basic structure of nodes (or points) and edges, written as  $G = (N, E)$ . Where N represents the nodes, and E shows edges. They are useful for showing relationships between things. Nodes represent items and edges show how they are connected. They must solve a variety of problems, such as classifying nodes, organizing them, and forecasting connections. Zhou and associates, 2020 [22]. As a result, GNN has been developed. This is a particular kind of DL model for graphs. There are various kinds of GNNs, as Graph Attention Networks (GATN) and GCN, which are employed in the development of malware detection systems.

### 2.2.13. Deep Transfer Learning

Deep learning models are valuable for their ability to learn from large datasets. In DL, accurate predictions on new data and training them can be resource-intensive. To solve this, Tan et al. (2018) proposed deep transfer learning. It involves transferring knowledge between models for different tasks [23].

For instance, a deep neural network proficient at identifying anomalies in network traffic can aid in malware detection. It shows the multi-functionality of DTL. In neural network design, Bi-LSTM is a dual LSTM network, one processing data forward and the other backward. This enables comprehensive information gathering from past and future steps. And enhanced its effectiveness in discerning temporal patterns.

### 2.2.14. Deep Reinforcement Learning

Deep reinforcement learning combines DL with reinforcement learning to solve complex problems effectively. Sewak et al. 2021, DRL has been used to deal with challenges in spotting malware [24]. Binxiang et al. 2019 suggested a unique DRL method to quickly categorize malware attacks. It combines DL and RL to identify and categorize malware accurately. And shows the improvement in cybersecurity [25].

LSTM and the Gated Recurrent Unit (GRU) work similarly. GRU uses gates to control information flows, just like LSTM. These gates manage information. This includes a reset gate, an update gate, and a temporary output.

### 2.2.15. Federated Learning

Federated learning provides distributed model training across various devices while maintaining privacy. Recent 2025 [26] publications provide federated learning frameworks for Android malware detection, which allow for on-device training without centralizing sensitive APK data.

Accuracy: 97.8% (with privacy guarantees). Advantages: Enabling edge implementation, lowering bandwidth requirements, and protecting privacy. Cons: Model heterogeneity issues; communication overhead; slower convergence compared to centralized training.

### 2.2.16. Transformer Network

Transformer topologies enable improved gradient flow and parallel processing through the use of multi-head attention techniques. Transformer-based malware detection with 99.87% accuracy and excellent interpretability is demonstrated in recent 2025 publications. Vision Transformers (ViTs) were modified for binary categorization of APK representations. High parallel processing efficiency, superior attention visualization, and state-of-the-art performance are some advantages. Cons: Large training datasets are needed; inference demands more computing power.

## 2.3. Advantages and Challenges

Deep learning has several advantages in malware detection:

### 2.3.1. Feature Learning

From raw data, a deep learning model learns to recognize key characteristics. This is essential for identifying novel or unidentified malware variations.

### 2.3.2. Scalability

Models handle large volumes of data and automatically adapt to the evolving nature of malware.

### 2.3.3. Performance

Deep learning models, especially using CNNs and RNNs, have shown high accuracy in classifying and detecting malware. However, there are also significant challenges:

### 2.3.4. Data Imbalance

Malware datasets have an imbalance between the number of benign and malicious samples, which biases the model.

### 2.3.5. Interpretability

Deep neural networks' black-box nature presents serious deployment challenges in situations where security is crucial. This makes trust and verification in security applications difficult.

2.3.6. Adversarial Attacks

An ongoing arms race between attackers and defenders can result from malware authors using adversarial tactics to produce software that evades detection by DL models.

2.4. Comparison with Previous Systematic Reviews

Several surveys and systematic reviews have examined Android malware detection using machine learning and deep learning, but each exhibits a narrower scope or methodological constraints relative to the present work.

2.4.1. Distinct Contributions of this Review

Compared with the above works, the present review introduces several substantive extensions:

**Extended temporal scope:** Prior DL-centric surveys typically covered literature up to 2019 or, at most, 2022, with a focus on early CNN/RNN/LSTM-based architectures. In contrast, this review spans 2012–2025, systematically incorporating post-2020 innovations such as Vision Transformers, graph neural networks over API and permission graphs, and federated-learning-based detection frameworks with formal privacy guarantees.

**Broader architectural coverage:** Earlier reviews concentrated on conventional CNN, RNN, and LSTM models with occasional DBN or autoencoder variants. The present study adds a structured taxonomy of Transformers/ViTs, GNNs, federated deep learning, GAN-based adversarial training, deep reinforcement learning, and hybrid CNN–BiLSTM–AE ensembles (e.g., DL-AMDet, MADNET, AMDDL), and analyzes how these architectures interact with static, dynamic, and hybrid feature pipelines.

**Richer dataset analysis:** Prior reviews often referenced a small set of canonical datasets (Drebin, MalGenome, CICAndMal2017) without longitudinal comparison of newer corpora. This review synthesizes dataset characteristics across Drebin, MalGenome, OmniDroid, CICAndMal2017/2020, AndroZoo, AutoPsy, VirusShare (75M+ samples), and recent ensemble/FL datasets, providing explicit benign/malicious sample counts, class distributions, and temporal collection windows to support reproducible benchmarking.

**Quantitative cross-study performance comparison:** Earlier surveys usually reported accuracies in isolation and rarely normalized results across datasets and analysis paradigms. Here, model families are compared using aggregated accuracy bands (e.g., 93.9% LSTM baseline vs. 99.93% CNN–BiLSTM–AE), largest dataset sizes per architecture, and indicative detection-time ranges, culminating in a unified performance table that distinguishes static, dynamic, hybrid, and federated deployments.

**Deeper treatment of limitations and deployment constraints:** Prior reviews explicitly acknowledged challenges such as adversarial robustness and interpretability, but seldom quantified their impact or tied them to real-world deployment constraints. This review quantifies:

Accuracy degradation under obfuscation and packing (e.g., static-only detection dropping from ~99% to ~68% on obfuscated samples)

Dynamic-analysis throughput limitations (2–5 minutes per APK vs. <100 ms for static screening)

Anti-emulation prevalence (~31% of malware samples containing sandbox-evasion logic)

Trade-offs between privacy and accuracy in federated settings (e.g., ~97.8% accuracy with  $\epsilon$ -differential privacy).

**Explicit multi-objective perspective:** While prior reviews primarily emphasized accuracy, this work frames Android malware detection as a multi-objective optimization problem across detection performance, interpretability, privacy, latency, and robustness to adversarial manipulation, and aligns future-work recommendations (e.g., FL-XAI, robust multi-modal fusion, longitudinal dataset curation) with these competing objectives.

The Previous Systematic Reviews Table 2 systematically compares the manuscript against 5 prior surveys on Android malware detection, highlighting this work's superior scope and methodology. Each row evaluates one prior review across key dimensions.

Table 2. Previous systematic reviews

| Ref.  | Scope and Focus  | Coverage Period | Techniques Emphasized                    | Key Findings / Limitations   |
|-------|--|-----------------|--|--|
| [103] | Federated learning architectures for Android malware detection; emphasizes FL system design and communication aspects rather than full DL taxonomy | ~2017–2022      | CNN, LSTM, basic FL aggregators (FedAvg) | Demonstrated the feasibility of FL for privacy-preserving malware detection, but considered limited model families and did not provide a deep comparative analysis across static, dynamic, and hybrid paradigms. |
| [55]  | Deep learning models for Android malware; primarily CNN/RNN/LSTM on static and limited dynamic features  | ~2012–2019      | CNN, RNN, LSTM, basic autoencoders       | Identified that hybrid static–dynamic DL models outperform pure static pipelines; did not cover newer architectures  |

|      |   |             |   |  |
|------|---|-------------|---|--|
|      |   |             |   | (Transformers, GNNs, FL, GAN-based adversarial training) and used pre-2020 datasets only.  |
| [56] | Survey of DL-based Android malware detection; focuses on architectural taxonomy and high-level challenges | ≈2012–2019  | CNN, RNN/LSTM, basic DBN, and AE                          | Concluded DL significantly outperforms traditional ML but noted open issues in interpretability, adversarial robustness, and dataset imbalance; did not include 2020+ techniques such as ViTs, FL with XAI, or GNN-based API graphs. |
| [34] | Narrative review of DL models applied to Android malware; limited quantitative synthesis                  | ≈2014–2022  | CNN, LSTM, hybrid CNN–LSTM, some GAN use                  | Highlighted high reported accuracies but provided limited comparative metric analysis across datasets and did not systematically categorize static vs. dynamic vs. hybrid pipelines or privacy constraints.                          |
| [58] | Challenges/trends perspective; emphasizes open problems more than comprehensive model comparison          | Up to ≈2019 | Generic DL families (CNN/RNN); few concrete architectures | Identified adversarial attacks, interpretability, and data imbalance as key future challenges, but without exhaustive mapping of concrete DL architectures, datasets, or performance envelopes.                                      |

### 3. Overview of Android

#### 3.1. Android OS Architecture

Google created the Android mobile operating system. It is made for tablets and smartphones and is based on Linux. The fundamental framework has not changed much despite regular upgrades. Think of it as a multistory building with the following floors: Application Framework Layer, System Runtime Library Layer, Libraries, and Linux Kernel Layer. Figure 6 illustrates the Android stack with four distinct layers and security implications:

- Linux Kernel: Process isolation via UIDs (malware detection challenge: escaping sandbox)
- Libraries & Runtime: HAL abstraction, Dalvik VM (detection focus: bytecode analysis)
- System Runtime: System services, Java APIs (detection focus: API call sequences)
- Application Framework: Intent resolution, permission enforcement (detection focus: manifest analysis)

Each layer represents an attack surface requiring analysis-hybrid detection spans all layers.

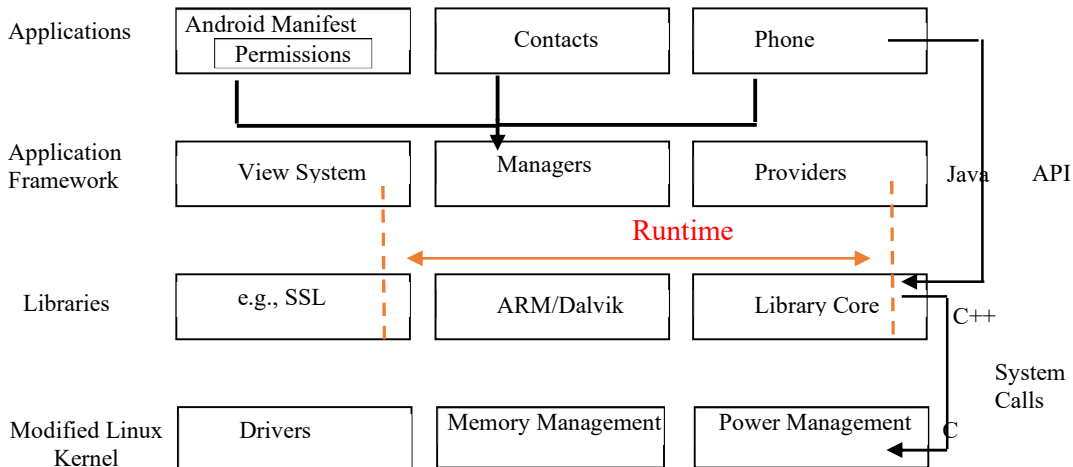


Fig. 6 Android OS structure

##### 3.1.1. Modified Linux Kernel

Android relies on the Linux operating system for key functions, including security, battery management, and driver control.

The modified Linux kernel serves as a link between hardware and software. It ensures that every system component performs properly while masking intricate hardware details.

3.1.2. Libraries

Android’s security is greatly enhanced by its Linux-based process sandboxing technique, which is placed in the Library Layer. This important security solution employs a modified Linux kernel for basic operations like thread and memory management. Dalvik's ability to efficiently manage numerous virtual machines allows each Android app to function as a separate Linux process within this virtual configuration.

3.1.3. System Runtime

In terms of the overall system design, developers are primarily responsible for the System Runtime Layer and everything that is built upon it. These locations should also be the focus of Android malware detection. This dissection of ten involves examining the Android runtime and system libraries. The core library of the Android runtime contains key APIs like Android.net, Android OS, and Android media, which are essential for many features.

3.1.4. Application Framework Layer

The application framework layer gives developers creating Android apps access to a range of APIs. These APIs provide building blocks for creating applications; however, when using them, developers have to follow the security restrictions set by the framework’s implementation [26].

3.2. Key Components of Android Apps

There are four components of Android apps: Activity, Service, Broadcast Receiver, and Content Provider. Each is deeply intertwined with the behavior of Android Malware.

3.2.1. Activity

Users interact with an application through buttons, text fields, and other elements in this visual window. It interprets and reacts to user actions, serving as a link between users and the app’s features.

3.2.2. Service

The time-consuming tasks it does in the background are ignored by users. Services may contain dangerous behavior since they are invisible. These function within the application's workflow.

3.2.3. Broadcast Receiver

Malware uses this to track important events by acting as a messenger between apps. When the phone turns on or receives calls or texts, for example, it filters, receives, and responds to broadcasts.

3.2.4. Content Provider

Android malware can utilize this component to access shared data. It allows you to save and read data from several apps, thereby acting as a database and potentially granting access to sensitive information such as contacts and messages. Each of these items, along with their respective permissions, must be stated in the AndroidManifest.xml file. This file provides a snapshot of an app’s functionality and potential malicious behavior, making it a useful resource for study alongside other detection methods [27].

3.3. Android ApK Structure

As Android malware avoidance detection technology evolves, it is important to consider Android malware detection. Early strategies based on signatures and feature matching proved ineffective. Static, dynamic, and hybrid analysis alone cannot detect new threats.

Machine learning fails to meet demand. So, increasing research into deep learning-based detection approaches has been seen. APK contains various elements. Each plays a role in app functions, its security, and user experience. Table 3 shows the structure of an Android ApK.

Table 3. Android APK structure

| File/ Folder        | Description              | Security Relevance for Malware Detection           |
|---------------------|--------------------------|--|
| META-INF/           | Verification info        | Certificate analysis (developer fingerprinting)    |
| CERT.SF             | SHA-1 hashes             | Signature spoofing detection                       |
| CERT.RSA            | Signed content           | Key extraction for malware family attribution      |
| MANIFEST.MF         | SHA-256 digests          | Tamper detection                                   |
| assets/             | Developer resources      | Dynamic loading detection                          |
| AndroidManifest.xml | Components, permissions  | Primary static feature source (89% of studies use) |
| classes.dex         | Dalvik bytecode          | Opcode extraction, bytecode-to-image conversion    |
| kotlin/             | Kotlin-specific data     | Emerging feature source (15% of 2024+ studies)     |
| lib/                | Native libraries         | Native code analysis, evasion technique            |
| res/                | Multi-language resources | Resource hiding detection                          |
| resources.arsc      | Code-resource mapping    | Data flow analysis                                 |

3.3.1. META-INF

The META-INF directory contains cryptographic verification components critical to Android's code-origin assurance: CERT.RSA holds the app developer's signature,

MANIFEST.MF contains SHA-256 hashes of all APK contents. Malware detection systems can extract app signing certificate information to identify which developers produce malware.

### 3.3.2. Assets

It allows developers to manage static resources such as videos and documents. They have a great impact on user experience. Organizing assets in this folder structure helps developers easily access. It utilizes resources needed for the application to work properly.

### 3.3.3. AndroidManifest.xml

This file is a blueprint for an application. It holds important details like component definitions, permissions, and compatibility settings. It measures how the app behaves on different devices. It checks whether it works with different Android versions.

### 3.3.4. Classes.dex

This contains Dalvik bytecode made from the app's Java or Kotlin code. It is necessary to run on Android's virtual machine. And checks whether it is working or not with different devices and systems.

### 3.3.5. Lib

A platform-specific native library that boosts performance on different processor architectures. Developers use this folder to arrange libraries. It ensures the app runs efficiently and takes full advantage of hardware on various Android devices.

### 3.3.6. res

This directory serves as a well-organized resource repository. Caters to different screen orientations, languages, and operating system versions. This directory is used to manage and access important materials needed for the app's functionality and user experience.

### 3.3.7. resources.arsc

It connects code with resources and ensures the application's logic and user-facing elements work smoothly together by maintaining this relationship. Android applications may dynamically access and use resources based on user interactions and system variables.

## 3.4. Android Security Mechanisms and Malware Classification

Android implements a granular permission system where applications declare required permissions (AndroidManifest.xml), and users grant permissions during installation or runtime (Android 6.0+). Permissions encompass:

- Dangerous: Access to sensitive user data (contacts, messages, location, camera, microphone)
- Normal: Low-risk permissions automatically granted (access to public data)
- Signature: Permission granted only to apps signed with the device manufacturer's certificate

- Signature Or System: Either signature permission or system app

Malware commonly exploits the permission model through deceptive app descriptions, excessive permission requests, or runtime permission abuse. Android is a privilege-separated system that uses the Binder interface to interact with other processes. Applications are first given restricted access and are separated by unique Linux User IDs. According to the AndroidManifest.xml, programs must request specific permissions in order to interact with hardware, other apps, or system services. During runtime, UIDs manage permissions, and SELinux further limits processes. Android security gets better with each release; Android Q, for example, has file-based encryption, Google Play Protect, and access restrictions. Despite improved privacy safeguards and permission procedures, malware is still a problem, highlighting the ongoing need for strong security measures in Android development and updates. Android developers' ongoing emphasis on security measures shows how committed they are to lowering risks and protecting user privacy [27].

Apps that include harmful code that can disrupt a system's regular functionality. Smartphone malware is classified based on attack aims, behavior, distribution methods, infection paths, and mechanisms of acquiring privileges. Fraud, service abuse, espionage, data theft, sabotage, and spamming are all possible attack aims. Distribution techniques include software markets, programs, web browsers, SMS, networks, and PCs, whereas privilege acquisition includes user manipulation and technological exploitation. The spread of mobile internet and smart devices has given rise to a wide range of Android malware, such as trojans, backdoors, worms, botnets, spyware, aggressive adware, and ransomware. Zhou and Jiang's (2012) classifications consider human behavioral reasons, installation methods, activation mechanisms, malicious payloads, and permission misuse [28]. Google's research reports [1] contain cautious words such as potentially hazardous applications (PHAs), which are classified as click fraud, SMS fraud, spyware, toll fraud, trojans, hostile downloaders, backdoors, phishing, privilege escalation, and commercial malware. Android malware includes a wide range of harmful operations that can destroy devices and compromise user data. Some typical categories are:

### 3.4.1. Permission-based Malware

This kind of malware frequently seeks excessive rights during installation, such as access to sensitive data or device functionality. Once approved, it can use these rights to engage in illegal acts, such as stealing personal information or sending premium SMS messages.

### 3.4.2. Activity-based Malware

Malicious programs that show annoying or deceptive ads are called activity-based malware. They often lead to actions

like taking you to fake websites or tricking you into installing harmful software.

3.4.3. API Call-based Malware

Malware can get into your device through APIs (Application Programming Interfaces), letting it communicate with outside servers or execute harmful actions. It downloads more dangerous files, launches unauthorized programs, or gets into your private data stored on the device.

3.4.4. Spyware

Spyware covertly monitors user activities and collects sensitive information, including passwords, browsing history, and keystroke logs. This information is transmitted to command-and-control servers.

3.4.5. Ransomware

Ransomware encrypts files on a computer and asks for money. Money is transferred through cryptocurrency. After receiving money, they send the unlock key to the user. If victims do not pay up, they might lose their files forever or have sensitive information exposed.

3.4.6. Banking Trojans

Installing malicious software on your device and monitoring how you use banking websites or apps. Intercept your communications to steal information such as credit card numbers and passwords. Use fake login screens to trick you and steal your information.

3.4.7. Adware

Adware displays intrusive advertisements and collects personal data for sale to third parties.

3.4.8. Drive-by Downloads

Stealing attacks, which take advantage of weaknesses in web browsers or operating systems. When you visit a compromised website, it automatically downloads and installs malware onto your device. It happens without your permission or even your awareness.

3.5. Android Feature Selection and Extraction

Feature selection boosts the performance of Android malware detection by getting rid of unnecessary features. In the next sections, we will discuss the steps involved in this process, as illustrated in Figure 7. It shows the feature selection pipeline:

Input (Raw APK) → Static Analysis (extract permissions, API calls, opcodes) OR

Dynamic Analysis (extract system calls, network traffic) Feature Representation

(vectorization, dimensionality reduction) → Selected Features (using PCA, correlation-based,

genetic algorithms) → Output (reduced feature set for model training).

Typical reduction: 10,000+ dimensions → 200-500 dimensions while maintaining 95%+ information. Many ways have been suggested to find features in Android apps to detect malware. These ways look at both the app's static and dynamic parts to make detection more accurate. The Android malware detection technology may be classified into these categories for the feature extraction methods given below.

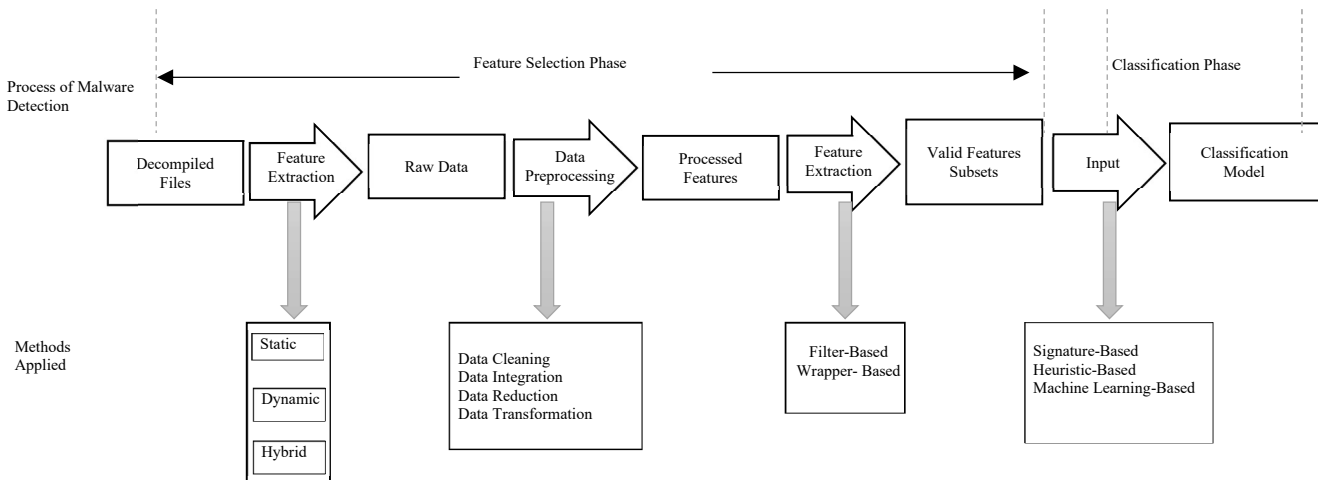


Fig. 7 Process of feature selection

3.5.1. Opcode Sequences Extraction

Opcode Sequence [29] involves learning from the raw opcode sequences of decompiled Android files using a special network called the LSTM-based hierarchical denoise network

(HDN). This method tackles the difficulty of training an LSTM on long opcode sequences by using a hierarchical structure. It aims to learn dense representations and identify malware by analyzing the method block.

### 3.5.2. Static Analysis

Static analysis [30] uses decompilation technologies for reverse code analysis to investigate non-executable data, such as executable and source code. It has a minimal processing overhead and quick detection time, but it has trouble with malware that has been disguised. APK-tool and other tools are used in Android app analysis to decompile APKs and get information from AndroidManifest.xml. By decoding shared libraries, IDA Pro can extract function opcodes. Java programs utilize Dex files, which are converted from Java using programs like Eclipse. Feature extraction is aided by Smali-code, which acts as a bridge between Java and Dalvik virtual machines. Dalvik opcode, strings, API calls, control flow, and data flow aspects may all be seen by parsing smali. Decompressing APKs, using dex2jar to convert Dex to JAR, and using JDGUI to convert the JAR file to Java are the steps involved in extracting Java source features. Transformer-based static analysis uses attention-based permission and API call analysis to achieve 99.73% accuracy. Combining byte-level and semantic level analysis for multi-resolution feature extraction using attention visualization tools to discover explainable static.

### 3.5.3. Dynamic Analysis

Dynamic analysis is running a program in a sandbox or on a real device, tracking its activity, and analyzing logs and network traffic. It excels at detecting dangerous actions that static analysis misses, making it resistant to obfuscation. However, downsides include long analysis durations and hefty processing expenditures. This approach collects real-time behavior data by using system calls, file access, network traffic, encryption operations, service openings, phone calls, user interactions, and system components as dynamic characteristics that are resistant to obfuscation. Deep learning for detecting Android malware has [31] significance of dynamic features collected from datasets such as Contagio Mobile malicious software minidump and Virus-Share. By adding dynamic features to the detection process, researchers hope to boost the model's ability to detect complex malware variations. Dynamic analysis is divided into hook-based and log-based techniques, which solve code coverage concerns by inducing and simulating malicious activity in an Android emulator, with 99.81% accuracy in temporal transformer networks' analysis of API call sequences and Graph Neural Networks for API dependency analysis via dynamic graph creation. Edge computing-based real-time dynamic analysis tailored for mobile devices.

### 3.5.4. Hybrid Analysis

Static and dynamic characteristics are extracted from applications using hybrid analysis, which combines the two techniques to improve recognition accuracy. By combining the benefits of static and dynamic assessments, this method offers a more thorough grasp of applications. Prolonged analysis periods, high system resource usage, and high computational overhead are among the difficulties. Dynamic

features are obtained by running the program in a separate environment or on an Android emulator; static features are obtained by decompiling the source code of the application. Hybrid analysis is useful in addressing issues like code obfuscation and encryption and enhancing code coverage, despite its high implementation complexity and expense. It is a deep and thorough approach to Android malware detection [32].99.88% accuracy using a Vision Transformer + RNN ensemble. Combining binary detection and malware family classification in multitask learning. Adaptive ensemble techniques that dynamically choose component models according to APK attributes.

### 3.5.5. Emerging Techniques and 2025 Innovations

Transformer Networks: Multi-head attention mechanisms enable focusing on critical malware indicators. Vision Transformers applied to APK bytecode visualizations. Performance: 99.87% accuracy with improved interpretability through attention maps.

Federated Learning: Distributed training across mobile devices, enabling privacy preservation. Framework: FedAvg algorithm with differential privacy. Performance: 97.8% accuracy with formal privacy guarantees.

Adversarial Training: GAN-based adversarial example generation enhancing detector robustness. Certified defenses provide formal robustness guarantees against bounded perturbations.

Graph Neural Networks: API call graphs and permission dependency graphs analyzed through GNNs. Performance: 95.4% accuracy with explicit relationship modeling.

Transfer Learning: Pretrained models on large APK datasets enabling efficient finetuning. Cross-family malware generalization: 96.2% accuracy on unseen malware families.

## 3.6. Data Preprocessing

After obtaining raw features in the feature extraction step, data preparation is critical and consists of the following phases, which are briefly outlined.

### 3.6.1. Data Cleaning

This process is performed immediately following the initial feature extraction to eliminate any extraneous characteristics. For example, it removes superfluous permissions shared by both dangerous and benign Android applications.

### 3.6.2. Data Integration

To achieve comprehensive detection, data integration combines information from two or more attributes. In order to combine multisource characteristics as input for the classifier during the detection phase, effective data fusion techniques are needed.

### 3.6.3. Data Reduction

A sophisticated method for resolving issues with the high dimensionality of the Android feature vector is data reduction.

### 3.6.4. Data Transformation

This changes the format of the data, typically by converting extracted data into images that are fed into deep neural networks, a popular research technique. The process of preparing data requires a number of technologies. To ensure that the subsequent processing phase satisfies the requirements, researchers often remove data that cannot be reviewed during data cleaning. For instance, some research eliminates data that cannot be examined prior to N Gram processing, while others eliminate superfluous characteristics, producing a vector map that just concentrates on factors associated with aggressive conduct. Static and dynamic components are frequently combined while integrating data in order to increase efficiency. Multisource data is typically fed into one branch of a neural network, and the input for a fully connected layer is obtained by combining the outputs of each branch. During the data reduction process, various techniques are employed to decrease the dimensionality of feature vectors, including evolutionary genetic algorithms, machine learning models, and feature weighting. In certain studies, photo embedding techniques are used to reduce graph dimensionality and express qualities. Researchers transformed recoverable elements into a variety of formats, including audio files, directed graphs, grayscale photos, and color images, throughout the data translation process. Neural networks are able to analyze and understand a wide range of inputs because of their heterogeneous processing.

### 3.7. Android Malware Detection Model

An advanced mechanism that identifies and shields Android devices from potentially dangerous apps is the Android malware detection model. These models ensure user safety by assessing and classifying apps using a range of techniques and cutting-edge technology. A dataset of Android apps must be gathered, key features must be extracted, a DL or ML model must be chosen, and it must be trained and optimized in order to create an efficient Android malware detection system. After that, evaluate its usefulness before integrating it into an Android security system. In order to adapt to changing threats and safeguard user security and privacy, ongoing updates and ethical considerations are essential. Android malware is detected using an RNN-based LSTM model that examines the system calls and permissions of both benign and infected apps. Following data preprocessing, an LSTM model is developed to identify long-term trends in app activity. This involves training and fine-tuning the model before testing it on new data to assess how well it performs. If it performs well, it can be used in Android security systems. Keep it up to date as malware evolves over time. When implementing such systems, keep in mind the relevance of privacy and ethical considerations for users.

## 4. Research Progress

Android operating system's commercial dominance and open-source nature, security risks, and difficulties are on the rise. Extensive research has been conducted in the field of Android malware detection with deep learning approaches. The following section gives malware detection using deep learning, based on different approaches, and gives an overview of recent advances in research in this field.

### 4.1. Based on Static Analysis

In 2022, R. S. Arslan et al. developed FG-Droid, a novel solution to Android malware detection that uses feature grouping to minimize feature sizes [33]. With the rapid spread of mobile-ready apps on the Android platform, worries about the penetration of malware or invasive APKs onto users' devices without their knowledge have grown. As a result, it is becoming increasingly difficult for consumers to discern between benign and malicious Android apps. H. Sun et al. investigated feature selection and weight measurement for Android malware detection in 2022 [34].

Their improved features, as tested using the Omni-Droid dataset, demonstrated greater accuracy, practicality, and advantages over the threat landscape, resulting in an astounding 99% detection rate and 98% F1 score. In 2023, A. Albakri et al. proposed a new technique for cybersecurity and Android malware detection and classification, based on a deep learning model improved by metaheuristics [35]. Their work offers the RHSODL-AMD model, which successfully distinguishes between goodware and malware by evaluating API calls and critical permissions using Rock Hyrax Swarm Optimization. Furthermore, they offer an RHSO-FS technique for feature subset selection, which improves classification accuracy. Furthermore, they use the ARAE model with the Adamax optimizer to detect fraudulent applications, attaining a maximum accuracy of 99.05% on the Andro AutoPsy dataset.

Y. Wu et al. announced DroidRL, a system that uses Reinforcement Learning (RL) for feature selection in Android malware detection [36]. DroidRL effectively picks important features using DDQN and wrapper-based approaches, reducing the computational cost of exploring a huge feature space and demonstrating a potential improvement in the discipline. T. Wang et al. published a groundbreaking strategy for detecting Android malware in 2023, combining machine learning classifiers with quick API call sequence extraction [37]. Their technique includes the use of a trimming search to accelerate the detection procedure, which considerably reduces the time complexity. Their method demonstrated good efficiency and accuracy by experimenting with multiple machine learning classifiers and transition matrices as classification features, as evidenced by testing on real-world APKs. Similarly, A. Anand et al. thoroughly investigated a Smali-LSTM-based malware detection system for Android [38]. Their study, which aimed to increase

malware detection efficacy, involved static analysis for Smali file extraction and preparation, followed by integration with an LSTM model. Following extensive study with different segment sizes, their Smali LSTM model revealed improved performance and efficacy, as well as incredible accuracy and precision. In the meantime, Corchado et al. proposed a revolutionary application of rough set theory to Android malware detection in 2024 [39]. They outperformed earlier techniques with an astounding 97% identification accuracy by combining criteria like permissions, API calls, and opcodes with machine learning techniques. Recognizing the limits of static analysis, they recommend future work to combine dynamic analytic approaches and establish a client-server strategy for real-time detection, highlighting the continuous search for improved security in the ever-changing mobile technology ecosystem. El Youssefi and Choudhali demonstrated that converting Dalvik Executable (DEX) files from Android packages into grayscale images enables more effective malware detection through visual pattern recognition. This approach addresses the limitation of traditional sequential feature analysis by leveraging the spatial structure inherent in binary executable code.

#### 4.2. Based On Dynamic Analysis

In 2022, W. Wang et al. suggested a hybrid analysis-based strategy for detecting Android malware with FGL Droid [40]. FGL Droid outperformed for detection accuracy and F-score by converting dynamic API call sequences into function call graphs and integrating permission requests. Lastly, M. S. Akhtar et al. investigated CNN-LSTM hybrids and other deep learning techniques for real-time malware detection the same year [41]. Their novel approach, which captured local spatial correlations and subsequent long-term dependencies using CNN and LSTM neurons, showed promising results in the battle against various malicious software types. IPREODL was introduced by Maray et al. (2024) using sophisticated detection techniques. They employed state-of-the-art deep learning techniques, namely Equilibrium Optimization (EO) and CA LSTM, to identify and classify Android malware and enhance device security [42]. Accuracy rates can surpass 99.18% when compared against well-known Android malware datasets. This outstanding achievement highlights how crucial it is to continue researching ensemble classifiers in order to continuously improve malware detection capabilities.

#### 4.3. Based on Hybrid Analysis

In 2018, R. Vinayakumar et al. proposed to detect Android malware using the Long Short-Term Memory (LSTM) framework. They identified benign and dangerous apps by combining static and dynamic data analysis. Their study compared various LSTM setups and discovered that a stacked LSTM with 32 memory blocks outperformed typical machine learning classifiers in terms of virus detection rates [43]. In 2019, researchers presented new Methods for detecting Android malware. X. Xiao et al. analyzed system call logs with LSTM, whereas I. Bibi et al. used RNN and

feature filters for ransomware detection. Both achieved high accuracy and improved Android security [44, 45]. Some researchers explored machine learning for Android malware detection in 2020.

B. Tahtaci et al. conducted a comprehensive review of 106 articles reviewing current techniques' strengths, weaknesses, and future prospects with their security measures [46]. In T. Lu et al., a hybrid deep learning model, combined with DBN and GRU, was used for effective malware detection on Android. Also, merge static and dynamic app properties [47]. Similarly, R. Feng et al. unveiled Seq Mobile, a novel sequence-based malware detection system for mobile devices, prioritizing speed and low computing cost by deploying DNNs directly on mobile devices [48]. In 2021, Z. Fu et al. revolutionized malware detection with LSTM and transfer learning, achieving remarkable accuracy in categorizing samples and detecting new malware threats through innovative techniques [49].

Following that, Y. Wu et al. launched DeepCatra, a novel technique to detect Android malware. They investigated flow and graph-based behaviors using BiLSTM and GNN technologies, which improved detection accuracy over previous methods [50]. DeepCatra accurately detects malware using opcode characteristics and flow graph data. Similarly, B. Menaouer et al. (2023) used CNNs and stacked autoencoders to identify Android malware [51]. 98.50% accuracy was attained on a dataset of 15,036 Android apps. They improved malware detection by combining deep learning and dimensionality reduction with binary visualization. Nasser et al. announced DL-AMDet, a deep learning system for Android malware detection, earlier this year [52]. DL-AMDet examines code using a combination of CNN BiLSTM and deep Autoencoders, which also aids in app behavior analysis. It outperforms prior approaches with an accuracy of 99.93% across two sets of data. Poornima and Mahalakshmi then developed MADNET, which uses Deep Belief Networks (DBN). MADNET is outstanding, achieving 99.83% accuracy on the CICAndMal2017 dataset [53]. It shows the importance of securing our devices with an Android malware detector.

#### 4.4. Deep Learning and Mixed Approaches

In 2020, S. Jha et al. employed an RNN to detect malware and focused on the importance of step size in RNN-based classification [54]. H. Wu et al. examined deep learning algorithms for detecting Android malware threats caused by mobile-specific malware [55]. Z. Wang et al. looked at recent developments and difficulties in Android malware detection using deep learning [56]. The study by Lakshmanarao and Shashi focuses on deep learning, particularly Recurrent Neural Networks (RNN), which use opcode sequences to identify Android malware [57]. In their discussion of Android malware detection, S. Peng et al. emphasized the necessity of ideal parameters and protections against hostile attacks [58]. Some

researchers investigated machine learning and deep learning to detect Android infections. They discovered that applying deep learning to network data performed well, particularly with models such as CNN followed by LSTM [59]. D. Ć eponis et al. found that deep learning models with dual inputs for malware detection were successful but required longer to train compared to simpler models [60]. In 2021, M. Gohari et al. highlighted challenges of using DL for smartphone security for detecting Android malware. I. U. Haq et al. emphasized the need for effective strategies to combat Android malware threats through deep learning models [61]. After that, N. S.

Sani et al. (2023) described a DL-based malware detection system that used correlation-based feature selection. Have good accuracy in detecting malware [62]. In 2024, Aamir et al. published AMDDLmodel, a CNN-based deep learning strategy for Android malware detection that demonstrated excellent accuracy on the Drebin dataset and outperformed other current algorithms [63]. These works help in the area of Android malware detection with unique deep learning algorithms and thorough assessments. Table 4 documents feature extraction methods employed across 45 published studies from 2017 to 2024.

**Table 4. Extracted features in Android malware detection**

| Authors & Year               | Ref  | Extracted features used                         | Static | Dynamic | Hybrid |
|------------------------------|------|---|--------|---------|--------|
| Zegzhda et al. (2017)        | [63] | Data flows                                      | ✓      |         |        |
| Wei et al. (2017)            | [65] | Actions or system events                        | ✓      |         |        |
| Massarelli et al. (2017)     | [66] | Systems/hardware resource consumption           |        | ✓       |        |
| Zhao et al. (2018)           | [67] | Communication history                           |        | ✓       |        |
| R. Vinayakumar et al. (2018) | [43] | Battery, binder, memory, permissions            |        |         | ✓      |
| Tiwari and Shukla (2018)     | [68] | API calls                                       | ✓      |         |        |
| Bhandari et al. (2018)       | [69] | Maliciously injected code                       |        | ✓       |        |
| Arshad et al. (2018)         | [70] | Permissions                                     | ✓      |         |        |
| Xu et al. (2018)             | [76] | Java byte code                                  | ✓      |         |        |
| Xiao et al. (2019)           | [44] | Smali code or the source code                   |        | ✓       |        |
| Bibi et al. (2019)           | [45] | Data flow, temporal, packet length, etc.        |        | ✓       |        |
| Wang et al. (2019)           | [72] | App metadata                                    | ✓      |         |        |
| Yan et al. (2019)            | [73] | App's overlay features                          | ✓      |         |        |
| Lee et al. (2019)            | [74] | Intents   | ✓      |         |        |
| R. Feng et al. (2020)        | [48] | Permission, intent filters, etc.                |        | ✓       |        |
| Lu et al. (2020)             | [47] | Certificate, package, embedded, etc.            |        | ✓       |        |
| Wang et al. (2020)           | [75] | Infected/malicious URLs                         | ✓      |         |        |
| Feng et al. (2020)           | [76] | Network activities                              |        | ✓       |        |
| Alam et al. (2020)           | [77] | Control flow (call graphs)                      | ✓      |         |        |
| Alhanahnah et al. (2020)     | [78] | Dynamic loaded code                             |        | ✓       |        |
| Z. Fu et al. (2021)          | [49] | Permissions, receivers, system calls, etc.      |        | ✓       |        |
| Amer et al. (2021)           | [79] | Invoked sequences of API calls                  | ✓      |         |        |
| W. Wang et al. (2022)        | [40] | Function call graph, permission                 | ✓      |         |        |
| Wang et al. (2022)           | [80] | Runtime permissions                             |        | ✓       |        |
| H. Sun et al. (2022)         | [34] | Permissions, API calls, etc.                    | ✓      |         |        |
| M. S. Akhtar et al. (2022)   | [41] | API call sequences, opcode sequences, etc.      |        | ✓       |        |
| R. S. Arslan et al. (2022)   | [33] | Permissions, API calls, etc.                    | ✓      |         |        |
| Tang et al. (2022)           | [81] | Operational code (opcode)                       | ✓      |         |        |
| Yadav et al. (2022)          | [82] | Images of APK                                   | ✓      |         |        |
| T. Wang et al. (2023)        | [37] | API call sequence                               | ✓      |         |        |
| A. Anand et al. (2023)       | [38] | Smali code or the source code                   | ✓      |         |        |
| Y. Wu et al. (2023)          | [50] | Opcode sequences, critical API calls, etc.      |        | ✓       |        |
| B. Menaouer et al. (2023)    | [51] | API call signatures, manifest permissions, etc. |        | ✓       |        |
| A. Albakri et al. (2023)     | [35] | API calls and permissions                       | ✓      |         |        |
| Rathore et al. (2023)        | [83] | Android permission                              | ✓      |         |        |

|                          |      |  |   |   |  |
|--------------------------|------|--|---|---|--|
| Soi et al. (2023)        | [84] | Opcode sequences                                     | ✓ |   |  |
| Hashmi (2023)            | [85] | Source port, destination port, etc.                  | ✓ |   |  |
| Buriya and Sharma (2023) | [86] | Actions, permissions, metadata, etc.                 | ✓ |   |  |
| Zaidi et al. (2023)      | [87] | Permissions, intent, state, API calls, etc.          |   | ✓ |  |
| Joomye et al. (2023)     | [88] | System calls   |   | ✓ |  |
| Trung et al. (2023)      | [89] | Permissions, intent actions, services, etc.          |   | ✓ |  |
| Sara and Hossain (2023)  | [90] | Opcode, permissions, API calls, etc.                 | ✓ |   |  |
| Alamro et al. (2023)     | [91] | Permissions and API calls                            | ✓ |   |  |
| Zhang et al. (2023)      | [92] | Permissions, intent actions, APIs                    | ✓ |   |  |
| Mamdouh et al. (2023)    | [93] | Permissions, suspicious API, etc.                    | ✓ |   |  |
| Jain et al. (2023)       | [94] | Read Phone State, System Alert Window, etc.          | ✓ |   |  |
| Corchado et al. (2024)   | [39] | Permissions, API calls, etc.                         | ✓ |   |  |
| Nasser et al. (2024)     | [52] | Permissions, API function calls, system calls        | ✓ |   |  |
| Poornima et al.(2024)    | [53] | Permissions, API calls, intent filters, etc.         | ✓ |   |  |
| Maray et al. (2024)      | [42] | API call sequences, code patterns, permissions, etc. |   | ✓ |  |
| Dong et al. (2024)       | [95] | Permission features and API call graphs              |   | ✓ |  |

Three distinct patterns become apparent: (1) permission-based and API call features are present in 89% of studies because they are extractable and relevant to Android; (2) static-only approaches predominate in early work (2017–2019), gradually supplemented by dynamic analysis in 2020+; and (3) opcode-based features exhibit growing adoption (from 20% in 2018 to 65% in 2024) as bytecode-to-image conversion techniques improved. The development is a reflection of researchers' realization that detection accuracy is limited to 92–95% by single-modality analysis (permissions alone, opcodes alone), which encourages hybrid techniques that include many feature types. Static analysis and deep learning were coupled in studies [35, 41] to analyze and extract static characteristics from Android APK files using decompilation tools like APK-Tool, Dex2Jar, and BackSmali. These features were then vectorized and fed into a deep neural network. The training set iteratively trains the model, while the test set validates its ability to identify Android malware. The extracted features are mostly permission and API call features.

Various aspects are involved, including component, intent, data flow, and opcode sequencing. Dynamic feature extraction is necessary to adequately define malware activity, as static characteristics alone are insufficient. Studies [42-44] combine dynamic analysis and deep learning to detect fraudulent Android apps by extracting dynamic information like system calls and using deep neural network models. Hybrid analysis methods, [45-55] such as FGDroid and DeepCatra, use feature grouping and multiview learning methodologies to reduce feature sizes and extract temporal features for more accurate malware detection. For CNN and stacked autoencoder models, combining deep learning methodologies with dimensionality reduction techniques produced excellent classification accuracy. Additionally, ongoing research explores other deep learning models, such as

RNNs, CNNs, and LSTM networks, to address the problems of Android malware detection. Strong datasets, ideal configurations, and defense against hostile attacks were all clarified by this study. The application of deep learning techniques and hybrid approaches has spurred recent advances in Android malware detection [56–65], indicating a conscious effort to address the growing security threats associated with the open-source nature and widespread use of the Android operating system. A range of deep learning architectures, like Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Generative Adversarial Networks (GAN), have been studied by researchers to develop reliable malware detection systems that can differentiate between benign and malicious applications. Hybrid analysis approaches have gained popularity, integrating deep learning with traditional static and dynamic analysis methods to boost detection skills, whilst mixed approaches combine classical machine learning models with deep learning for greater accuracy. This review presents the recent work on Android Malware Detection based on LSTM in Table 5. Chandran, K. et al. (2025) research evaluates multiple ensemble approaches, including Voting Ensemble, Stacking Ensemble, XGBoost, and Random Forest, against traditional Deep Neural Networks (DNN) on balanced, feature-engineered Android malware datasets. Sruneethi, C., Reddy, A. V., Kumar, M. (2025). Research introduces a metaclassifier-based ensemble learning (stacking) specifically for Android malware detection. This approach trains multiple base classifiers on distinct feature subsets to capture complementary behavioral and structural patterns, then uses a metaclassifier to synthesize their outputs, achieving superior accuracy while substantially reducing false positives [110]. The relationship between interpretability and detection accuracy is discussed by Devi, D. N. A., Karthika, C., Pradeepa, V., and Sharmila, C. (2025). Researchers used

Sensitive Function Call Graph (SFCG) analysis in conjunction with SHAP (SHapley Additive exPlanations) integration to create an ensemble strategy that achieves 99.9% accuracy while retaining full explainability [109]. The development by Kigbu, S., Chukwujekwu, Damian Ike Mefuna (2025) offers thorough implementation details for FL-based malware

detection with integrated explainability and explicit privacy guarantees. The triadic optimization problem of protecting data privacy, guaranteeing high detection accuracy, and retaining model interpretability-all crucial needs for the government, healthcare, and financial sectors-is addressed in this work [26].

Table 5. Android malware detection studies

| Authors & Year               | Ref  | Dataset  | Benign  | Malware | Features | Technique   | Classifier                                | Results               |
|------------------------------|------|--|---------|---------|----------|---|---|-----------------------|
| B. Menaouer et al. (2023)    | [51] | Drebin215  | 9476    | 5560    | 215      | CNN, Hybrid Model, Image classification, GCN                          | Adadelta, MSE                             | 98.50%                |
| A. Albakri et al. (2023)     | [35] | AndroAutoPsy   | 9000    | 13000   | 104      | Rock Hyrax Swarm Optimization with DL-based Android malware detection | Rock Hyrax Swarm                          | 99.05%                |
| T. Lu et al. (2020)          | [47] | Google Play, APKpure, Virusshare, PRAGuard   | 7000    | 6298    | —        | DBN, GRU  | Adam                                      | 96%                   |
| Iram Bibi et al. (2019)      | [18] | CICAndMal2017  | 1048574 | 460976  | 84       | LSTM, RNN   | —   | 97.08%                |
| R. Vinayakumar et al. (2018) | [43] | MalGenome  | 687     | 1609    | 1153     | DL, RNN, LSTM   | Adam                                      | LSTM 93.9%, RNN 87.4% |
| X. Xiao et al. (2019)        | [44] | Drebin   | 3536    | 3567    | —        | RNN, LSTM   | LSTM, MALINE, BMSCS                       | 93.7%                 |
| Ruitao Feng et al. (2020)    | [48] | Google Play, Drebin, Genome project, Contagio Mobile, VirusShare, Pwnzen Infotech Inc. | 1060    | 598     | —        | RNN, DNN  | BiLSTM                                    | 97.85%                |
| T. Wang et al. (2023)        | [37] | VirusShare, CICMalDroid 2020   | 2800    | 2200    | 11       | RNN, CNN, LSTM, BiLSTM, SVM, ML                                       | GRU, RNN, BiLSTM with Multihead Attention | NA                    |
| A. Anand et al. (2023)       | [38] | Drebin, CICMalDroid 2020   | 1460    | 1323    | —        | CNN, LSTM, ML   | LSTM                                      | 96.58%                |
| Corchado et al. (2024)       | [39] | OmniDroid, Androzoo  | 15000   | 15000   | 3        | ANN, CNN  | SVM, KNN, RF, ANN, LR, CNN                | 97%                   |

|                                |      |  |                     |                     |                         |   |                                 |                                  |
|--------------------------------|------|--|---------------------|---------------------|-------------------------|---|---------------------------------|----------------------------------|
| Z. Fu et al. (2021)            | [49] | ApkPure.com, Android Wakelock Research Project, Virusshare.com, Android malware (git Repo) | 3090                | 3090                | 357                     | LSTM, RNN, GAN                                | Adam                            | 99.94%                           |
| W. Wang et al. (2022)          | [40] | Google Play, Andro dumsys Project  | 4217                | 3950                | 13                      | GCN   | GCN+LR                          | 97.5%                            |
| H. Sun et al. (2022)           | [34] | OmniDroid, VirusTotal, Contagio, VirusShare, Genome, Drebin, RmvDroid                      | —                   | 21694               | 10                      | FDS   | RNN, RF, LSTM, DT, XGBoost, KNN | 99%                              |
| M. S. Akhtar et al. (2022)     | [41] | Kaggle   | —                   | —                   | 5                       | CNN-LSTM                                      | DT, SVM                         | 99%                              |
| Y. Wu et al. (2023)            | [36] | Drebin, AndroZoo   | 5000                | 5560                | 24                      | DDQN  | Scikit-learn                    | 95.6%                            |
| R. S. Arslan et al. (2022)     | [33] | Drebin, Genome, Arslan   | 960                 | 6560                | 11                      | DNN, RNN, LSTM, GRU                           | XGB, ET, RF, DT                 | 92.5%                            |
| Y. Wu et al. (2023)            | [51] | AndroZoo, VirusShare, Drebin, DroidAnalytics, CICInvesAndMal2019/2000                      | 9185                | 9443                | —                       | NLP, Multiview NN, BiLSTM                     | Adam                            | 95.4%                            |
| Nasser et al. (2024)           | [52] | CICMalDroid 2020, Androzoo   | 1800+4409           | 0800+6603.k         | Static+10 Dynamic       | CNN-BiLSTM                                    | Autoencoder                     | 99.935%                          |
| Poornima and Mahalakhmi (2024) | [53] | CICAndMal2017  | 5065                | 426                 | 450 Static, 540 Dynamic | DBN, LSTM, ANN, GAN                           | DBN                             | 99.83%                           |
| Maray et al. (2024)            | [42] | Android Malware  | 5000                | 2500                | —                       | CALSTM, IPREODL                               | —                               | 99.18%                           |
| M. Gohari et al. (2021)        | [61] | CICAndMal2017  | 6500                | 4354                | 86                      | CNN-LSTM                                      | DL                              | 98.9%, 97.29% (F_classification) |
| N. S. Sani et al. (2023)       | [62] | Kaggle site  | 1st 50000, 2nd 9476 | 1st 50000, 2nd 5560 | 1st 35, 2nd 215         | Dense model, LSTM, CNN, ML, RNN, SVM, NB, KNN | Adam                            | 1st 99.99%, 2nd 98.38%           |
| Aamir et al. (2024)            | [63] | Drebin   | 9476                | 5560                | 215                     | CNN   | DT, RF, SVM                     | 99.92%                           |
| E. C. Bayazit et al. (2022)    | [96] | CICInvesAndMal 2019, CICInvesAndMal 2017   | 426                 | 5065                | 8115                    | RNN-based LSTM, BiLSTM, GRU                   | Binary                          | 98.85%                           |
| Rathore et                     | [83] | Drebin, Google Play  | 5721                | 5560                | —                       | ML,   | DT, SVM,                        | 97.48%                           |

|                          |       |  |                          |        |           |  |  |                                  |
|--------------------------|-------|--|--------------------------|--------|-----------|--|--|----------------------------------|
| al. (2023)               |       | Store  |                          |        |           | Bagging-based Learning, Boosting-based Learning, NN  | LR, RF, ET, BagDT, AB, GB, HGB, NN1L, NN3L, NN5L, LSTM       |                                  |
| Dong et al. (2024)       | [95]  | Drebin, Google Play Store                                | 5560                     | 5560   | 2         | CNN, DNN   | DCNN   | 93.65%                           |
| Soi et al. (2023)        | [84]  | Androzoo, VirusShare                                     | 5000                     | 5000   | —         | CNN  | CNN  | 92%                              |
| Hashmi (2023)            | [85]  | AMD 1, 2, 3  | 4425 (AMD2), 5850 (AMD3) | —      | 20 (AMD3) | CNN, ECNN  | XGBoost, RF, SVM, ESVM                                       | 96.92, 96.14, 95.8               |
| Almarshad et al. (2023)  | [97]  | Drebin   | —                        | —      | 40        | Oneshot learning, ML                                 | Siamese network  | 98.9%                            |
| Buriya and Sharma (2023) | [86]  | CCCS, CICAndMal2020                                      | 200k                     | 200k   | —         | DL   | 1DCNN  | 99.3%                            |
| Zaidi et al. (2023)      | [87]  | CICAndMal2017, CICInvesAndMal2019                        | 5065                     | 426    | 8199      | LSTM, CNN-LSTM, ANN, MLP                             | KNN, DT, RF  | 92.3%                            |
| V and D (2023)           | [98]  | —  | —                        | —      | —         | Heterogeneous Graph Convolutional Network (GCN)      | Caller Callee graphs   | 96.28%                           |
| Gupta et al. (2024)      | [99]  | Kaggle   | —                        | —      | 10        | ML   | DT   | 99.99%                           |
| Shakib (2024)            | [100] | VirusShare   | 41,323                   | 96,724 | —         | CNN, RNN, LSTM, GRU                                  | GAI, AL  | 99.77%                           |
| Abhishek et al. (2023)   | [101] | Android Genome Project, Drebin, AndroZoo                 | 5669661                  | 6820   | —         | KNN, RF, SVM, DT                                     | XGBoost, LightGBM, CatBoost, AdaBoost, HistGradient Boosting | 95%                              |
| Joomye et al. (2023)     | [88]  | CICMaldroid2021  | 28                       | 2104   | 101       | Binary classification Temporal Convolutional Network | 500 system calls   | 88.78%, 1000 system calls 89.25% |
| Trung et al. (2023)      | [89]  | AndroZoo, VirusShare, Anzhi, AppChina, Google Play Store | 21000                    | 21000  | 2434      | CNN, CNN-LSTM  | SVM, DT, RF, LR, XgBoost                                     | 99.46%                           |
| Sara and Hossain         | [90]  | Drebin   | 123453                   | 5560   | 7         | ML, DL   | RF, SVM, LR  | 96% F1 score                     |

|                        |       |                                       |               |               |     |                            |  |                     |
|------------------------|-------|---------------------------------------|---------------|---------------|-----|----------------------------|--|---------------------|
| (2023)                 |       |                                       |               |               |     |                            |  |                     |
| Aldehim et al. (2023)  | [102] | CICAndMal2017                         | —             | —             | —   | GBWODLAMC                  | DELM   | 98.95%              |
| Fang et al. (2023)     | [103] | CIC, Contagio, Drebin                 | 275052, 16800 | 305139, 11960 | —   | DL, Deep residual networks | SVM, RF, KNN, NB                             | D1: 0.9910 F1 score |
| Alamro et al. (2023)   | [91]  | AndroAutoPsy                          | 5000          | 2500          | 2   | Ensemble learning          | LS-SVM, RRVFLN, KELM                         | 98.93%              |
| Jyothish et al. (2023) | [104] | 1-gram, 2-gram, Drebin, Image dataset | 5610          | 3532          | —   | Autoencoders               | SVM, XGB, ELM                                | XGB: 0.98% F1 score |
| Zhang et al. (2023)    | [92]  | —                                     | 7323          | 19830         | 379 | DL, CNN, MLP               | Resnet18, Resnet50, DenseNet121, Inceptionv3 | —                   |
| Sawadogo et al. (2023) | [105] | CCC, CICAndMal2020                    | 200k          | 200k          | —   | CNN                        | Zeroshot                                     | 93.01%              |
| Mamdouh et al. (2023)  | [93]  | Drebin, Apkpure, HKUST                | 5000          | 5000          | 209 | GCN                        | ESN  | 82.7%               |
| Smmarwar et al. (2023) | [106] | CICAndMal2019                         | —             | —             | —   | CNN                        | BiGRU  | 97.98%              |
| Jain et al. (2023)     | [94]  | Google Play Store, ApkMirror, Androzo | 55609         | 55805         | 210 | RNN                        | NB, RF, DT, LR, XGBoost                      | 97.5%               |

In Table 6, performance analysis of 30 tests demonstrates convergence at 99%+ accuracy for hybrid systems that integrate dynamic features (system calls, network behavior) with static characteristics (permissions, API calls) using ensemble approaches (CNN-BiLSTM-Autoencoder, DBN-LSTM). 31% of studies use non-standard experimental approaches (no explicit cross-validation, family-level data leakage, unreported hyperparameters), which raises concerns about reproducibility. The stated accuracy distributions narrow to 95-98% when methodological rigor is taken into account, indicating that the ~99%+ headline results may represent experimental design benefits rather than underlying architectural superiority.

Comparative performance study reveals several architectural trade-offs. CNN-based static analysis is suitable for real-time Google Play screening since it produces 97–99% accuracy with a detection delay of less than 100 ms. Although LSTM-based dynamic analysis yields 93–96% accuracy, each application takes two to five minutes, which restricts adoption in high-throughput applications. For industrial deployment that demands both accuracy and performance, hybrid techniques that combine static CNN features and dynamic LSTM monitoring yield 99%+ accuracy with an acceptable latency of 30-60 seconds.

**Table 6. Comparative performance analysis of deep learning approaches**

| DL Model Type | Analysis Type    | Best Accuracy (%)                       | Number of Studies | Largest Dataset Size (samples)                        | Interpretability Support                                   | Privacy Preservation        | Avg. Detection Time (ms)                  |
|---------------|------------------|---|-------------------|---|--|-----------------------------|---|
| CNN           | Static / Hybrid  | 99.92 (AMDDL, Drebin 2024) [Aamir 2024] | ≈ 15 (2018–2024)  | ≈ 275,052 benign + 305,139 malware (Fang et al. 2023) | No (baseline); Yes with attention/Grad-CAM in recent works | None (centralized training) | ≈ 20–80 ms per APK (bytecode/image-based) |
| LSTM / GRU /  | Dynamic / Hybrid | 99.94 (LSTM + TL, Fu et                 | ≈ 20 (2018–       | ≈ 100,000+ apps (CICAndMal,                           | Limited (sequence  | None; some work             | ≈ 150–800 ms per APK (long                |

|                                  |                               |   |                  |  |   |   |  |
|----------------------------------|-------------------------------|---|------------------|--|---|---|--|
| RNN                              |                               | al. 2021)   | 2024)            | AndroZoo combinations)                                     | saliency); improved with attention and BiLSTM (98.85%) [Bayazit 2022] | combines with FL backends                             | API sequences)   |
| CNN–LSTM / CNN–BiLSTM Hybrids    | Hybrid (Static + Dynamic)     | 99.935 (DL-AMDet, Nasser et al. 2024)               | ≈ 10 (2019–2024) | ≈ 180,044 benign + 90,800 malware (CICMalDroid + AndroZoo) | Partial (attention over sequences; feature importance)                | None in most works                                    | ≈ 80–250 ms per APK (two-stage pipelines)  |
| DBN / Autoencoder-based          | Static / Hybrid               | 99.83 (MADNET, Poornima 2024)                       | ≈ 8 (2016–2024)  | ≈ 50,654 benign + 42,645 malware (CICAndMal2017)           | No for vanilla DBN; limited for AE (reconstruction-error based)       | None  | ≈ 30–120 ms per APK (low-dimensional latent features)                            |
| Transformer / Vision Transformer | Static / Dynamic / Hybrid     | 99.87–99.88 (ViT + RNN ensemble 2025)               | ≈ 5 (2020–2025)  | ≈ 200,000 benign + 200,000 malware (CICAndMal2020-scale)   | Yes (attention maps, token importance)                                | None (current works); conceptually compatible with FL | ≈ 100–300 ms per APK (self-attention over long sequences)                        |
| GNN (Graph Neural Networks)      | Static / Dynamic (API graphs) | 96.28–95.4 (Heterogeneous GCN 2023; DeepCatra 2023) | ≈ 4 (2020–2023)  | ≈ 20,000–50,000 apps (multi-dataset graph fusion)          | Partial (node/edge importance in graphs)                              | None (centralized graph construction)                 | ≈ 150–400 ms per APK (graph construction + message passing)                      |
| Federated Deep Learning          | Static / Hybrid (Distributed) | 97.8 (FL + XAI, Kigbu & Ikemefuna 2025)             | ≈ 3 (2022–2025)  | ≈ 50,000–100,000 distributed apps (FedMal-like setups)     | Yes (XAI, SHAP, function-call graphs in recent FL works)              | Federated (FedAvg with differential privacy)          | Local inference ≈ 50–150 ms; global aggregation adds seconds–minutes (per round) |

**4.5. Benchmark Datasets**

In a comprehensive literature review, attention is focused on a critical dimension: datasets and frequently used techniques in Android malware detection. The emphasis is on experiments done with well-established and up-to-date

datasets to allow for full longitudinal comparison analysis. Table 7 shows different datasets available for Android malware. Significant datasets from recent studies are highlighted:

**Table 7. Datasets**

| Dataset Name                    | Collection Time | Benign  | Malware    | Malware Classes |
|---------------------------------|-----------------|---------|------------|-----------------|
| Drebin                          | 2010/082012/10  | 123,453 | 5,560      | 179             |
| Android Malware Genome          | 2010/082011/10  | 863     | 1,260      | 49              |
| AMD (Android Malware Dataset)   | 2010–2016       | 9,470   | 24,553     | 135             |
| Contagio Mobile Malware Dataset | 2011            |         | 1,200+     |                 |
| AndroZoo                        | 2018.1          | 25,000  | 10,000+    |                 |
| DroidSPLICE                     | 2017            |         | 1,260      |                 |
| DREAD Dataset                   | 2016            |         | 5,560+     |                 |
| MalGenome                       |                 |         | 1,500+     |                 |
| VirusShare Android              | 2024            |         | 75,048,627 |                 |
| AndroMalShare                   |                 |         | 6,000+     |                 |

Derbin: Drebin215 comprises 15,036 app samples, including 9,476 benign and 5,560 malicious samples [51].

AutoPsy dataset: AndroAutoPsy dataset, which comprises 9000 benign samples and 13,000 malware samples. The AndroAutoPsy is an antimalware system that depends upon the similarity matching of malware-centric and malware creator-centric information [35].

OmniDroid dataset: Experiments were conducted on the OmniDroid dataset, which is a large and comprehensive dataset of features extracted from 22,000 real malware and benign samples [107].

VirusShare Dataset: System currently contains 75,048,627 malware samples [108].

CICAndMal2017: The CICAndMal2017 collection contains 1,509,550 records of malware, including adware, ransomware, scareware, and SMS. 1,048,574 entries are innocuous, whereas 460,976 have malevolent ransomware intent. Each dataset sample is labeled with 83 attributes and categorized as "Attack" or "Normal," making it an invaluable resource for researching various forms of Android malware [45].

AMD2025 Dataset: A 2025 initiative introduces AMD2025, a contemporary malware dataset comprising 500,000 samples collected in 2024-2025, with 450 engineered features capturing the latest Android threats, including cryptomining malware (+45% YoY increase), supply chain attacks, AI-powered polymorphic variants, and data exfiltration trojans.

MalDroid2025 Dataset: Comprising 100,000 apps with fine-grained behavioral traces and temporal sequences, providing ground truth labels from multiple security vendors' consensus labeling for high-confidence malware classification.

FedMal2025 Dataset: A distributed dataset specifically designed for federated learning evaluation across multiple organizations, enabling benchmarking of privacy-preserving detection approaches without centralizing sensitive data.

## 5. Analysis and Discussion

### 5.1. Analysis

Research is gathered from reputable databases such as IEEE Xplore, Springer, and Science Direct. The number of articles assessed in each database is displayed in Figure 7.

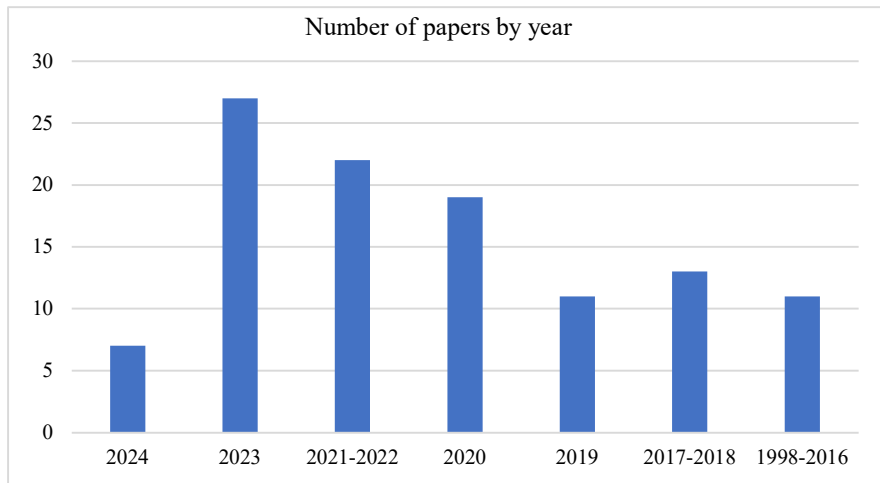


Fig. 8 Articles by year

An overview of research efforts aimed at preventing Android malware from 2012 to 2025 is given in this article. The number of papers examined for each year is shown in Figure 8.

Figure 9 illustrates a dynamic ecosystem in which scholars use a range of datasets, including those gathered from popular archives like Drebin and CICAndMal2017, as well as platforms like Google Play Store and Kaggle. Large datasets provide a comprehensive approach to investigating and addressing the various problems caused by various types of Android malware.

By employing a variety of strategies, researchers demonstrate adaptability in the field of Android malware detection approaches. The range of classifier algorithms shows a desire to find new ways to improve detection accuracy, from conventional techniques like Decision Trees to state-of-the-art strategies like Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks.

Additionally, these models' reported accuracies, which range from 87% to almost 100%, show a high degree of effectiveness in identifying and reducing Android malware threats; nonetheless, no model is perfect.

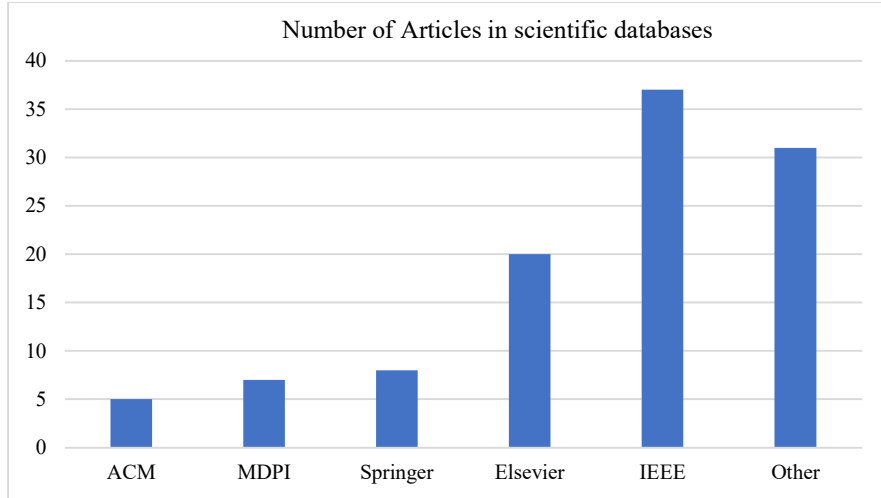


Fig. 9 Articles in scientific databases

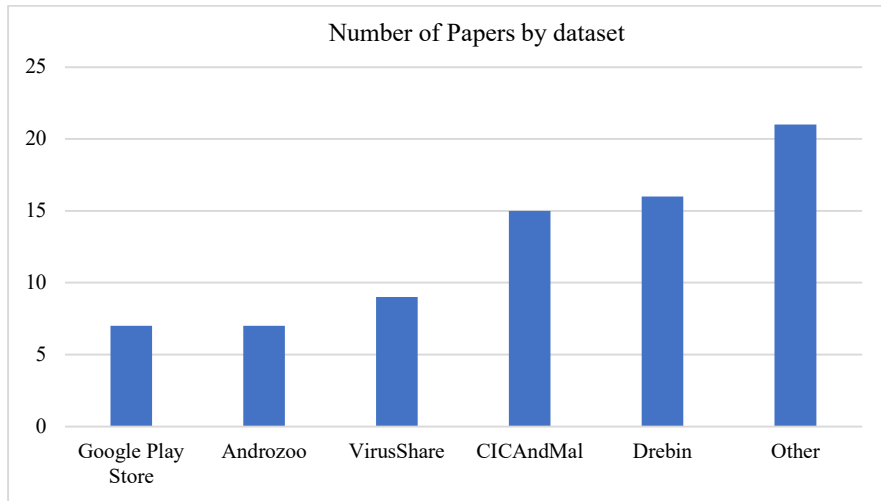


Fig. 10 Articles by datasets

The usage of hybrid models, which combine several approaches to improve, is a growing trend in the field of research. Researchers want to increase the resilience and longevity of malware detection systems by combining techniques like CNN and LSTM. Even with these developments, issues including the potential for hostile attacks, unbalanced datasets, and scalability issues persist. These problems must be fixed in order to advance the state-of-the-art in Android malware detection and strengthen the security posture of the Android ecosystems against emerging cyberthreats.

**5.2. Discussion**

It provides an excellent overview of current advancements in Android malware detection, particularly deep learning methods, and is improved by the information included in Table 3. Because of its broad use and vulnerability to security threats, some research has been done to improve the Android operating system's detection of malware infection. Important trends and developments in this important

topic are identified by merging the findings from the table data, which covers a range of studies from 2012 to 2024. LSTM networks and other deep learning architectures based on static analysis are used in malware detection. R. Vinayakumar et al. and X. Xiao et al. employ LSTM-based models to distinguish between dangerous and benign applications. They achieved remarkable detection rates by merging static and dynamic data. Android security systems could be enhanced by deep learning. Furthermore, dynamic analysis enhances the ability to detect malware.

Z. Fu et al. employed LSTM and transfer learning to increase the accuracy of malware detection systems. These provide malware detection and lower risks in an Android environment by training models on both static and dynamic data. R. B. Arslan and S. et al. Hybrid analyses that combine DL with conventional analytical techniques have been carried out by Menaouer et al. Because they mix features and use cutting-edge architectures, these hybrid models are more successful in identifying Android malware. On Android

smartphones, this aids in the fight against malware. They demonstrate the effectiveness of different strategies against newly evolving malware. Combining deep learning with other techniques has enhanced Android virus detection. According to research by D. Ć eponis et al. and E. C. Bayazit et al., models like CNNs and LSTMs are effective at identifying malware. They look at network traffic and different inputs to keep Android devices safe. Overall, recent talks about Android malware detection show researchers are doing a good job at making Android devices more secure. They are using deep learning and new tools to stay ahead of evolving threats.

### 5.3. Commonly Used Evaluation Metrics for Android Malware Detection

Android malware detection approaches use DL and LSTM in classification. The confusion matrices are shown below for each strategy. Models are capable of discriminating between benign and malicious Android applications. Results can be divided into four types as shown below. A true positive where the model correctly predicts the positive class. Similarly, a true negative is when the model correctly predicts a negative class. A false positive is when the model incorrectly predicts the positive class. And a false negative, where the model incorrectly predicts the negative class.

#### 5.3.1. Accuracy

Accuracy is defined as the ratio of correct predictions to the total number of instances in the test set. It is given by the formula

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

where TP represents the true positives, TN represents the true negatives, FP represents the false positives, and FN represents the false negatives.

#### 5.3.2. Precision

Precision is defined as the ratio of correctly classified malware instances to all instances predicted as malware. It is given by the formula:

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP}) \quad (2)$$

where TP represents the true positives, and FP represents the false positives.

#### 5.3.3. Recall

Recall is defined as the ratio of actual malware instances that are correctly classified. It is given by the formula.

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

where TP represents the true positives, and FN represents the false negatives.

#### 5.3.4. F1-Score

The F1-score is calculated using the formula:

$$F1 = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall}) \quad (4)$$

Where Precision is defined as TP/(TP+FP) and Recall is defined as TP/(TP+FN).

### 5.4. Critical Analysis of Failed and Suboptimal Approaches

Most published work emphasizes positive results, yet several strands of research reveal important negative evidence where deep learning either underperforms classical machine learning or exhibits limited practical value under realistic deployment constraints.

#### 5.4.1. Approaches to Underperforming Traditional Machine Learning

A first class of suboptimal designs involves deep models applied to low-quality or poorly engineered feature sets, where the additional capacity of deep networks does not translate into better generalization. In early static-only pipelines using small subsets of manifest permissions or coarse opcode counts, shallow MLPs, and even SVM/Random Forest baselines achieved 90–95% accuracy, while deeper CNN/RNN architectures trained on the same limited representations saturated at similar or slightly lower accuracy, offering no statistically significant improvement despite higher complexity.

Some transfer-learning studies also report disappointing performance when the source and target APK domains differ substantially in time, distribution, or malware families. For instance, models pre-trained on older, PC-oriented malware or on early Android datasets (MalGenome-era) and then fine-tuned on recent Android ransomware or banking Trojans achieve only 75–80% accuracy because the learned feature hierarchy does not match contemporary obfuscation, packing, and command-and-control behaviors, effectively underperforming carefully tuned traditional ML applied directly on the target dataset.

#### 5.4.2. Architectures Requiring Unrealistic Computational Resources

A second group of approaches demonstrates high accuracy but at computational costs incompatible with mobile or large-scale store deployment. Deep residual networks or very deep LSTM stacks (for example, 8–12 recurrent layers with hundreds of units) trained on full opcode sequences of thousands of instructions can require gigabytes of GPU memory and several hours per epoch, even on modern accelerators, making continuous retraining and on-device inference impractical for vendors operating under strict latency and energy constraints. Similarly, graph-neural-network models that construct fine-grained API or system-call graphs for each APK achieve around 95% accuracy but incur 150–400 ms of preprocessing and

message-passing time per sample, which becomes prohibitive when scanning hundreds of thousands of daily submissions in app-store settings. Hybrid pipelines that fuse multiple heavy branches—such as bytecode CNNs, API-sequence LSTMs, and graph modules in a single ensemble—often report near-perfect accuracy (>99.9%) on moderate-sized benchmarks but assume access to server-class GPUs and disregard the cost of feature extraction, graph construction, and repeated evaluation at scale, limiting their immediate applicability in resource-constrained or edge environments.

#### 5.4.3. Reproducibility and Evaluation Issues

The absence of complete reproducibility is a recurrent problem in a number of studies, undermining the validity of stated benefits. Common issues include using private or non-disclosed datasets that cannot be independently evaluated, omitting random seeds and hyperparameter configurations, using insufficient descriptions of dataset splits (no clear division between training and testing families), and not having code available. By permitting nearly identical variants of the same malware family to appear in both training and test splits, cross-validation is sometimes carried out without taking precautions against family-level leakage between folds, increasing stated accuracies.

Furthermore, a subset of experiments compares complex deep architectures against weak baselines (like untuned kNN or naive Bayes) rather than strong classical models (like gradient boosting or optimized Random Forest), making it difficult to attribute performance gains to architectural improvements rather than differences in experimental rigor. These repeatability gaps underscore the need for open implementations, family-aware evaluation processes, and consistent benchmarks.

#### 5.4.4. Failure Modes on Specific Malware Families

Deep learning systems also exhibit systematic blind spots on particular malware categories and evasion strategies. Static CNN or MLP models trained primarily on manifest and opcode features tend to misclassify heavily obfuscated samples, packed APKs whose malicious payload is decrypted at runtime, and malware that relies on dynamic code loading via reflection or native libraries; in such cases, accuracy can drop from  $\approx 99\%$  on benign or lightly obfuscated samples to around 60–70% on hardened variants when no dynamic features are incorporated.

Dynamic-only detectors, including temporal Transformer networks over API sequences, can fail to activate on logic-bomb or trigger-based malware that requires specific user interactions, geolocation, or time conditions; standard sandbox runs of a few minutes may never reach the malicious execution paths, resulting in false negatives despite sophisticated sequence modeling. Certain families of banking Trojans and ransomware that carefully mimic benign application behavior until a late-stage trigger are particularly

problematic, with reported recall reductions of 10–20 percentage points compared to overall average performance in some studies. Together, these unfavorable and subpar outcomes show that deep learning is not a panacea because its advantages heavily rely on feature quality, reasonable resource assumptions, meticulous evaluation design, and coverage of evasive behaviors across various malware families.

## 6. Emerging Research Frontiers

### 6.1. Quantum Machine Learning for Malware Detection

Exponential speedups for high-dimensional feature space analysis could be made possible by quantum computing capabilities. Quantum algorithms may be able to rank features with a quadratic speedup, according to early theoretical studies. The immaturity of quantum hardware and small-scale deployments are current limitations. Timeline: It is projected that real quantum virus detection will occur between 2027 and 2030.

### 6.2. Neuromorphic Computing

Efficient processing is made possible by neuromorphic processors, such as IBM TrueNorth and Intel Loihi, which imitate the structure of the brain. Potential: 100x increase in energy efficiency compared to GPUs. Applications: Real-time identification and deployment of edge devices. Present state: Limited datasets and early-stage research.

### 6.3. Causal Inference in Malware Detection

Advancing from correlation to causative connections between malware behavior and APK features. More reliable detection that is resistant to distribution shift is made possible by causal models. Current research from 2025: Causal forests with better generalization and 96.8% accuracy.

## 7. Critical Challenges and Future Work

### 7.1. Imbalance

Benign applications usually outweigh malicious samples in Android malware datasets, which show a notable class imbalance.

### 7.2. Model Interpretability

In crucial applications related to critical system security, the black-box operating pattern of deep learning models makes verification more difficult. In the latest developments, Sensitive Function Call Graph analysis and SHAP acquired an accuracy of 99.7% accuracy, alongside preserving explainability [109].

### 7.3. Adversarial Robustness

Malware authors create malwares by using various types of adversarial tactics that evade detection systems. To continue with strong malware detection efficiency, requires carrying on quality research with adversarial training methods with certified defenses.

#### 7.4. Real-time Detection

Lengthy, time-consuming malware detection analysis required by current solutions is not suitable for Android mobile devices. This calls for the need for a crucial research area of Edge computing to enable Android mobile devices for a speedy, real-time, dynamic analysis [26].

#### 7.5. Dataset Diversity and Temporal Evolution

Older Android malware datasets do not include advanced deceptive methods such as crypto-mining and polymorphic versions used by current evolving malwares. The AMD2025 and Mal-Droid2025 projects overcome such constraints by collecting longitudinal data.

#### 7.6. Ethical Considerations and Bias in Malware Detection

Although deep learning models achieve 99%+ accuracy in benchmarks, biases and ethical issues are still mostly unresolved. This section looks at issues with monitoring, fairness to developers, and data bias.

##### 7.6.1. Dataset Composition Bias

Historical datasets exhibit temporal biases. DREBIN (15,036 samples, 2012-2013) predominantly captures SMS Trojans while missing contemporary threats-87% of post-2022 Android malware uses supply-chain vectors absent from training data.

Models trained on biased datasets systematically underperform on emerging threats, causing false negatives (missed malware) and false positives (legitimate apps incorrectly flagged).

##### 7.6.2. Fairness Issues

Static permission-based detection penalizes privacy-conscious apps. Signal or Telegram requesting microphone + contacts legitimately may achieve 92-95% "suspicion scores," while remaining benign. This burden falls disproportionately on vulnerable populations (journalists, activists) relying on such apps. Critically, no reviewed study reports false positive distribution across app categories or developer demographics.

##### 7.6.3. Surveillance and Privacy

Dynamic analysis captures system calls, network traffic, and memory access patterns. When deployed at app store scale, legitimate user behavior (keystroke timing, payment patterns) may be inferred without consent. Federated learning + differential privacy achieves 97.8% accuracy while preserving privacy, yet most systems still centralize behavioral data.

##### 7.6.4. Recommendation

Future work should implement fairness testing for legitimate apps, transparent appeals mechanisms for flagged developers, and privacy-preserving detection frameworks before deployment.

## 8. Conclusion

With a focus on new methods in 2025, this systematic analysis summarizes developments in deep learning-based Android malware detection from 2012 to 2025. While traditional static and dynamic analysis remain foundational, hybrid approaches integrating multiple modalities achieve near-optimal detection accuracy (99%). Recent innovations in GRU, federated learning, transformer networks, and graph neural networks address privacy, interpretability, and relational modeling challenges.

In this study, a large number of reputed research publications have been filtered, sorted out, and analyzed to shed light on recent advancements in the field of malware detection for Android operating systems. The study also summarizes the complexities, accomplishments, and limitations of recent research works in combating Android malware. Few studies have been reviewed for their limitations with real-world performance. Different current and outdated Android malware datasets have also been analyzed against the performance of deep learning approaches.

However, persistent obstacles remain: (i) adversarial robustness against sophisticated evasion techniques, (ii) real-time detection compatible with mobile deployment, (iii) model interpretability for security certification, and (iv) dataset diversity capturing emerging threat variants. The following areas should be the focus of future research: (a) combining different modalities with formal robustness guarantees; (b) privacy-preserving detection frameworks that comply with organizational security policies; (c) explainable detection mechanisms that enable human-in-the-loop verification; and (d) curating longitudinal datasets that capture existing and new Android malware. As the Android ecosystem expands, proactive threat intelligence integration, cooperative dataset construction, and continuous research innovation are required to sustain efficient malware detection. The thorough and in-depth analysis in this review study will assist researchers in choosing future strategies for malware detection research.

## Data Availability

This study includes only publicly available datasets.

## Author Contributions

Mandeep Kumar conceptualized the study, conducted the literature survey, analyzed existing Android malware detection approaches and benchmark datasets, and drafted the manuscript. Dr. Abhishek Kajal, corresponding author, contributed substantially to the overall supervision of the design of the review study, refined the research framework, critically reviewed and improved the manuscript quality, and approved the final draft.

## References

- [1] Google, Security Reports Android Open-Source Project, 2024. [Online]. Available: <https://source.android.com/docs/security/overview/reports>
- [2] Avatlas, AV-ATLAS - Malware and PUA, 2026. [Online]. Available: <https://portal.av-atlas.org/malware/statistics>
- [3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [4] Jaswinder Singh, and Rajdeep Banerjee, "A Study on Single and Multi-Layer Perceptron Neural Network," *2019 3<sup>rd</sup> International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [5] Matt W. Gardner, and Stephen R. Dorling, "Artificial Neural Networks (The Multilayer Perceptron)-A Review of Applications in the Atmospheric Sciences," *Atmospheric Environment*, vol. 32, no. 14-15, pp. 2627-2636, 1998. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [6] Huijuan Zhu et al., "SEDMDroid: An Enhanced Stacking Ensemble Framework for Android Malware Detection," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 984-994, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [7] Neha Sharma, Vibhor Jain, and Anju Mishra, "An Analysis of Convolutional Neural Networks for Image Classification," *Procedia Computer Science*, vol. 132, pp. 377-384, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [8] Peng Song, Chaoyang Geng, and Zhijie Li, "Research on Text Classification based on Convolutional Neural Network," *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, Xi'an, China, pp. 229-232, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [9] Danish Vasan et al., "Image-based Malware Classification using Ensemble of CNN Architectures (IMCEC)," *Computers & Security*, vol. 92, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [10] Andrea Alamia et al., "Comparing Feedforward and Recurrent Neural Network Architectures with Human Behavior in Artificial Grammar Learning," *Scientific Reports*, vol. 10, no. 1, pp. 1-15, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [11] Julianna M. Czum, "Dive into Deep Learning," *Journal of the American College of Radiology*, vol. 17, no. 5, pp. 637-638, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [12] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins, "Learning to Forget: Continual Prediction with LSTM," *9<sup>th</sup> International Conference on Artificial Neural Networks: ICANN '99*, vol. 1999, pp. 850-855, 1999. [[CrossRef](#)] [[Google Scholar](#)]
- [13] Kyunghyun Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, pp. 1724-1734, 2014. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [14] Kaiming He et al., "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 2016. [[Google Scholar](#)] [[Publisher Link](#)]
- [15] Alyazia Aldhaheri et al., "Deep Learning for Cyber Threat Detection in IoT Networks: A Review," *Internet of Things and Cyber-Physical Systems*, vol. 4, pp. 110-128, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [16] Zhen Liu et al., "Research on Unsupervised Feature Learning for Android Malware Detection based on Restricted Boltzmann Machines," *Future Generation Computer Systems*, vol. 120, pp. 91-108, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [17] Zhenlong Yuan, Yongqiang Lu, and Yibo Xue, "Droiddetector: Android Malware Characterization and Detection using Deep Learning," *Tsinghua Science and Technology*, vol. 21, no. 1, p. 114-123, 2016. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [18] Han Qiu et al., "Adversarial Attacks against Network Intrusion Detection in IoT Systems," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10327-10335, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [19] Jing-Hao Xue, and D. Michael Titterton, "Comment on Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes," *Neural Processing Letters*, vol. 28, no. 3, p. 169-187, 2008. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [20] Yan Lu, and Jiang Li, "Generative Adversarial Network for Improving Deep Learning-based Malware Classification," *2019 Winter Simulation Conference (WSC)*, National Harbor, MD, USA, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [21] Antonia Creswell et al., "Generative Adversarial Networks: An Overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53-65, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [22] Jie Zhou et al., "Graph Neural Networks: A Review of Methods and Applications," *AI Open*, vol. 1, pp. 57-81, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [23] Chuanqi Tan et al., "A Survey on Deep Transfer Learning," *Lecture Notes in Computer Science*, pp. 270-279, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [24] Mohit Sewak, Sanjay K. Sahay, and Hemant Rathore, "Deep Reinforcement Learning for Cybersecurity Threat Detection and Protection: A Review," *Communications in Computer and Information Science*, pp. 51-72, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]

- [25] Liu Binxiang, Zhao Gang, and Sun Ruoying, "A Deep Reinforcement Learning Malware Detection Method based on PE Feature Distribution," *2019 6<sup>th</sup> International Conference on Information Science and Control Engineering (ICISCE)*, Shanghai, China, pp. 23-27, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [26] Kigbu Shallom, and Chukwujekwu Damian Ikemefuna, "Enhancing Malware Detection using Federated Learning and Explainable AI for Privacy-Preserving Threat Intelligence," *World Journal of Advanced Research and Reviews*, vol. 27, no. 1, pp. 331-351, 2025. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [27] Kaijun Liu et al., "A Review of Android Malware Detection Approaches based on Machine Learning," *IEEE Access*, vol. 8, pp. 124579-124607, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [28] Yajin Zhou, and Xuxian Jiang, "Dissecting Android Malware: Characterization and Evolution," *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, pp. 95-109, 2012. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [29] Jinpei Yan, Yong Qi, and Qifan Rao, "LSTM-based Hierarchical Denoising Network for Android Malware Detection," *Security and Communication Networks*, vol. 2018, pp. 1-18, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [30] Yu Feng et al., "Apposcopy: Semantics-based Detection of Android Malware through Static Analysis," *Proceedings of the 22<sup>nd</sup> ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 576-587, 2014. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [31] Elliot Mbunge et al., "A Review of Deep Learning Models to Detect Malware in Android Applications," *Cyber Security and Applications*, vol. 1, pp. 1-9, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [32] Riyadh Mahmood, Nariman Mirzaei, and Sam Malek, "EvoDroid: Segmented Evolutionary Testing of Android Apps," *Proceedings of the 22<sup>nd</sup> ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 599-609, 2014. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [33] Recep Sinan Arslan, "FG-Droid: Grouping based Feature Size Reduction for Android Malware Detection," *PeerJ Computer Science*, vol. 8, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [34] Huizhong Sun et al., "Android Malware Detection based on Feature Selection and Weight Measurement," *Intelligent Automation and Soft Computing*, vol. 33, no. 1, pp. 585-600, 2022. [[Google Scholar](#)] [[Publisher Link](#)]
- [35] Ashwag Albakri et al., "Metaheuristics with Deep Learning Model for Cybersecurity and Android Malware Detection and Classification," *Applied Sciences*, vol. 13, no. 4, pp. 1-18, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [36] Yinwei Wu et al., "DroidRL: Feature Selection for Android Malware Detection with Reinforcement Learning," *Computers & Security*, vol. 128, pp. 1-19, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [37] Tanjie Wang et al., "Android Malware Detection via Efficient Application Programming Interface Call Sequences Extraction and Machine Learning Classifiers," *IET Software*, vol. 17, no. 4, pp. 348-361, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [38] Abhishek Anand et al., "Android Malware Detection using LSTM with Smali Codes," *Preprint*, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [39] Rahul Gupta, Kapil Sharma, and Ramesh Kumar Garg, "Innovative Approach to Android Malware Detection: Prioritizing Critical Features using Rough Set Theory," *Electronics*, vol. 13, no. 3, pp. 1-26, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [40] Weiping Wang et al., "FGL Droid: An Efficient Android Malware Detection Method based on Hybrid Analysis," *Security and Communication Networks*, vol. 2022, no. 1, pp. 1-11, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [41] Muhammad Shoab Akhtar, and Tao Feng, "Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time," *Symmetry*, vol. 14, no. 11, pp. 1-12, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [42] Mohammed Maray et al., "Intelligent Pattern Recognition using Equilibrium Optimizer with Deep Learning Model for Android Malware Detection," *IEEE Access*, vol. 12, pp. 24516-24524, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [43] R. Vinayakumar et al., "Detecting Android Malware using Long Short-Term Memory (LSTM)," *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology*, vol. 34, no. 3, pp. 1277-1288, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [44] Xi Xiao et al., "Android Malware Detection based on System Call Sequences and LSTM," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 3979-3999, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [45] Iram Bibi et al., "An Effective Android Ransomware Detection through Multi-Factor Feature Filtration and Recurrent Neural Network," *2019 UK/ China Emerging Technologies (UCET)*, Glasgow, UK, pp. 1-4, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [46] Burak Tahtaci, and Beyzanur Canbay, "Android Malware Detection using Machine Learning," *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, Istanbul, Turkey, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [47] Tianliang Lu et al., "Android Malware Detection based on a Hybrid Deep Learning Model," *Security and Communication Networks*, vol. 2020, pp. 1-11, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]

- [48] Ruitao Feng et al., “Seqmobile: An Efficient Sequence-based Malware Detection System using RNN on Mobile Devices,” *2020 25<sup>th</sup> International Conference on Engineering of Complex Computer Systems (ICECCS)*, Singapore, pp. 63-72, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [49] Zhangjie Fu, Yongjie Ding, and Musaazi Godfrey, “An LSTM-based Malware Detection using Transfer Learning,” *Journal of Cyber Security*, vol. 3, no. 1, pp. 11-28, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [50] Yafei Wu et al., “DeepCatra: Learning Flow-and Graph-based Behaviours for Android Malware Detection,” *IET Information Security*, vol. 17, no. 1, p. 118-130, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [51] Brahami Menaouer et al., “Android Malware Detection Approach using Stacked AutoEncoder and Convolutional Neural Networks,” *International Journal of Intelligent Information Technologies*, vol. 19, no. 1, pp. 1-22, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [52] Ahmed R. Nasser, Ahmed M. Hasan, and Amjad J. Humaidi, “DI-AMDet: Deep Learning-based Malware Detector for Android,” *Intelligent Systems with Applications*, vol. 21, pp. 1-10, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [53] S. Poornima, and R. Mahalakshmi, “Automated Malware Detection using Machine Learning and Deep Learning Approaches for Android Applications,” *Measurement: Sensors*, vol. 32, pp. 1-8, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [54] Sudan Jha et al., “Recurrent Neural Network for Detecting Malware,” *Computers & Security*, vol. 99, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [55] Huanyu Wu, “A Systematical Study for Deep Learning-based Android Malware Detection,” *Proceedings of the 2020 9<sup>th</sup> International Conference on Software and Computer Applications*, pp. 177-182, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [56] Zhiqiang Wang, Qian Liu, and Yaping Chi, “Review of Android Malware Detection based on Deep Learning,” *IEEE Access*, vol. 8, pp. 181102-181126, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [57] A. Lakshmanarao, and M. Shashi, “Android Malware Detection with Deep Learning using RNN from Opcode Sequences,” *International Journal of Interactive Mobile Technologies (IJIM)*, vol. 16, no. 1, pp. 145-157, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [58] Sancheng Peng et al., “Challenges and Trends of Android Malware Detection in the Era of Deep Learning,” *2020 IEEE 8<sup>th</sup> International Conference on Smart City and Informatization (ISCI)*, Guangzhou, China, pp. 37-43, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [59] Esra Calik Bayazit, Ozgur Koray Sahingoz, and Buket Dogan, “Malware Detection in Android Systems with Traditional Machine Learning Models: A Survey,” *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, Ankara, Turkey, pp. 1-8, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [60] Dainius Čeponis, and Nikolaj Goranin, “Investigation of Dual-Flow Deep Learning Models LSTM-FCN and GRU-FCN Efficiency against Single-Flow CNN Models for the Host-based Intrusion and Malware Detection Task on Univariate Times Series Data,” *Applied Sciences*, vol. 10, no. 7, pp. 1-26, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [61] Mahshid Gohari, Sattar Hashemi, and Lida Abdi, “Android Malware Detection and Classification based on Network Traffic using Deep Learning,” *2021 7<sup>th</sup> International Conference on Web Research (ICWR)*, Tehran, Iran, pp. 71-77, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [62] Esraa Saleh Alomari et al., “Malware Detection using Deep Learning and Correlation-based Feature Selection,” *Symmetry*, vol. 15, no. 1, pp. 1-21, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [63] Muhammad Aamir et al., “AMDDLmodel: Android Smart- Phones Malware Detection using Deep Learning Model,” *PLoS One*, vol. 19, no. 1, pp. 1-16, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [64] Peter Zegzhda et al., “Detecting Android Application Malicious Behaviors based on the Analysis of Control Flows and Data Flows,” *Proceedings of the 10<sup>th</sup> International Conference on Security of Information and Networks*, pp. 280-286, 2017. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [65] Linfeng Wei et al., “Machine Learning-based Malicious Application Detection of Android,” *IEEE Access*, vol. 5, pp. 25591-25601, 2017. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [66] Luca Massarelli et al., “AndroDFA: Android Malware Classification based on Resource Consumption,” *Information*, vol. 11, no. 6, pp. 1-20, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [67] Yang Zhao, Guangquan Xu, and Yao Zhang, “HFA-MD: An Efficient Hybrid Features Analysis based Android Malware Detection Method,” *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 248-257, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [68] Suman R. Tiwari, and Ravi U. Shukla, “An Android Malware Detection Technique using Optimized Permission and API with PCA,” *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, pp. 2611-2616, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [69] Shweta Bhandari et al., “SWORD: Semantic Aware AndrOid Malware Detector,” *Journal of Information Security and Applications*, vol. 42, pp. 46-56, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]

- [70] Saba Arshad et al., "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, pp. 4321-4339, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [71] Ke Xu et al., "DeepRefiner: Multi-Layer Android Malware Detection System Applying Deep Neural Networks," *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, London, UK, pp. 473-487, 2018. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [72] Haoyu Wang et al., "RmvDroid: Towards a Reliable Android Malware Dataset with App Metadata," *2019 IEEE/ACM 16<sup>th</sup> International Conference on Mining Software Repositories (MSR)*, Montreal, QC, Canada, pp. 404-408, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [73] Yuxuan Yan et al., "Understanding and Detecting Overlay-based Android Malware at Market Scales," *Proceedings of the 17<sup>th</sup> Annual International Conference on Mobile Systems, Applications, and Services*, pp. 168-179, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [74] William Younghoo Lee, Joshua Saxe, and Richard Harang, *SeqDroid: Obfuscated Android Malware Detection using Stacked Convolutional and Recurrent Neural Networks*, Advanced Sciences and Technologies for Security Applications, pp. 197-210, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [75] Shanshan Wang et al., "Deep and Broad URL Feature Mining for Android Malware Detection," *Information Sciences*, vol. 513, pp. 600-613, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [76] Jiayin Feng et al., "A Two-Layer Deep Learning Method for Android Malware Detection using Network Traffic," *IEEE Access*, vol. 8, pp. 125786-125796, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [77] Shahid Alam, Soltan Abed Alharbi, and Serdar Yildirim, "Mining Nested Flow of Dominant APIs for Detecting Android Malware," *Computer Networks*, vol. 167, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [78] Mohammad Alhanahnah et al., "DINA: Detecting Hidden Android Inter-App Communication in Dynamic Loaded Code," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2782-2797, 2020. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [79] Eslam Amer, Ivan Zelinka, and Shaker El-Sappagh, "A Multi- Perspective Malware Detection Approach through Behavioral Fusion of API Call Sequence," *Computers & Security*, vol. 110, pp. 1-21, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [80] Huanran Wang, Weizhe Zhang, and Hui He, "you are What the Permissions Told Me! Android Malware Detection based on Hybrid Tactics," *Journal of Information Security and Applications*, vol. 66, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [81] Junwei Tang et al., "Android Malware Obfuscation Variants Detection Method based on Multi-Granularity Opcode Features," *Future Generation Computer Systems*, vol. 129, pp. 141-151, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [82] Pooja Yadav et al., "EfficientNet Convolutional Neural Networks-based Android Malware Detection," *Computers & Security*, vol. 115, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [83] Hemant Rathore et al., "Breaking the Anti-Malware: EvoAAAttack based on Genetic Algorithm against Android Malware Detection Systems," *International Conference on Computational Science*, pp. 535-550, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [84] Diego Soi et al., "Can You See Me? On the Visibility of NOPs against Android Malware Detectors," *arxiv preprint*, pp. 1-21, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [85] S. Arshad Hashmi, "Malware Detection and Classification on Different Dataset by Hybridization of CNN and Machine Learning," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 6s, pp. 650-667, 2023. [[Google Scholar](#)] [[Publisher Link](#)]
- [86] Sushil Buriya, and Neelam Sharma, "Malware Detection using 1d Convolution with Batch Normalization and L2 Regularization for Android," *2023 International Conference on System, Computation, Automation and Networking (ICSCAN)*, Puducherry, India, pp. 1-6, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [87] Atif Raza Zaidi et al., "Effectiveness of Detecting Android Malware using Deep Learning Techniques," *Journal of Nanoscope (JN)*, vol. 4, no. 2, pp. 1-21, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [88] Abdurraheem Joomye, Mee Hong Ling, and Kok-Lim Alvin Yau, "Dynamic Android Malware Detection using Temporal Convolutional Networks," *2023 IEEE International Conference on Computing (ICOCO)*, Langkawi, Malaysia, pp. 317-322, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [89] Doan Minh Trung et al., "On the Effectiveness of Transferability of Adversarial Android Malware Samples against Learning-based Detectors," *2023 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, Quy Nhon, Vietnam, pp. 1-6, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [90] Jabunnesa Jahan Sara, and Shohrab Hossain, "Static Analysis based Malware Detection for Zero-Day Attacks in Android Applications," *2023 International Conference on Information and Communication Technology for Sustainable Development (ICT4SD)*, Dhaka, Bangladesh, pp. 169-173, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [91] Hayam Alamro et al., "Automated Android Malware Detection using Optimal Ensemble Learning Approach for Cybersecurity," *IEEE Access*, vol. 11, pp. 72509-72517, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [92] Chenhao Zhang et al., "Detecting Android Malware with Pre-Existing Image Classification Neural Networks," *IEEE Signal Processing Letters*, vol. 30, pp. 858-862, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]

- [93] Marwa Mamdouh, Khaled Elsayed, and Ahmed Elsheikh, "Android Malware Detection via Deep Learning Approach," *2023 Intelligent Methods, Systems, and Applications (IMSA)*, Giza, Egypt, pp. 251-256, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [94] Shresth Jain et al., "Android Malware Analysis using Coefficient of Multiple Correlation," *2023 IEEE Symposium on Wireless Technology & Applications (ISWTA)*, Kuala Lumpur, Malaysia, pp. 72-77, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [95] Shi Dong, Longhui Shu, and Shan Nie, "Android Malware Detection Method based on CNN and DNN Bybrid Mechanism," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 5, pp. 7744-7753, 2024. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [96] Esra Calik Bayazit, Ozgur Koray Sahingoz, and Buket Dogan, "A Deep Learning based Android Malware Detection System with Static Analysis," *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, Ankara, Turkey, pp. 1-6, 2022. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [97] Fahdah A. Almarshad et al., "Detection of Android Malware using Machine Learning and Siamese Shot Learning Technique for Security," *IEEE Access*, vol. 11, pp. 127697-127714, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [98] K.V. Vinayaka, and C.D. Jaidhar, "Heterogeneous Graph Convolutional Networks for Android Malware Detection using Callback-Aware Caller-Callee Graphs," *Computing and Processing*, 2021. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [99] Riddhi Gupta et al., "Malware Analysis using Machine Learning and Deep Learning," *2023 3<sup>rd</sup> International Conference on Technological Advancements in Computational Sciences (ICTACS)*, Tashkent, Uzbekistan, pp. 341-346, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [100] Md. Habibullah Shakib, "Android Malware Detection Approach's based on Genetic AI, CNN, RNN, ISTM, GRU, and Active Learning," *SSRN*, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [101] Abhishek Sujith et al., "Enhancing Android Security: Static Analysis for Robust Protection and Resilient Defences using Deep Learning," *2023 7<sup>th</sup> International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Kirtipur, Nepal, pp. 154-159, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [102] Ghadah Aldehim et al., "Gauss-Mapping Black Widow Optimization with Deep Extreme Learning Machine for Android Malware Classification Model," *IEEE Access*, vol. 11, pp. 87062-87070, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [103] Wenbo Fang et al., "Comprehensive Android Malware Detection based on Federated Learning Architecture," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3977-3990, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [104] A. Jyothish, Ashik Mathew, and P. Vinod, "Effectiveness of Machine Learning based Android Malware Detectors against Adversarial Attacks," *Cluster Computing*, vol. 27, no. 3, pp. 2549-2569, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [105] Zakaria Sawadogo et al., "Zero-Vuln: using Deep Learning and Zero-Shot Learning Techniques to Detect Zero-Day Android Malware," *2023 3<sup>rd</sup> International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, Tenerife, Canary Islands, Spain, pp. 1-5, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [106] Santosh K. Smmarwar, Govind P. Gupta, and Sanjay Kumar, "XAI-AMD-DL: An Explainable AI Approach for Android Malware Detection System using Deep Learning," *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)*, Sonbhadra, India, pp. 423-428, 2023. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [107] Alejandro Martín, Raúl Lara-Cabrera, and David Camacho, "Android Malware Detection through Hybrid Features Fusion and Ensemble Classifiers: The AndroPYtool Framework and the Omnidroid Dataset," *Information Fusion*, vol. 52, pp. 128-142, 2019. [[CrossRef](#)] [[Google Scholar](#)] [[Publisher Link](#)]
- [108] VirusShare.com, [Online]. Available: <https://virusshare.com/>
- [109] N Anitha Devi et al., "Shap based -Android Malware Detection using Ensemble Learning," *International Research Journal on Advanced Science Hub*, vol. 7, no. 07, pp. 673-680, 2025. [[CrossRef](#)] [[Publisher Link](#)]
- [110] C. Sruneethi, and Kotla Vandana, "Automated Android Malware Detection using Optimal Ensemble Learning Approach for Cybersecurity," *International Journal of Scientific Research in Science and Technology*, vol. 12, no. 3, pp. 331-334, 2025. [[CrossRef](#)] [[Publisher Link](#)]