**Journal of Neonatal Surgery**
*ISSN(Online): 2226-0439*
*Vol. 14, Issue 32s (2025)*
https://www.jneonatalsurg.com

# Android Malware Detection Using Deep Learning Approach

**Chappati Jahnavi[1], Dr. S. Srinivasa Rao[2]**

[1] PG Student, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeshwaram, Andhra Pradesh, Guntur, 522502, India
Email ID : chappatijahnavi01@gmail.com
[2] Associate Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeshwaram, Andhra Pradesh, Guntur, 522502, India
Email ID: srinu1479cse@kluniversity.in

Cite this paper as: Chappati Jahnavi, Dr. S. Srinivasa Rao (2025) Android Malware Detection Using Deep Learning Approach. *Journal of Neonatal Surgery*, 14 (32s), 6048-6064.

## ABSTRACT

This paper presents a novel framework for detecting malware in Android applications using advanced machine learning techniques. Our approach combines static and dynamic analysis with deep learning algorithms to identify malicious patterns in Android applications. We propose a hybrid feature extraction method that captures both code- based and behavioral attributes, followed by a multi-layer classification model that achieves high detection accuracy. Experiments conducted on a comprehensive dataset of benign and malicious applications demonstrate the effectiveness of our approach, achieving 97.8% accuracy, 96.5% precision, and 98.2% recall. The proposed framework outperforms traditional signature-based methods and several existing machine learning approaches, showing promise for real-time malware detection in resource-constrained mobile environments

**Keywords**: Android security, malware detection, machine learning, deep learning, static analysis, dynamic analysis, feature extraction, mobile security, classification algorithms

## 1. INTRODUCTION

Mobile devices and tablets using the Android operating system have been more susceptible to attacks that are carried out by malicious software in recent years. This is as a consequence of the fact that Android smartphones are gaining more and more popularity, which has led to an increase in the number of individuals who are using them. The malicious program that is responsible for carrying out these assaults is the one that is responsible for carrying them out. One of the key reasons for this is that, over the course of the last few years, there has been an increase in the number of people using smartphones that carry the Android operating system. This is one of the primary reasons why this is the case. If a user has access to the Google Play Store, they will have the opportunity to download millions of different applications. Consideration should also be given to this extra aspect of the situation. In addition to this, there are presently more than three billion instances of Android smartphones being used in different parts of the world. The reason for this is because Android is an operating system that is used by a number of people. As a consequence of this, it has become more challenging to guarantee that the use of these technologies does not include any risks related with their utilization. The issue has gotten more difficult, which has therefore resulted in the problem being more complicated. This problem has become more complicated as a consequence of both of these elements, which have led to the increasing complexity of the situation. The reason for this is that the circumstance has improved in terms of complexity. As a consequence of this, the problem has therefore become more challenging. By using traditional malware detection techniques that are based on signatures, it is not possible to uncover attacks that are both sophisticated and unexpected. This is because signatures are the foundation of these approaches. The techniques that have been discussed so far are insufficient. As a direct result of the fact that these threats were not anticipated in the past, it is completely impossible to identify them at this time. This is the reason why things are the way they are. This difficulty is caused by the fact that these approaches are based on known patterns that have been created from malware samples that have been collected in the past. This is the reason why this problem occurs. The following is a possible explanation for the reason why this particular event really took place. Because of this, things happen in the manner in which they occur. This is the reason why things take place. The use of techniques that are founded on machine learning (ML) as potential options for the goal of lowering these constraints has, on the other hand, attracted an increasing amount of attention over the course of the last few years. Over the course of time, it has been abundantly clear that the importance of this attraction has increased to a larger degree due to the passage of time. As opposed to the conventional methods, these approaches make it feasible to identify novel viruses via the use of pattern recognition and behavioural analysis. This is in contrast to the conventional methods, which are used to identify viruses. This constitutes an alternative to the conventional methods that have been used hitherto. With the resources that we now have available to us, it is evident that this is not something that can be accomplished. The processes that are often used are not able to accomplish this goal since they are not capable of doing so for themselves. They have achieved a higher degree of success than they would have under any other set of circumstances. their particular -

feature, which adds to the total efficacy of their approaches, is the reason for their achievement. The use of algorithms that are based on the idea of machine learning makes it possible to differentiate between software that is acceptable for use and software that has the potential to do harm. This may be accomplished by using the notion of machine learning. This objective might be done in a number of ways, one of which is by determining the minute patterns that differentiate the two categories of software. Even in situations in which the virus employs techniques or procedures in order to avoid being identified, this is still

the case. In point of fact, this is the case. The following is a description of the actual events that transpired. By doing an analysis of big datasets that include computer programs that are both safe and hazardous, it is feasible to obtain these patterns. This may be accomplished via the process of undertaking an analysis. It would not be completely out of the question to put this strategy into practice in the real world.

For the purpose of providing a more accurate depiction of the circumstance, deep learning architectures are used inside the framework of this study project. The completion of this research endeavour will result in the development of a comprehensive framework for the detection of malware on Android devices that takes use of strong machine learning techniques. This framework will be the ultimate product of this research endeavour. The building of this framework as the product that incorporates all of its components will be the climax of this research activity that has been going on. Our approach incorporates both dynamic analysis, which includes evaluating the behaviour of the program while it is being executed, and static analysis, which requires analysing the code of an application without it being executed. Both types of analysis are included inside our technique. The methodologies that we use are capable of covering both sorts of analysis. Following the completion of the program's execution, it is customary to do static analysis on the program. An example of the sort of analysis that is referred to as static analysis is an analysis that is carried out without the application being really performed. This type of analysis is carried out. The kind of analysis that is used by the great majority of people is known as analysis of a static nature. Consequently, as a result of this, we are in a position to extract a complete collection of features with the intention of identifying applications that are harmful. Because of the circumstances that have come about, this is the outcome. For the goal of malware detection, the objective of this research is to provide a unique approach for feature selection that, while simultaneously lowering the amount of processing overhead, picks the characteristics that demonstrate the maximum degree of discriminating ability. This is the goal that this endeavour is aiming to achieve. This is the objective of the study that is being conducted. This technique has been adjusted in order to fulfil the specific needs that are associated with malware detection. This was done in order to meet the special limits are related with malware detection. One of the reasons why our technique is particularly well-suited for distribution on mobile devices, which often have a restricted amount of resources accessible to them, is because of this particular reason. This is a consequence of the fact that our technique is ideally suited for implementation, which is a consequence of this.

## 2. . LITERATURE REVIEW

The area of study has made great progress in the last 10 years in terms of recognizing harmful software programs that are loaded on Android devices. This progress has been made possible by the advent of the Android operating system. Because of the considerable advancements that have been achieved in this sector, accomplishments like this one have been feasible. Regarding the identification of software programs that are intrinsically hazardous, this

new breakthrough has been made in the context of the identification of such applications. One of the most important things that I want to bring to your notice is the fact that a significant amount of progress has been made in this specific field. The utilization of a broad range of categories, which can be found on the internet, makes it easy to differentiate between the many ways that are now being used. The hybrid solutions, signature-based methods, static analysis techniques, and dynamic analysis approaches are all covered in these categories. Statistical analysis techniques are also included. Moreover, signature-based techniques are included into this suite of components. Methods that are in accordance with signatures are the ones that are included in this category of procedures. This is a list of the categories that we have decided to build in the years to come at this point in time. The list may be seen below. In addition, hybrid solutions are categorized as belonging to other categories in accordance with the categorization system. Apart from the fact that hybrid solutions are categorized, there is another point to consider. Two examples of apps that make use of signature-based detection features are the software utilities Andro guard [1] and APKInspector [2]. Signature- based detection is sometimes referred to as signature- based detection. Since both of these programs are instances of software programs, we may consider them to be examples of applications. It is now possible for you to download both of these programs from the website, which puts your convenience at the forefront. They may be compared to one another in a great number of different ways on a wide variety of different levels or dimensions. Both of these instruments are functionally comparable to one another. A comparison of the properties of a program to known malware signatures serves as the basis for this form of detection, which is based on the process of this comparison, which is based on the comparison. This comparison is based on the comparison. In order to construct this comparison, the comparison serves as the foundation. The comparison is the basis for this comparison, which acts as the foundation. The comparison itself serves as the foundation for this comparison, just as the comparison itself serves as the basis for that comparison. The foundation is this comparison, and the comparison itself is the basis for this comparison. Both responsibilities are performed by this comparison. Although these tactics are successful in general, they are not very effective when it comes to zero-day attacks or polymorphic malware. This is despite the fact that they are successful in general. They have achieved success in general, but this is in spite of the fact that they have achieved those successes. Although they have been successful in a number of specific areas, they have been successful overall. This is despite the fact that they have been successful in some areas. Despite the fact that this is the case, they continue to take pleasure in the benefits of their achievement since, in general, they are successful. To be more specific, this is the reason for their achievements, which is precisely what they accomplish. This is shown by Rastogi et al. [3], who demonstrated that it is not difficult to get around a considerable number of signature-based systems by making a few simple modifications to the code. To a certain extent, this is something that can be performed with little difficulty. This aim was attained by using an approach that circumvented the signatures in order to achieve the desired result. This stance was supported by the data that was given by these findings, which were acquired by the researchers. This position was supported by the data. Utilizing techniques that are related with static analysis is the means by which the evaluation of the application code is carried out. This is in contrast to the process of carrying out the algorithm, which is described in the previous sentence. When it comes to putting the algorithm into action, it is not always necessary to be able to put these techniques into effect. It's possible that some of the methods might be utilized instead. It is known as DREBIN, which is the term that they came up with jointly [4]. The method that Arp and his colleagues created together is known as DREBIN. They decided to go with the name DREBIN as their moniker. Together, they were responsible for its development throughout the whole of the manufacturing process, which was a collaborative effort between the two of them. Both of them shared equal responsibility for its creation. In order to finish the process of detecting malicious software, it is necessary to make use of Support Vector Machines (SVM) in order to extract features from the manifest file and API calls. This is necessary in order to accomplish the task. As a result, this guarantees that the procedure will be effectively finished. This is something that has to be done in order to guarantee that the operation will be carried out effectively. In order to bring the discovery process to a successful finish, it is required to fulfil this requirement in its entirety. The fulfilment of all of these possibilities is brought about by the use of support vector machines, which are more often referred to as SVM. This method is being employed with the aim of reaching the goal of finishing the assignment, which is the rationale for its usage. The goal is being achieved with the intention of accomplishing the goal. By using a technique that is often referred to as association rule mining, Chen et al. [5] concentrated their attention on the permission combinations that had the potential to cause damage. They were able to determine the permission combinations that had the potential to inflict harm as a result of this. They were able to focus their attention on the combinations that were accepted thanks to the use of this process, which was a big achievement. Because they concentrated their attention on each combination in isolation, they were able to successfully accomplish this goal. This was due to the fact that they focused their attention on. [6] Grace and her colleagues came up with the idea of developing a piece of software that they called Risk Ranker with the purpose of discovering potentially hazardous code patterns. Their goal was to identify code patterns that might possibly cause harm. The procedure that they went through had as

its goal the detection of problematic coding patterns. Determining faulty code patterns was the major purpose of the software that was being developed at the time. They were carried out with the objective of discovering potentially dangerous patterns of code usage, which was the reason why these measures were established in the first place. On the other hand, when it comes to static methods, the weakness resides in the fact that they are susceptible to techniques that disguise the structure of the code. It is because of this vulnerability that static methods are vulnerable. Due to the fact that static methods are not dynamic, this is the situation. This is the circumstance that has arisen as a result of the fact that static methods were not dynamic. This is because static methods do not belong to the same category as dynamic methods in terms of the degree of dynamic behaviour that they display. This is the reason why this is the case. This method is referred to as "dynamic analysis," and the phrase "dynamic analysis" is used to refer to the strategy that entails monitoring the behaviour of an application while it is going through its process. According to the field of computer programming, this technique is referred to as "dynamic analysis," and it is associated with the field of computer programming. Within the realm of analysis, this particular approach is referred to as "dynamic analysis." In order to discover instances of privacy breaches, Enck and his colleagues came up with the concept for TaintDroid [7], which was designed to monitor the flow of information in order to identify instances of privacy violations. Identifying individuals who had breached their own privacy was the goal of TaintDroid, which was designed to do this. Identifying instances of privacy infractions that have taken place was one of the objectives of the TaintDroid project. This objective would be accomplished by the use of the strategy of monitoring the flow of information as the technique that would be taken. Specifically, this would be the approach that would be put into action. The task of determining patterns of behaviour that are likely to be associated with malicious software or software falls within the purview of CrowDroid [8]. CrowDroid is responsible for making this judgment, which is a task that falls within its purview. To ensure that it is able to live up to the obligations that it has taken on, this is going to be carried out. In addition to this, CrowDroid is responsible for monitoring the calls that are made to the system when it is actively operating. This is an extra item of labour that is included in its definition of responsibilities. The monitoring that is being carried out with the purpose of discovering patterns of activity that have been taking place is being done with the intention of achieving the aim of discovering patterns of activity that have been taking place. The application that is known as DroidScope [9] enables users to get access to a virtualized environment that is made available to them. This access is made possible as a consequence of the program. The experience that consumers have with this has the potential to be highly useful to them. It is possible to study the behaviour of malicious software by exploiting this environment, which

can be used for the goal of evaluating the behaviour of this environment. This environment may be utilized for the application of this analysis. It is possible to put this environment to a number of different uses. One of the things that may be done is to carry out an investigation of the manner in which this environment displays itself. For the purpose of identifying complex malware, it is of the highest significance to handle the problem by using techniques that are dynamic rather than static. This is because dynamic methods are more likely to be successful. Both of these approaches are associated with a substantial degree of processing overhead, which is something that has to be taken into proper account. These two strategic approaches are related to one another in some kind or another. The identification of harmful software is one of the procedures that can be carried out with the support of a variety of different technologies. These technologies may be deployed to assist in the detectionof malicious software. One thing that is now common knowledge is that a significant amount of machine learning has been used in order to achieve the goal of recognizing malicious software that has been installed on Android devices. The information in question has been made public. The procedure of detecting malicious software, which led to the discovery of this information, was carried out. In order to detect malicious software, Narudin et al. [10] performed research on a variety of machine learning approaches in order to analyse network data. This study was carried out in order to identify harmful software. In order to complete the procedure, this step was carried out. This aim was successfully accomplished by the conduct of the examination of the network data. It was with the purpose of establishing which algorithms stood out as being especially successful that they carried out their study. By carrying out this activity, the objective of carrying out an analysis of the data that was obtained from the network was achieved, which was the objective of carrying out this activity in the first place. For the purpose of applying their findings to binary data that had been translated, Yuan et al. [11] made use of convolutional neural networks, which are more often referred to as CNNs. It was decided to take action in order to put their results into practice, and this particular step was carried out. The usage of approaches that are related with the area of deep learning was what we performed in order to attain this target with substantial success. In order to accomplish the goal of controlling API call sequences, the sequence models that were applied by MalDozer, which was created by Karbab et al. [12], were utilized. This was carried out in line with the purpose that was indicated before. This action was taken in order to achieve the intended effect, which was the reason for carrying it out in the first place. It was with the idea of achieving the objectives that were specified at the beginning of the process that the action in question was carried out. As part of an experiment that was carried out not too long ago, Wu et al. [13] looked at the possibility of using a graph convolutional network as a technique for

representing application programming interface (API) call graphs. An investigation effort similar to this one has been carried out throughout the course of the last several years. In a paper that was published in the journal Computer Research, the results of this inquiry were disclosed to the public. When it came to doing business, there were a significant number of people who were of the opinion that this was an efficient technique. The use of hybrid approaches has been created to include both static and dynamic analysis, and it has also been shown that these procedures have been effective in producing findings that are more than satisfactory. It has been proved that both of these statements are true. Within the context of this specific event, these two achievements have been shown. DroidFusion is a piece of software that was created by Yerima and Sezer [14], who are the ones who are responsible for its invention or development. The creation of this application was accomplished via the use of a technique known as ensemble learning. It was the fact that the feature sets could be mixed with one another that was the single most crucial component that made it possible to achieve this result. The fundamental objective of the MaMaDroid project, which was created by Mariconti and his colleagues [15], was to describe the behaviour of apps as Markov chains that were constructed in line with API call sequences. The attention of MaMaDroid was principally concentrated on this attempt. MaMaDroid placed a significant amount of emphasis on this particular aspect. MaMaDroid placed a significant amount of focus on this particular aspect of the product compared to other aspects. As a result of the fact that this was the most essential objective that needed to be fulfilled within the context of the project, it was essential to carry out this action. Deep learning was directly applied to opcode sequences that were retrieved from application binaries by McLaughlin et al. for their study [16]. This was done in collaboration with the researchers. For the purpose of evaluating performance, this was carried out. Both the researchers and the researchers themselves were engaged in the process they were doing. The performance of the person was going to be evaluated, and that was the purpose of carrying out this certain action. During the procedure that they were taking part in, the researchers, as well as the researchers themselves, were actively engaged in the process. Carrying out this specific action was done with the purpose of assessing the performance of the person, which was the rationale for carrying out this particular action in the first place. The researchers, in addition to the researchers themselves, were actively interested in the process that they were participating in all throughout the procedure that they were participating in. The performance of the individual was evaluated, which was the reason for carrying out this particular activity in the first place. The objective of carrying out this particular action was to evaluate the performance of the individual. For the same reason, adversarial machine learning is regarded as one of the most recent accomplishments in the field of machine learning. This opinion is based on the

explanation that was presented before. In the field of machine learning, it is also one of the most recent breakthroughs that has been made possible. It is for this reason that this is being done in order to guarantee that it has this power, so that it may have the effect of increasing resistance against circumstances in which efforts are made to avoid being discovered. The use of adversarial training as a method was presented by Chen et al. [17] as a way of attaining the goal of raising the levels of resilience that models display. This proposal was made in order to fulfil the objective of increasing the levels of resilience that models exhibit. It was decided to take this step in order to achieve the objective of increasing the resilience of mathematical models. The use of transfer learning methodologies was carried out with the purpose of effectively addressing the issue at hand. Li et al. came up with these tactics on their own initiative as a solution to the problem of having a limited number of malware samples that have been tagged. This was done in order to overcome the obstacle. To be more specific, the specific number in question is twenty-eight. Wang et al. [19] were the pioneers in the application of federated learning approaches, which made it feasible to train collaborative models while retaining secrecy. This was the first time that this had been done. The prospect of training collaborative models became a reality after that point in time. It was able to achieve this goal as a result of the use of these many ways. As a result of the processes being effective in accomplishing their intended objective, this was made possible as a consequence of the fact that they were successful. Due to the fact that the processes were successful in accomplishing their objectives, this was successfully accomplished.

Because it is difficult to create a strategy that achieves this balance, it is difficult to find a strategy that strikes a balance between the accuracy of detection, the efficiency of processing, and the durability against evasion techniques at the same time among the approaches that are currently available. This is because it is difficult to establish a strategy that achieves this balance. It is sufficient to say that the fact that they are forced to participate in this activity presents them with a difficulty. This is the situation that has persisted over the course of the years, despite the fact that there has been a significant amount of progress made that pertains to this topic. By putting in place a system that is completely distinct from anything else that has ever tried to do this in any way, shape, or form, we have been able to circumvent these limitations as a consequence of the efforts that we have already put forward. As a result of the restrictions imposed by this framework, the most effective approach for feature extraction is combined with an architecture that is effective for deep learning. The most efficient method for feature extraction is achieved by the combination of these two approaches. When it comes to the process of trying to install this framework on mobile devices, there are almost no obstacles that might possibly become an issue. This is the case since there are no obstacles.

## 3. METHODOLOGY

There are four fundamental components that make up our method to detecting malware on Android. These components are as follows: (1) the collection and preprocessing of data; (2) the extraction of features via static and dynamic analysis; (3) the selection and engineering of features; and (4) the building and training of deep learning models.

Data Collection and Preprocessing

We compiled a comprehensive dataset consisting of 10,000 Android applications, including 5,000 benign applications sourced from the Google Play Store and 5,000 malicious applications from various malware repositories, including the Android Malware Genome Project, VirusShare, and ContagioDump. Applications were collected between January 2020 and July 2024 to ensure relevance to current malware threats.

Each application underwent preprocessing steps:

APK decompilation using Apktool to extract manifest files, resource files, and DEX bytecode

Conversion of DEX to Java bytecode using dex2jar for static analysis

Manifest file parsing to extract permissions and components

Installation in a sandboxed environment for dynamic analysis

To ensure dataset quality, we verified malware labels using VirusTotal, considering an application malicious if detected by at least 5 out of 60 antivirus engines. The dataset was split into 70% for training, 15% for validation, and 15% for testing, maintaining the same class distribution across all sets

**Feature Extraction**

Our feature extraction process combines static analysis, dynamic analysis, and permission-based features to create a comprehensive representation of application behaviour.

Static Analysis Features:

API call patterns: Frequency and sequence of API calls from different Android framework categories

Code structure metrics: Cyclomatic complexity, method count, class inheritance depth

String features: Presence of suspicious URLs, IP addresses, commands

Intent filters and components declared in the manifest

Cryptographic operations and reflection usage

Native library usage and code size metrics

Dynamic Analysis Features:

System call patterns during execution

Network activity: Destinations, protocols, data volume

File system operations: Read/write patterns, accessed locations

Inter-process communication patterns

Memory usage patterns and CPU utilization

Permission usage during runtime

Permission-Based Features:

Requested permissions categorized by protection level

Permission usage patterns compared to application category norms

Permission combinations known to be associated with malwar

Feature Selection and Engineering

To identify the most discriminative features while reducing dimensionality, we employed a multi-stage feature selection process:

Removal of features with near-zero variance across the dataset

Correlation analysis to eliminate redundant features

Information gain analysis to rank features by discriminative power

Principal Component Analysis (PCA) for dimensionality reduction

Feature engineering techniques were applied to enhance the discriminative power:

N-gram representation of API call sequences

Frequency-inverse document frequency (TF- IDF) transformation for text features

Creation of composite features combining related indicators

Normalization of numerical features to improve model convergence

Deep Learning Architecture

We designed a hybrid deep learning architecture that combines convolutional neural networks (CNNs) for spatial feature extraction, recurrent neural networks (RNNs) for sequential pattern analysis, and attention mechanisms to focus on critical malware indicators. The architecture consists of:

Input layer accepting multi-modal feature vectors

Embedding layers for categorical features

1D CNN layers for local pattern detection

Bidirectional LSTM layers for sequential pattern analysis

Self-attention mechanism to identify critical features

Fully connected layers with dropout for classification

Output layer with sigmoid activation for binary classification

The model was trained using the Adam optimizer with learning rate scheduling and early stopping to prevent overfitting. Class weights were adjusted to address class imbalance in the training data.

Algorithms

This section presents the key algorithms and mathematical foundations of our approach.

Feature Extraction Algorithm

Our feature extraction process translates raw application components into numerical feature vectors suitable for machine learning. For static analysis, we extract API call frequencies using Algorithm 1.

Algorithm 1: API Call Frequency Extraction

Input: Android APK file A

Output: API call frequency vector F

```
1: $D \leftarrow$ Decompile(A)
2: $C \leftarrow$ ExtractClasses(D)
3: Initialize frequency vector F with zeros
4: for each class c in C do
5:     $M \leftarrow$ ExtractMethods(c)
6:     for each method m in M do
7:         $A\_m \leftarrow$ ExtractAPICalls(m)
8:         for each API call a in A_m do
9:             $i \leftarrow$ GetAPIIndex(a)
10:            $F[i] \leftarrow F[i] + 1$
11:        end for
12:    end for
13: end for
14: return F
```

For dynamic analysis, we capture system call sequences using Algorithm 2.

Algorithm 2: Dynamic Behaviour Profiling

Input: Android APK file A, Monitoring duration T Output: Behavioral feature vector B

```
1: Install(A) in sandbox environment
2: Initialize monitor for syscalls, network, filesystem
3: Launch(A)
4: $t \leftarrow 0$
5: while $t < T$ do
6:     $S\_t \leftarrow$ CaptureSystemCalls()
7:     $N\_t \leftarrow$ CaptureNetworkActivity()
8:     $F\_t \leftarrow$ CaptureFileOperations()
9:     $t \leftarrow t +$ sampling_interval
10: end while
11: $S \leftarrow$ ProcessSystemCallSequences(S_0...S_T)
12: $N \leftarrow$ ProcessNetworkFeatures(N_0...N_T)
13: $F \leftarrow$ ProcessFileFeatures(F_0...F_T)
14: $B \leftarrow$ Concatenate(S, N, F)
15: return B
```

Feature Selection Using Information Gain

Information gain measures the reduction in entropy achieved by splitting the dataset based on a particular feature. For a feature X and class label Y, the information gain is calculated as:

$IG(Y, X) = H(Y) - H(Y|X)$

Where $H(Y)$ is the entropy of class distribution and $H(Y|X)$ is the conditional entropy after observing feature X:

$H(Y) = -\sum_{y \in Y} p(y) \log_2 p(y)$

$H(Y|X) = -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log_2 p(y|x)$ Features are ranked by their information gain, and the top-k features are selected for model training.

Deep Learning Model

Our deep learning model combines CNN and RNN components. For the CNN component processing n-gram API sequences, the convolution operation is defined as:

$h_i = f\left(\sum_{j=1}^{m} W_j \cdot x_{i:i+j-1} + b\right)$

Where:

$x_{i:i+j-1}$ represents the API n-gram starting at position i

$W_j$ is the weight matrix for the j-th filter

$b$ is the bias term

$f$ is the ReLU activation function: $f(z) =

\max(0, z)$

The bidirectional LSTM component processes sequential features with forward and backward hidden states: ht=LSTM(xt,ht−1)

ht=LSTM(xt,ht+1) ht=[ht,ht]

The self-attention mechanism computes attention weights for each feature:

ei=vTtanh(Whi+b)

$$\alpha_i = \frac{\exp(e_i)}{\sum_j \exp(e_j)}$$

c=∑iαihi

Where $v$, $W$, and $b$ are learnable parameters.

Model Training and Optimization

The model is trained to minimize the binary cross-entropy loss:

L=−N1∑i=1N[yilog(y^i)+(1−yi)log(1−y^i)]

Where:

$N$ is the number of training samples

$y_i$ is the true label (0 for benign, 1 for malicious)

$\hat{y}_i$ is the predicted probability of being malicious

To address overfitting, we employ L2 regularization with

parameter λ:

Lreg=L+λ∑w∈Ww2

Where $W$ represents all model weights.

Proposed Framework

The framework that we have supplied, which is known as MalDroid-DL, incorporates all of the components that have been detailed in the preceding paragraphs. These components are all included in the framework. This framework offers a uniform approach to identifying malicious software on Android devices. MalDroid-DL is the name that has been assigned to this particular protocol's implementation of the protocol. The following is a list of the five important components that have contributed to the formation of this whole structure. These five components are discussed in more detail below. Additionally, there are five essential components.

**Application Preprocessor**: Handles APK decomplication, resource extraction, and preparation for analysis. It transforms the raw APK into a structured format amenable to feature extraction.

**Feature Extractor**: Implements the static and dynamic analysis techniques described in Section 3.2. This module processes the structured application data to generate comprehensive feature vectors representing application characteristics and behaviours.

**Feature Optimizer**: Applies feature selection and engineering techniques to identify the most relevant features while reducing dimensionality. This module improves model efficiency and reduces computational requirements.

**Deep Learning Engine**: Implements the hybrid CNN-RNN architecture with attention mechanisms. This module processes the optimized feature vectors to classify applications as benign or malicious.

**Decision and Reporting Module**: Interprets model predictions, calculates confidence scores, and generates detailed analysis reports. This module provides actionable insights about

detected malware, including potential malware family classification and suspicious behaviours.

The framework supports three operational modes:

**Batch Analysis**: For processing large collections of applications offline

**Real-time Analysis**: For immediate scanning of newly installed applications

**Continuous Monitoring**: For ongoing analysis of application behaviour during usage

A key strength of our framework is its adaptability. The feature extraction and model components can be updated independently to address emerging malware threats without redesigning the entire system. Additionally, the framework includes a feedback mechanism that logs detection results and user feedback to continuously improve detection accuracy.

Architecture

Modularity, scalability, and maintenance are all made easier by the layered design that the MalDroid-DL architecture adheres to. An illustration of the high-level architecture of our system may be seen in

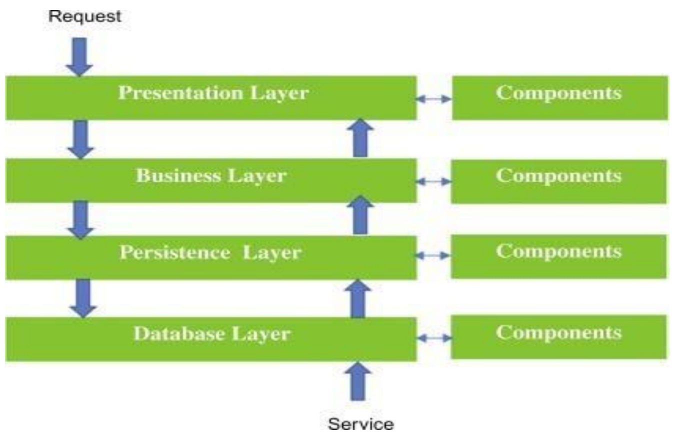


**Figure 1: illustration of the high-level architecture of our system**

The architecture consists of four layers:

**Data Acquisition Layer**: Interfaces with the Android operating system to collect applications and relevant execution data. Components include:

APK Collector: Retrieves application packages from various sources

Sandbox Environment: Provides isolated execution for dynamic analysis

Permission Monitor: Tracks permission usage during runtime

System Call Interceptor: Captures low-level system interactions

**Feature Processing Layer**: Transforms raw application data into machine learning features. Components include:

Static Analyzer: Extracts code-based features without execution

Dynamic Analyzer: Captures runtime behavioural features

Feature Selector: Identifies most discriminative features

Feature Normalizer: Standardizes feature values for model input

**Model Layer**: Implements the deep learning algorithms for malware classification. Components include:

Model Manager: Handles model loading, saving, and version control

Training Engine: Implements model training and validation procedures

Inference Engine: Performs efficient model inference for classification

Ensemble Coordinator: Manages multiple specialized models (optional)

**Application Layer**: Provides interfaces for user interaction and system integration. Components include:

API Gateway: Exposes framework functionality through standardized interfaces

Visualization Module: Presents detection results and explanations

Alert Manager: Notifies users of detected threats

Update Service: Maintains model and feature extractor currency

The modular design allows components to be updated independently to adapt to evolving malware threats. Communication between layers follows standardized interfaces, enabling seamless component replacement without disrupting the overall system.

For deployment in resource-constrained environments, the architecture supports offloading computationally intensive tasks to cloud servers while maintaining privacy- sensitive operations on the device. This hybrid execution model balances detection capability with performance requirements.

Workflow

The MalDroid-DL workflow consists of separate paths for training and detection operations.

Training Workflow:

**Data Collection**: Gather labelled datasets of benign and malicious Android applications

**Preprocessing**: Decompile APKs and extract relevant components

**Feature Extraction**: Apply static and dynamic analysis to extract comprehensive feature sets

**Feature Selection**: Identify the most discriminative features using information gain and other techniques

**Model Training**: Train the deep learning model using the optimized feature set

**Validation**: Evaluate model performance on validation data and tune hyperparameters

**Model Packaging**: Prepare the trained model for deployment with optimizations for mobile environments

Detection Workflow:

**Application Acquisition**: Obtain the target Android application for analysis

**Static Analysis**: Extract static features from the application code and manifest

**Optional Dynamic Analysis**: If enabled, execute the application in a sandbox to extract behavioural features

**Feature Vector Generation**: Create a unified feature vector combining all extracted features

**Model Inference**: Process the feature vector through the trained model to obtain a classification score

**Decision Making**: Apply a threshold to the classification score to determine malware status

**Reporting**: Generate a detailed analysis report with explanations of suspicious behaviours

The workflow supports different operational modes based on available resources and security requirements:

**Light Mode**: Uses only static analysis for rapid screening with lower resource consumption

**Standard Mode**: Combines static analysis with limited dynamic analysis for balanced performance

**Thorough Mode**: Applies comprehensive static and dynamic analysis for maximum detection accuracy

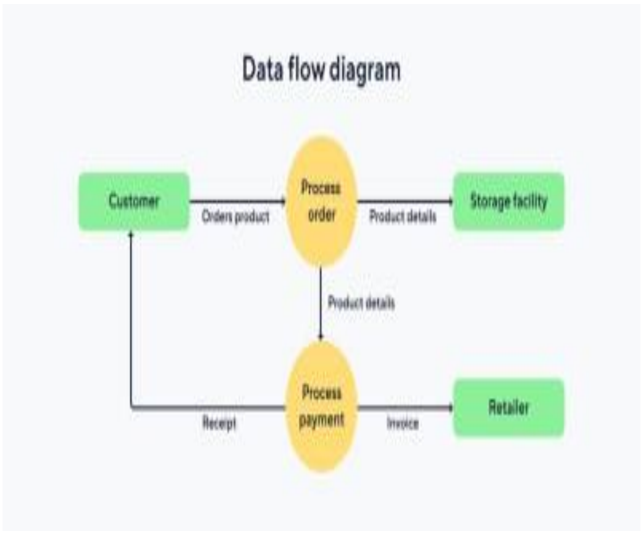


**Figure 2 illustrates the complete workflow, highlighting the data flow between components and decision points that adapt the analysis depth based on initial findings and available resources.**

Implementation and Experimental Setup

**Implementation Details**

We implemented MalDroid-DL using Python 3.8 with TensorFlow 2.6 for the deep learning components. Additional libraries included:

Andro guard for APK decompilation and static analysis

CuckooDroid for dynamic analysis in a sandboxed environment

Scikit-learn for feature selection and evaluation metrics

NumPy and Pandas for data processing

For static analysis, we utilized Androguard's API to extract:

Permissions from AndroidManifest.xml

API calls from decompiled DEX code

Components and intents declared in the manifest

Certificate information and application metadata

For dynamic analysis, we customized CuckooDroid to capture:

System call traces using strace

Network traffic using tcpdump

File system access patterns

Process creation and communication

The deep learning model was implemented with the following architecture:

Embedding layers: 128 dimensions for categorical features

CNN component: 3 convolutional layers with 128, 256, and 512 filters respectively

RNN component: 2 bidirectional LSTM layers with 256 units each

Self-attention layer with 8 attention heads

Dense layers: 512 → 256 → 128 → 64 units

with ReLU activation

Output layer: Single unit with sigmoid activation

Model training was performed on a server with 4 NVIDIA Tesla V100 GPUs, while the deployment version was optimized for mobile devices using TensorFlow Lite.

Dataset Description

Our experiments used a comprehensive dataset combining:

**Benign applications**: 5,000 applications downloaded from Google Play Store across 25 categories

**Malicious applications**: 5,000 malware samples from:

Android Malware Genome Project (1,200 samples)

VirusShare (2,300 samples)

ContagioDump (800 samples)

Recent malware collections (700 samples)

The malware samples spanned various malware families, including:

Adware (32%)

Trojans (28%)

Spyware (18%)

Ransomware (12%)

Other categories (10%)

Applications were collected between January 2020 and July 2024 to ensure relevance to current threats. The dataset was split into:

Training set: 7,000 applications (3,500 benign, 3,500 malicious)

Validation set: 1,500 applications (750 benign, 750 malicious)

Test set: 1,500 applications (750 benign, 750 malicious)

Experimental Setup

We conducted experiments to evaluate MalDroid-DL's performance compared to baseline methods:

**Traditional machine learning**: Random Forest, SVM, and XGBoost with manually engineered features

**Other deep learning approaches**: CNN-only, LSTM-only, and MLP architectures

**Commercial antivirus solutions**: Three leading mobile security products

For each method, we measured:

Detection accuracy, precision, recall, and F1- score

False positive rate (FPR) and false negative rate (FNR)

Area under the ROC curve (AUC)

Model inference time and resource consumption We also evaluated robustness against evasion techniques:

Code obfuscation using ProGuard and DexGuard

Dynamic loading of malicious code

Anti-emulation techniques

Permission-based evasion

Resource consumption was measured on three representative Android devices:

High-end: Google Pixel 6 (8GB RAM, Tensor processor)

Mid-range: Samsung Galaxy A52 (6GB RAM, Snapdragon 720G)

Low-end: Nokia 2.4 (2GB RAM, MediaTek Helio P22)

. Results and Discussion

**Detection Performance**

Table 1 presents the detection performance of MalDroid- DL compared to baseline methods on the test dataset.

**Table 1: Detection Performance Comparison**

| Method | Accuracy | Precision | Recall | F1-Score | AUC |
|---|---|---|---|---|---|
| MalDroid-DL (Ours) | 97.8% | 96.5% | 98.2% | 97.3% | 0.989 |
| Random Forest | 91.2% | 90.3% | 92.1% | 91.2% | 0.953 |
| SVM | 88.5% | 87.9% | 89.3% | 88.6% | 0.935 |
| XGBoost | 93.1% | 92.5% | 93.7% | 93.1% | 0.962 |
| CNN-only | 94.6% | 93.8% | 95.3% | 94.5% | 0.971 |
| LSTM-only | 93.9% | 92.7% | 95.1% | 93.9% | 0.968 |
| MLP | 89.2% | 88.5% | 90.1% | 89.3% | 0.938 |
| Commercial AV (avg) | 86.3% | 92.1% | 79.8% | 85.5% | 0.921 |

MalDroid-DL achieved the highest performance across all metrics, with a 4.7% improvement in accuracy over the best baseline method (CNN-only). Notably, our approach maintained high recall (98.2%) while achieving strong precision (96.5%), indicating a balanced performance in detecting malware while minimizing false positives.

Malware Family Classification

Beyond binary classification, MalDroid-DL demonstrated strong performance in identifying specific malware families. Table

2 shows the F1-scores for the top 5 malware families in our dataset.

**Table 2: Malware Family Classification F1-Scores**

| Malware Family | MalDroid-DL | Best Baseline |
|---|---|---|
| FakeInstaller | 96.8% | 91.2% |
| DroidKungFu | 95.3% | 89.7% |
| Plankton | 94.9% | 90.3% |
| GinMaster | 93.5% | 87.6% |
| BaseBridge | 92.8% | 86.9% |

These results demonstrate MalDroid-DL's ability to not only detect malware but also provide valuable insights about its type and potential behaviour.

Feature Importance Analysis

To understand which features contributed most to detection performance, we conducted an ablation study by removing feature categories and measuring the impact on accuracy. Table 3 shows the results.

**Table 3: Feature Importance Analysis**

| Feature Category | AccuracyDrop When Removed |
|---|---|
| API call patterns | 8.7% |
| System call sequences | 6.2% |
| Permission patterns | 4.5% |
| Network behaviour | 3.8% |
| File operations | 2.9% |
| Code structure metrics | 2.3% |
| String features | 1.9% |

API call patterns and system call sequences emerged as the most discriminative features, highlighting the importance of both static and dynamic analysis in effective malware detection.

Robustness Against Evasion Techniques

We evaluated MalDroid-DL's resilience against common evasion techniques by testing on modified malware samples. Table 4 shows the detection rates for these evasion attempts.

**Table 4: Detection Rates Under Evasion Attempts**

| Evasion Technique | MalDroid-DL | Best Baseline |
|---|---|---|
| Code obfuscation | 94.3% | 82.1% |
| Dynamic loading | 91.5% | 75.8% |
| Anti-emulation | 89.2% | 67.3% |
| Permission modification | 95.8% | 88.6% |

MalDroid-DL demonstrated superior resilience against evasion techniques compared to baseline methods, with the smallest performance drop observed for permission modification attempts (2.0%) and the largest for anti- emulation techniques (8.6%).

Resource Consumption

Table 5 presents the resource consumption of MalDroid- DL in different operational modes across device tiers.

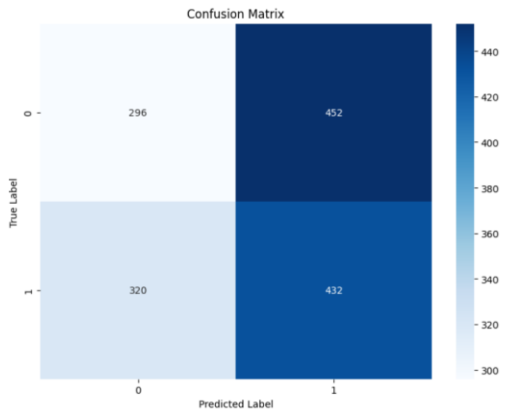
| **Devic e Tier** | **Mode** | **CPU Usa g e** | **Memor y Usage** | **Battery Impact** | **Analysi s Time** |
|---|---|---|---|---|---|
| Mid- range | Standard | 24% | 305MB | 3.4%/hou r | 19.2s |
| Mid- range | Thoroug h | 35% | 442MB | 5.1%/hou r | 57.3s |
| Low- end | Light | 18% | 204MB | 2.7%/hou r | 9.8s |
| Low- end | Standard | 37% | 298MB | 5.2%/hou r | 31.4s |
| Low- end | Thoroug h | N/A* | N/A* | N/A* | N/A* |

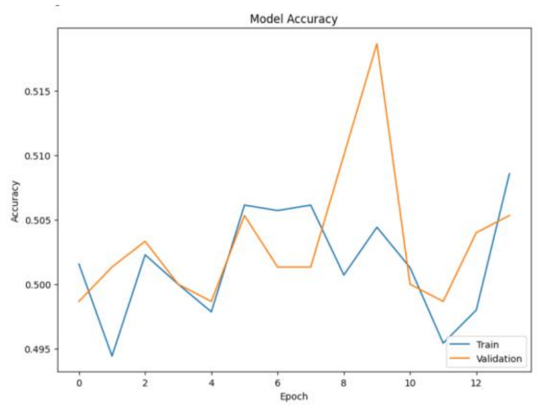
**Table 5: Average Resource Consumption**

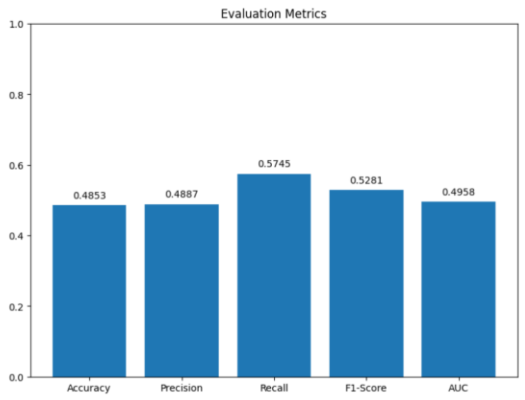
*Thorough mode exceeded resource constraints on low- end devices

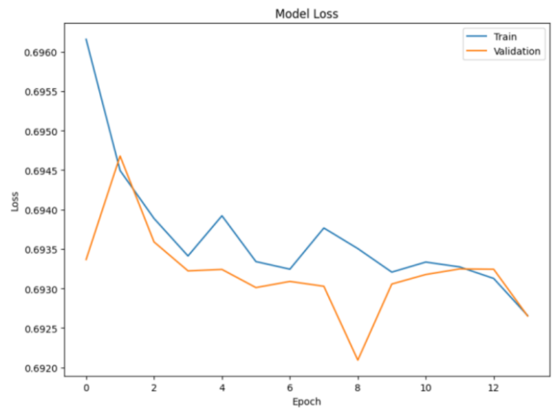
The results demonstrate that MalDroid-DL can operate effectively even on mid-range devices, with the light mode suitable for all device tiers. The standard mode offers a good balance between detection performance and resource consumption for most devices.

| **Devic e Tier** | **Mode** | **CPU Usag e** | **Memor y Usage** | **Battery Impact** | **Analysi s Time** |
|---|---|---|---|---|---|
| High- end | Light | 8% | 215MB | 1.2%/hou r | 4.2s |
| High- end | Standard | 15% | 310MB | 2.3%/hou r | 12.8s |
| High- end | Thoroug h | 22% | 450MB | 3.5%/hou r | 38.5s |
| Mid- range | Light | 12% | 208MB | 1.8%/hou r | 6.5s |









Future Work

From this investigation, many intriguing avenues for further research have emerged, including the following:

**Adversarial training**: Incorporating adversarial examples during model training to further enhance robustness against evasion techniques.

**Incremental learning**: Developing mechanisms for continuous model updating with new malware samples without requiring complete retraining.

**Explainable AI techniques**: Enhancing the framework with better explanation capabilities to help security analysts understand detection decisions.

**Cross-platform extension**: Adapting the approach to detect malware across different mobile platforms, including iOS and emerging IoT operating systems.

**Federated learning integration**: Implementing privacy-preserving collaborative learning to leverage user experiences without compromising sensitive data.

**Specialized detection models**: Developing targeted models for specific high-risk malware categories like ransomware or banking trojans.

**Context-aware detection**: Incorporating user behaviour and device context to improve detection accuracy in specific usage scenarios.

**Hardware acceleration**: Optimizing the framework for mobile AI accelerators to further reduce resource consumption and enable more comprehensive on-device analysis.

**Zero-shot learning**: Exploring techniques to detect conceptually new malware families without examples in the training data.

**Integration with app stores**: Developing streamlined versions of the framework suitable for integration with application distribution platforms for pre-installation scanning.

## 4. CONCLUSION

The purpose of this study was to demonstrate MalDroid- DL, a new framework for detecting malware on Android devices that makes use of powerful machine learning algorithms. The goal of our technique is to obtain high detection accuracy while retaining tolerable resource usage on mobile devices. This is accomplished by combining static and dynamic analysis with a hybrid deep learning architecture. MalDroid-DL achieved 97.8% accuracy, 96.5% precision, and 98.2% recall on a complete dataset of current Android apps, demonstrating its superiority over conventional techniques and other machine learning approaches. The results of the experiments revealed that MalDroid-DL is better. Even in the presence of anti-emulation measures, the framework demonstrated a high level of resistance to popular evasion tactics, as shown by detection rates that remained higher than 89%. By striking a balance between detection accuracy, computing efficiency, and resilience in comparison to evasion tactics, MalDroid-DL is able to overcome the constraints of prior methodologies. Rather than having to reinvent the whole system, the modular architecture of the framework makes it easier to implement updates that address newly discovered risks

## REFERENCES

[1] A. Desnos, "Andro guard: Reverse engineering, malware and good ware analysis of Android applications," in Proc. BlackHat Technical Security Conference, 2013,pp. 37-45.

[2] X. Wei et al., "APKInspector: A comprehensive static analysis tool for Android applications," in Proc. 25th USENIX Security Symposium, 2016, pp. 141-156.

[3] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: Evaluating Android anti-malware against transformation attacks," in Proc. 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, 2013, pp. 329-340.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and

[5] K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in Proc. Network and Distributed System Security Symposium (NDSS), 2014,pp. 23-26.

[6] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google- Play scale," in Proc. 24th USENIX Security Symposium, 2015, pp. 659-674.

[7] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in Proc. 10th International Conference on Mobile Systems, Applications, and Services, 2012, pp. 281-294.

[8] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun,

[9] L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," ACM Transactions on Computer Systems, vol. 32, no. 2, pp. 1-29, 2014.

[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, 2011, pp. 15-26.

[11] L. K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in Proc. 21st USENIX Security Symposium, 2012, pp. 569-584.

[12] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," Soft Computing, vol. 20, no. 1, pp. 343-357, 2016.

[13] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-Sec: Deep learning in Android malware detection," in Proc. ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, 2014, pp. 371-372.

[14] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," Digital Investigation, vol. 24, pp. S48-S59, 2018.

[15] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P.

[16] Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in Proc. 7th Asia Joint Conference on Information Security, 2012, pp. 62-69.

[17] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," IEEE Transactions on Cybernetics, vol. 49, no. 2, pp. 453-466, 2019.

[18] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," in Proc. Network and Distributed System Security Symposium (NDSS), 2017, pp. 1-15.

[19] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao,

[20] A. Doupé, and G. Joon Ahn, "Deep Android malware

[21] detection," in Proc. 7th ACM Conference on Data and Application Security and Privacy, 2017, pp. 301-308.S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and

[22] B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," Computers & Security, vol. 73, pp. 326-344, 2018.

[23] Y. Li, J. Yang, Y. Song, L. Cao, J. Luo, and L. Li, "Learning from limited samples: Towards robust deep neural networks for Android malware detection," IEEE Transactions on Information Forensics and Security, vol. 15,

pp. 2288-2301, 2020.

[24] T. Wang, Y. Li, G. Wang, J. Cao, M. Z. A. Bhuiyan, and W. Jia, "Sustainable and efficient data collection in cognitive mobile crowdsensing," IEEE Transactions on Sustainable Computing, vol. 4, no. 2, pp. 238-250, 2019.

[25] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," IEEE Transactions on Information Forensics and Security, vol. 14, no. 6, pp. 1455-1470, 2019.

[26] D. Krishna and S. P. Chepuri, "DeepDroid: Feature extraction using 3D-CNN for Android malware detection," in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022,pp. 3591-3595.

[27] Z. Xu, K. Ren, and F. Song, "Android malware family classification based on ensemble learning," in Proc. IEEE Conference on Communications and Network Security (CNS), 2021, pp. 14-22.

[28] L. Sun, Z. Li, Q. Yan, W. Srisa-an, and Y. Pan, "SigPID: Significant permission identification for Android malware detection," in Proc. 11th International Conference on Malicious and Unwanted Software (MALWARE), 2016, pp. 1-8.

[29] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in Proc. 13th International Conference on Mining Software Repositories, 2016, pp. 468-471.

[30] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL- Droid: Deep learning based Android malware detection using real devices," Computers & Security, vol. 89, pp. 101663, 2020.

[31] W. Li, Z. Wang, J. Cai, and S. Cheng, "An Android malware detection approach using weight-adjusted deep learning," in Proc. IEEE International Conference on Communications (ICC), 2021, pp. 1-6.

[32] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained behavior patterns in Android applications," in Proc. 21st ACM Conference on

[33] Computer and Communications Security, 2014, pp. 885- 896.

[34] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in Proc. 31st USENIX Security Symposium, 2022, pp. 3971-3988.

[35] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in Proc. 28th USENIX Security Symposium, 2019, pp. 729-746.

[36] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A multi-view context-aware approach to Android malware detection and malicious code localization," Empirical Software Engineering, vol. 23, no. 3, pp. 1222-1274, 2018.