Smartphone Malware Detection based on Enhanced Correlation -based Feature Selection on Permissions

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

MASTER OF SCIENCE

IN

MATHEMATICS

Submitted by:

Shagun

(2K21/MSCMAT/47)

Deepak Kumar

(2K21/MSCMAT/12)

Under the supervision of

Dr. ANSHUL ARORA

DEPARTMENT OF APPLIED MATHEMATICS

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

We, Shagun (2K21/MSCMAT/47) and Deepak Kumar (2K21/MSCMAT/12), students of MSc in Mathematics, hereby solemnly declare that the Dissertation titled "Smartphone Malware Detection based on Enhanced Techniques," which we have submitted to the Department of Applied Mathematics at Delhi Technological University, Delhi, fulfills the requirements for the degree of Master of Science. We affirm that this project is entirely original and has been developed independently, without any form of plagiarism or improper citation. We have diligently conducted our research, and the content presented in this dissertation is the result of our own intellectual efforts. Furthermore, this work has not been previously submitted or utilized in any way to obtain any other degree, diploma, associateship, fellowship, or similar academic recognition. We acknowledge the importance of academic integrity and take full responsibility for the authenticity and originality of our project.

Place: Delhi SHAGUN AND DEEPAK KUMAR

Date: 25 th May 2023

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I, Anshul Arora, hereby confirm that the Project Dissertation titled " Smartphone Malware Detection based on Enhanced Techniques " has been submitted by Shagun (2K21/MSCMAT/47) and Deepak Kumar (2K21/MSCMAT/12) from the Department of Applied Mathematics at Delhi Technological University. This dissertation fulfills partial requirements for the Master of Science degree.

I attest that this project represents the diligent efforts of the students under my supervision. To the best of my knowledge, this work has not been previously submitted, either partially or in its entirety, for any degree or diploma at this University or elsewhere.

Place: Delhi

Date: 25 th May 2023

Dr ANSHUL ARORA

SUPERVISOR

ASSISTANT PROFESSOR

ABSTRACT

In the present day, smartphones are becoming increasingly ubiquitous, with people of all ages relying on them for daily use. The number of app downloads continues to skyrocket, with 1.6 million apps downloaded every hour in 2022, amounting to a staggering total of 142.6 billion downloads. Google Play outpaces iOS with 110.1 billion downloads compared to iOS's 32.6 billion. Given the growing threat of malware applications for Android users, it is essential to quickly and effectively identify such apps. App permissions represent a promising approach to malware detection, particularly for Android users. Researchers are actively exploring various techniques for analyzing app permissions to enhance the accuracy of malware detection. Overall, understanding the importance of app permissions in identifying potentially harmful apps is a critical step in protecting smartphone users from malware threats.

In our thesis, we have successfully employed the Enhanced Correlation-based Feature Selection (ECFS) technique to discern the nature of mobile applications, distinguishing between malicious and non-malicious ones. This approach leverages both feature-feature and feature-class correlation scores, specifically the ENMRS and crRelevance measures, to compute the relevance of each feature. By employing ECFS, we were able to identify the most informative features for accurate prediction.

We further assessed the performance of various Machine Learning Techniques by utilizing the ECFS scores. Notably, we achieved the highest accuracy of 92.25% by employing the Random Forest ML Technique. This accuracy was obtained by setting the n1 and n2 values to 0.9 and 0.1, respectively. Our findings highlight the effectiveness of ECFS in enhancing the prediction accuracy for distinguishing between malicious and non-malicious apps, with Random Forest emerging as the most successful ML Technique in this regard.

ACKNOWLEDGEMENT

The culmination of an endeavor, marked by its successful completion, owes its realization to the collective efforts of individuals who have been involved implicitly or explicitly. The meticulous execution of the planning and research phases of our project stands as a testament to the unwavering dedication exhibited by numerous individuals, and it is with utmost sincerity that we acknowledge their invaluable contributions. This acknowledgment serves as a humble expression of gratitude, recognizing their indispensable assistance in our pursuit.

First and foremost, we extend our heartfelt appreciation to our esteemed project guide, Prof. Anshul Arora ,whose profound guidance and expertise have consistently guided us through the complexities encountered and Prof. S. Sivaprasad Kumar (Professor), the esteemed Head of the Department of Applied Mathematics, along with all the faculties who have been instrumental in our learning journey. . It has been a great honor and privilege to work under their guidance. Their profound teachings and invaluable insights have greatly enriched our understanding and paved the way for the successful execution of our work on " Smartphone Malware Detection based on Enhanced Techniques ".

Last but certainly not least, we extend our heartfelt thanks to our colleagues, friends, and parents, who have been unwavering sources of encouragement and inspiration throughout the course of this project. Their constant support and belief in our abilities have propelled us forward, enabling us to overcome challenges and achieve our goals.

In conclusion, we are deeply indebted to all those who have played a significant role in our journey, and we wholeheartedly express our sincerest appreciation for their unwavering support, guidance, and inspiration.

Thanking You

Shagun

Deepak Kumar

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 COMMENCEMENT

Over the past decade, smartphones have experienced an unprecedented rise in popularity, transforming from niche gadgets to ubiquitous necessities in our modern world. With each passing year, smartphones have become more affordable, technologically advanced, and accessible to a wider range of users, leading to explosive growth in adoption rates. According to industry reports, the global smartphone market is projected to continue its upward trajectory, with an estimated 5.5 billion smartphone users by 2025. The global smartphone market size was valued at USD 457.18 billion in 2021 and is projected to grow from USD 484.81 billion in 2022 to USD 792.51 billion by 2029, exhibiting a CAGR of 7.3% during the forecast period.

This phenomenal growth can be attributed to several factors, including the increasing demand for mobile internet access, the proliferation of social media, the rise of e-commerce, and the integration of smartphones into various aspects of our daily lives. From communication and entertainment to productivity and beyond, smartphones have become indispensable tool for people of all ages and backgrounds. As smartphones continue to evolve with new features and capabilities, such as augmented reality, artificial intelligence, and 5G connectivity, their popularity is expected to continue growing in the foreseeable future, shaping the way we live, work, and connect in a rapidly changing digital landscape.

The versatility of smartphones is a key factor that contributes to their widespread appeal. They have become an all-in-one device that seamlessly integrates various aspects of our lives into a single device. In fact, for many people, smartphones have become the primary means of accessing the internet, checking emails, and staying connected with the world.The convenience and flexibility offered by smartphones have made them an essential tool for modern living.

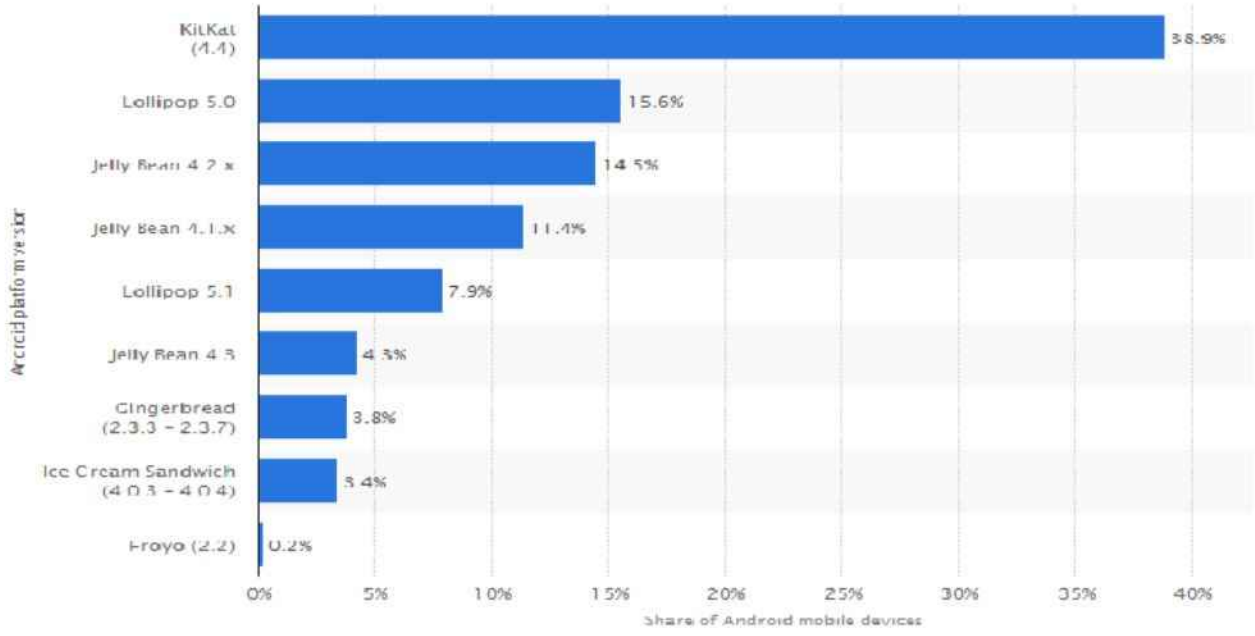
Moreover, smartphones have become more accessible and affordable than ever before. The availability of budget-friendly smartphone models and affordable data plans has expanded the

consumer base, allowing more people to purchase smartphones. This has resulted in increased smartphone adoption, particularly in emerging markets where affordability plays a crucial role in consumer choices. As smartphones become more affordable, they become more accessible to a wider range of individuals, driving their popularity even further.

Additionally, the constant evolution of technology has also contributed to the growing popularity of smartphones. Technological advancements in areas such as processor power, camera capabilities, and connectivity options have made smartphones more powerful, feature-rich, and attractive to consumers. As new technologies emerge and smartphones continue to innovate, consumers are drawn to the latest and greatest offerings, further fueling the popularity of these devices.

In conclusion, the adaptability of smartphones, their affordability, and the continuous evolution of technology are key factors that have contributed to the widespread popularity of smartphones. They have become indispensable companions in our modern lives, offering versatility, convenience, and accessibility that appeal to a wide range of consumers. As technology continues to advance, smartphones are likely to remain a dominant force in the realm of consumer electronics, shaping the way we live, work, and connect in the digital age.

Graph showing use of android in Fig.1.1



When it comes to mobile operating systems, Android has gained significant popularity in recent years, emerging as a dominant force in the smartphone market.

Android's widespread usage can be attributed to several factors. One of the main reasons is the sheer diversity of devices that run on Android, ranging from budget-friendly options to high-end flagship phones, catering to a wide range of consumers with varying budgets and preferences. This versatility has allowed Android to capture a larger market share, particularly in emerging markets where affordability plays a crucial role in consumer decisions. Additionally, Android's open-source nature has fostered a vibrant ecosystem of developers and app creators, resulting in a plethora of apps and customization options for users.

Android stands out as the undeniable front-runner when considering the global usage of mobile operating systems. According to recent findings by Stat counter, Android commands a staggering 71.45 percent of the worldwide market share, while iOS trails behind with a 27.83 percent share. Together, these two giants account for over 99 percent of the total market share, leaving scant room for other contenders like Samsung and KaiOS, which collectively make up less than 1 percent of the market. These numbers clearly highlight the indomitable dominance of Android and iOS as the preeminent mobile operating systems that remain unrivaled in the industry.

The ability to customize and personalize the user experience has been a significant draw for many Android users. Moreover, Android's seamless integration with Google services, such as Google Drive, Google Maps, and Google Assistant, has also played a pivotal role in its widespread adoption. Finally, Android's compatibility with a wide range of third-party devices and accessories, such as smartwatches, smart TVs, and smart home devices, has further cemented its position as a preferred choice for tech-savvy users who seek seamless connectivity across different devices. Overall, Android's flexibility, affordability, customization options, and compatibility have contributed to its growing popularity and market dominance in the realm of mobile operating systems.

1.2 MOTIVATION

Android has emerged as the primary target for malware apps due to several factors. First and foremost, Android's widespread adoption as the most widely used mobile operating system makes it an attractive target for cybercriminals seeking a large user base to exploit. Additionally, the open-source nature of Android allows for customization and flexibility, but it also means that potential vulnerabilities can be exploited by malicious actors. The decentralized nature of the Android app ecosystem, with multiple app stores and varying levels of app review processes, can also create opportunities for malware to slip through the cracks.

Furthermore, the diverse hardware and software configurations across different Android devices can make it challenging to implement uniform security measures.

Lastly, the popularity of third-party app stores and the availability of apps outside of the official Google Play Store can increase the risk of downloading malware-laden apps. Collectively, these factors make Android the biggest target for malware apps, necessitating robust security measures to safeguard users' devices and data.

Android users, like users of any operating system, may fall victim to social engineering attacks, such as phishing, scams, and other forms of social manipulation, which can lead to the unwitting installation of malware. Human error, lack of awareness, and risky online behavior can all contribute to the increased likelihood of malware attacks on Android devices.

During 2022, the worldwide number of malware attacks reached 5.5 billion, an increase of two percent compared to the preceding year. In recent years, the highest number of malware attacks was detected in 2018, when 10.5 billion such attacks were reported across the globe.

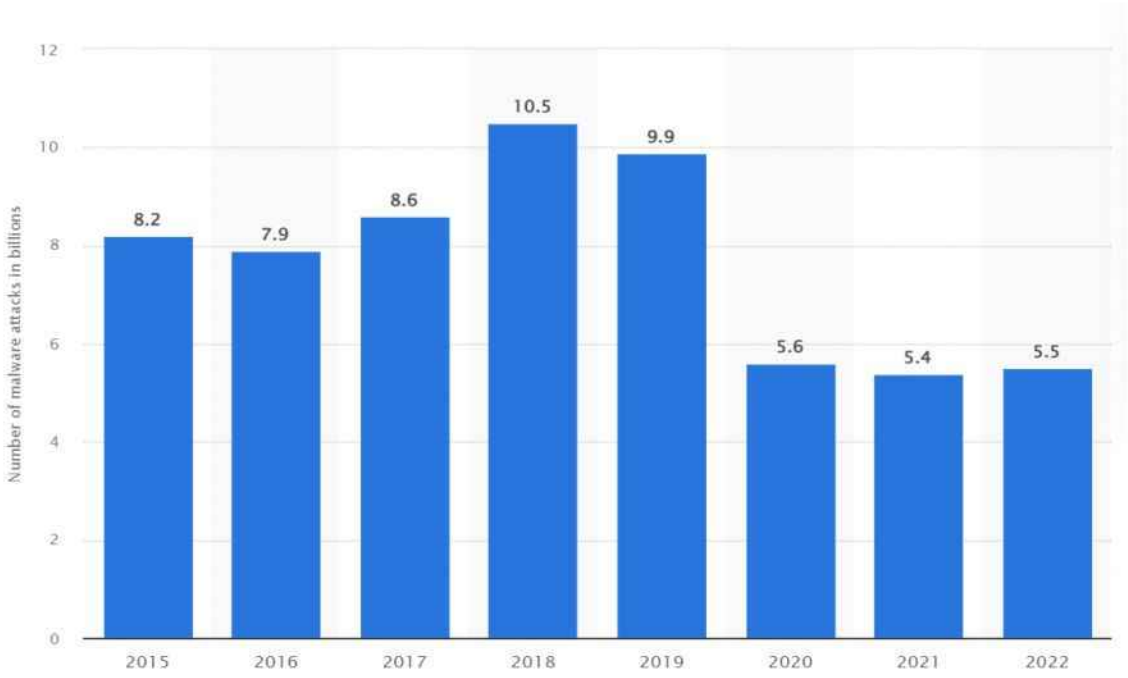


Fig. 1.2 showing number of malware attacks (in billions) from year 2015 to 2022.

Malware, or malicious software, can pose various risks to Android devices. Some potential risks of having malware on an Android device may include:

1. **Financial Loss:** Certain malware can initiate unauthorized transactions, make premium service subscriptions, or send SMS messages to premium numbers, resulting in unexpected charges and financial losses.

2. **Data Theft:** Malware can steal sensitive information from your device, such as passwords, credit card details, personal documents, and more. This information can be used for identity theft or financial fraud.

3. **Privacy Invasion:** Malware may access and collect personal data, including contacts, messages, photos, and browsing history, without your consent. This invasion of privacy can lead to various forms of misuse or exploitation.

4. **Unauthorized Access:** Some malware can grant remote access to cybercriminals, allowing them to control your device, install additional malicious software, or use it as part of a botnet for illegal activities.

5. **Performance Degradation:** Malware can consume system resources, such as CPU and memory, causing your device to slow down, freeze, or crash frequently. This degradation in performance affects the overall user experience.

6. **Battery Drain and Excessive Data Usage:** Malware may run in the background, constantly consuming battery power and using up your data plan, resulting in faster battery drain and increased data charges.

7. **Malicious Activity Propagation:** Malware can spread to other devices through various means, such as infecting shared files, apps, or network connections. This can lead to a wider impact and potential harm to others.

It is crucial to install reputable antivirus software, regularly update your device's operating system and apps, and exercise caution when downloading apps or clicking on suspicious links to mitigate the risks associated with malware on Android devices.

Efforts to develop effective techniques for detecting malware in application stores are critical due to the dynamic nature of permission usage in apps.

Common issues with permission feature-based detection methods include:

1. Over-Privileged Apps: Permission-based detection relies on analyzing the permissions requested by apps. However, some legitimate apps may request excessive permissions that they don't actually require for their intended functionality. This can lead to false positives, where harmless apps are flagged as potentially malicious.

2. Limited Contextual Information: Permission-based detection focuses solely on the requested permissions of an app, without considering the context or purpose of those permissions. This can result in limited insight into the app's actual behavior and potential risks, as malicious apps can dynamically misuse granted permissions.

3. Permission Abuses: Malicious apps can employ various techniques, such as permission misuse or permission chaining, to bypass permission-based detection. They may request seemingly benign permissions individually but combine their capabilities to perform harmful actions once installed on a user's device.

4. Lack of Timeliness: Permission-based detection methods may not keep up with the evolving landscape of app permissions. As new permissions are introduced or existing ones are repurposed, detection systems may lag behind in detecting potential risks associated with these changes.

5. Difficulty Detecting Stealthy Malware: Sophisticated malware can employ obfuscation techniques to hide their malicious behavior and evade detection. Permission-based methods alone may struggle to identify such stealthy malware, as they primarily rely on static analysis of requested permissions.

6. Limited Coverage: Permission-based detection methods focus solely on permissions and may not consider other aspects of an app's behavior or characteristics. This can result in overlooking certain types of malware that do not exhibit suspicious permission patterns but engage in other malicious activities.

By leveraging innovative techniques, the proposed work aims to provide a fresh perspective on detecting malicious apps in Android.

Proposed work different from others :

The novelty of our proposed work can be best described on the basis of the statistical selection procedure that we have adopted, there are many works that use only feature class correlation adoption method, but we used both feature-feature and feature-class correlation based on Enhanced Correlation-based Feature Selection (ECFS). The preliminary results based on the work we did were satisfactory but need more evaluation on evaluating for various values of n1 and n2 in future work.

1.3 CONTRIBUTIONS

In this thesis, we have used a statistical feature selection technique called Enhanced Correlation-based Feature Selection (ECFS) which uses feature-feature correlation scores evaluated using ENMRS and feature-class correlation scores evaluated using crRelevance. The ECFS method was introduced by the authors of [3] for using these correlations effectively to extract relevant feature subsets from multi-class gene expression and other machine learning datasets. They then evaluated the performance of ECFS using decision tree, random forest, and KNN classifiers, and was found to be highly satisfactory across multiple benchmark datasets.

We have adopted this feature selection technique in our thesis for a multi-binary form of data which has features named as different permissions needed by malicious and benign apps. Moreover, the objects that were required by our adopted method are in the form of malicious and non-malicious application names, and as for the multi-class parameter we have defined Class A for non-malicious applications and Class B for malicious applications. The following points summarize the contributions of this work.

1. Data collection from Androzoo and google play store.
2. We extracted permissions from malicious and non-malicious applications.
3. Data cleaning, renaming permissions as feature 1, feature 2,.. feature 129. Also putting the columns and features into python 2-D lists for easier estimation of ENMRS and crRelevance
4. We evaluated ENMRS scores for both malicious and non-malicious applications.
5. We evaluated crRelevance scores for both malicious and non-malicious applications.
6. Next, we evaluated ECFS scores for different values of n1 and n2.
7. Lastly, we evaluated the accuracy for each combination of n1 and n2 using various machine-learning techniques.
8. We concluded our paper by noting the highest accuracy achieved is 92.25 \% for the combination n1=0.9 and n2=0.1 with the Random Forest technique.

CHAPTER 2 LITERATURE REVIEW

2.1 RELATED WORK

In this section, we shall embark on an intriguing expedition, delving into the depths of preexisting or interconnected studies conducted in this specialized domain. The authors in [3] proposed a permission-ensemble-based mechanism to detect Android malware with permission combinations. The authors in [4], developed a new method for Android application analysis that involved using static analysis to collect important features and passing them to a functional API deep learning model.

Li et al [5] described a reliable Android malware classifier using Factorization Machine architecture and app feature extraction. Their results showed that interactions among features were critical to revealing malicious behavior patterns. Qiu et. al. [6] proposed Multiview Feature Intelligence (MFI) for detecting evolving Android malware with similar capabilities. MFI extracts features via reverse engineering to identify specific capabilities from known malware groups and detect new malware with the same capability.

The authors in [7] proposed a hybrid deep learning-based malware detection method, utilizing Convolutional Neural Networks and Bidirectional Long Short-Term Memory (BiLSTM) to accurately detect long-lasting malware. The authors in [8] introduced a Malware Capability Annotation (MCA) to detect security-related functionalities of discovered malware. MCA analyzes zero-day family malware using knowledge from known malware groups that share similar abilities.

The authors in [9] proposed a malware detection mechanism using transparent artificial intelligence. This approach leverages app attributes to distinguish harmful from harmless malware.Khalid et. al [10] analyzed the impact of dynamic analysis categories and features on Android malware detection. Using filter and wrapper methods,identified the most significant categories and list important features within them..

The authors in [11] introduced SHERLOCK, a deep learning algorithm that uses self-supervision, and ViT model to identify malware. SHERLOCK learns distinguishing attributes from binary image-based representations to separate harmful from safe software.

The authors in [12] identified and ranked permissions commonly found in normal and malicious apps. They proposed a machine learning algorithm that detects Android malware by analyzing hybrid vectors containing permissions and traffic features.

Li et al. [13] proposed a stealthy backdoor that is triggered when a specific app is introduced and demonstrated the attack on common malware detectors. The authors in [14] introduced AndroOBFS, a released obfuscated malware dataset spanning three years (2018-2020). It consisted of 16,279 real-world malware samples across six obfuscation categories, providing valuable temporal information.

The authors in [15] proposed AdMat, a framework that uses an adjacency matrix to classify Android apps as images. This enables the Convolutional Neural Network to differentiate between benign and malicious apps,even identifying malware families, making it a simple yet effective way to analyze Android applications.

Canfora et al. [16] designed LEILA, a tool that uses model checking to verify Java bytecode and detect Android malware families. This paper presents LEILA's novel approach, design, and implementation as a formal tool for identifying malicious behavior in mobile devices.

Yousefi-Azar et al. [17] proposed Byte2vec, which improves static malware detection by embedding semantic similarity of byte-level codes into feature and context vectors. It allows for binary file feature representation and selection, enhancing malware detection capabilities. The authors in [18] presented Alterdroid, a dynamic analysis approach for detecting obfuscated malware components within apps. It works by creating modified versions of the original app and observing the behavioral differences.

Eom et al. [19] used three feature selection methods to build a machine learning-based Android malware detector, showing its effectiveness on the Malware Genome Project dataset and their own collected data. Zhang et al. [20] proposed a process for Android malware detection using static analysis and ensemble learning.It incorporates semantics-based features

to overcome obfuscation techniques, and collects features through static analysis of code and app characteristics..

Dissanayake et. al [21] study evaluates K Nearest Neighbor (KNN) algorithm's performance with different distance metrics and Principal Component Analysis (PCA). Results show improved classification accuracy and efficiency with the right distance metric and PCA.The authors in [22] focused on detecting Android malware in APK files by analyzing obfuscation techniques, permissions, and API calls. They highlighted the challenges faced by traditional antivirus software in detecting these malware variants.

Amenova et al [23] proposed a CNN-LSTM deep learning approach for Android malware detection, achieving high accuracy through efficient feature extraction. Mantoro et. al [24] employed dynamic analysis using the Mobile Security Framework to detect obfuscated malware. It showcases the effectiveness of dynamic analysis in detecting various types of malware.

The authors in [25] compared state-of-the-art mobile malware detection methods, addressing android malwares and various detection classifiers. It provided insights into the progress of the android platform and offers a clear understanding of the advancements in malware detection.

The authors in [26] proposed a framework, FAMD, for fast Android malware detection based on a combination of multiple features. The original feature set is constructed by extracting permissions and Dalvik opcode sequences from samples. The dimensionality-reduced features are then input into the CatBoost classifier for malware detection and family classification.

Awais et. al [27] introduced ANTI-ANT,a unique framework that detects and prevents malware on mobile devices. It used three detection layers, static and dynamic analysis, and multiple classifiers.Islam et. al [28] investigated the effectiveness of unigram, bigram, and trigram with stacked generalization and finds that unigram has the highest detection rate with over 97 percent accuracy compared to bigram and trigram. Overall, this study established a strong foundation for using n-gram techniques in developing android malware detection.

The authors in [29] proposed a fitting factor technique to identify duplicate malicious files in the Drebin datasets based on opcode occurrence. They found that 51.57 percent of malicious samples had duplicates. We evaluated popular detection models with and without duplicates, using all features and the top 26 features engineered by IG and AE techniques. Goyal et al.[30] proposed SafeDroid which is a lightweight, open-source distributed service for detecting malicious Android apps with static features and machine learning through three micro-services, providing user-friendly feedback upon malware detection.

The authors in [31] proposed Android malware installation by introducing robust and lightweight classifiers. Relevant malware behavior features are extracted through API-level analysis, and various classifiers are evaluated with the feature set. The study demonstrated the effectiveness of the proposed classifiers in mitigating Android malware installation.

The authors in [32] presented two novel deep learning-based methods for end-to-end Android malware detection. The approach included raw bytecode resampling from classes.dex files as input enhancing detection effectiveness and efficiency.

Kong et al.[33] proposed FCSCNN, a unique approach for Android malware detection by calculating mean centers of benign and malicious samples from a database, and measuring distances to determine class. The authors in [34] described a novel detection framework called PermPair, which utilized permission pairs extracted from an application's manifest file to construct and compare graphs for both malware and normal samples .The proposed method was innovative and aims to identify potential threats efficiently.

Mathur et al. [35] proposed a framework NATICUSdroid for detecting malware in Android devices with machine learning classifiers by examining the local and specialized Android permissions. The feature selection methods were used to determine the effectiveness of eight machine learning algorithms in detecting malware.

In [36], authors proposed a system for detecting mobile malware on Android devices using multi-criteria decision-making and fuzzy analytical hierarchy process. The system utilizes static analysis techniques, specifically permission-based features, to identify malicious activity.The authors in [37] introduced EveDroid, an Android malware detection system that used behavioral patterns in different events to effectively detect new malware. It used event

groups to describe app behaviors in the event level, which captures higher levels of semantics.

Srivastava et al. [38] developed a malware detector and analyzer with a focus on understanding malware attacks on mobile devices during the COVID-19 pandemic. They implemented a model that extracts intrinsic features from Android application files for quick and accurate analysis, resulting in improved app classification accuracy as benign or malicious.

The authors in [39] compareed existing methods for detecting Android malwares and provides a concise overview of the progress of the Android platform. Emphasis was given to various techniques for detecting Android malwares and presenting the current state of malware detection classifiers.Aktas et al.[40] proposed a dynamic analysis-based method for detecting malicious applications by extracting features from runtime behavior.Using machine learning techniques,they developed a detection system using these features to distinguish between malicious and benign applications.

The authors in [41] introduced contrasting permission patterns as a means to differentiate between malware and clean applications based on their permission usage. A framework is presented for Android malware detection that utilizes these contrasting permission patterns.

Xi et al. [42] proposed a new method for detecting malicious Android applications using a feature vector extracted from the AndroidManifest file, combining permission and component information, and leveraging the naive Bias classification algorithm. Garcia et al.[43] proposed RevealDroid,a machine learning-based approach for Android malware detection and family identification, utilizing categorized API usage, reflection-based features, and native binary features, without the need for complex program analyses or extensive feature extraction.

The authors in [44] discussed the dendritic cell algorithm (DCA) to detect anomalies in the behaviors of 100 Android applications based on logged system calls. The DCA features are then utilized for classifying the applications as benign or malicious, providing a concise approach for Android malware detection.The authors in [45] proposed a novel approach for Android malware detection using gray scale image visualization and GIST descriptor for

feature extraction. Three classifiers (KNN, RF, and DT) were utilized for detection and comparison, offering a concise and innovative approach.

Iqbal et al [46] proposed SpyDroid which is a real-time malware detection framework that can incorporate third-party detectors and facilitates efficient and controlled monitoring. It includes monitoring and detection modules, and application layer sub-detectors are supported. The detection module determines when to flag an app as malicious.

The authors in [47] presented an Android malware detection approach using the XGBoost model and evaluated the impact of feature selection on classification. Results showed high effectiveness and accuracy, comparable to SVM but with less time consumption.The authors in [48] presented DroidGraph which uses hierarchical behavior graphs with 136 identical nodes, representing the semantics of Android API calls in APK files.

The authors in [49] defined Droid-NNet, a deep learning framework for Android malware classification, outperforming existing methods on Malgenome-215 and Drebin-215 datasets. The authors in [50] introduced a permission-based malware detection system and re-implement Juxtapp for malware and piracy detection. Performance is evaluated on a dataset with original, pirated, and malware-infected applications.

The authors in [51] introduced DynaMalDroid,a dynamic analysis-based framework for detecting malicious Android apps. It employs system call extraction and three modules: dynamic analysis, feature engineering, and detection.

CHAPTER 3 PROPOSED METHODOLOGY

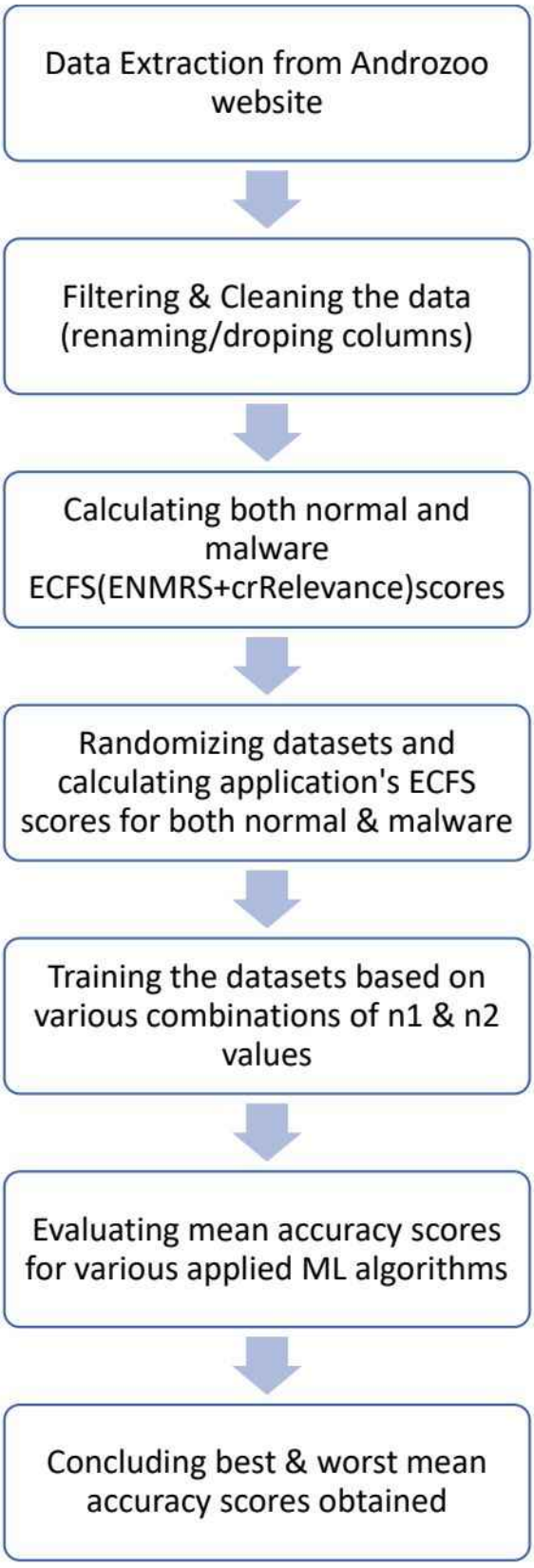
We explain the system design in various sub-phases described below.

Our study involved the use of two datasets, one comprising normal apps, and the other containing malicious apps. The dataset for normal apps was collected from the Google Play Store, while the dataset for malicious apps was obtained from the AndroZoo. In March 2023, we collected data on the apps available in the Google Play Store, which amounted to a total of 2,673,292 apps.

It is important to note that our study solely focused on apps in the Google Play Store and did not consider apps available on other platforms. The Google Play Store was chosen due to its popularity and the accessibility of its data, rather than being an accurate representation of all available apps across all devices.

The AndroZoo website is a growing library of Android apps collected from various sources, including the official Google Play app market. It also contains a collection of app-related metadata aimed at facilitating research on Android devices. The library currently contains 15,097,876 unique APKs, which have been scanned by multiple antivirus programs to identify malicious software. Each software in the dataset has over 20 different types of metadata, including VirusTotal reports. Our dataset consisted of 111,010 applications, with 55,505 labeled as malicious and the remaining 55,505 labeled as normal.

3.1 FLOW OF PROCEDURE



3.2 Feature Extraction

Feature extraction is a vital process in Android malware analysis as it helps in identifying the characteristics of malware and distinguishing it from benign applications. Android permissions are commonly used as features for building a probable model for Android malware analysis.

Permissions related to the network, reading privacy, receiving and sending SMS, dialing, and others are considered dangerous permissions and are used to distinguish between malicious and benign applications. Hence, we have selected permissions as the feature for experiments in this proposed work.

Android permission extraction is a crucial process used to detect potential malware by extracting and analyzing permissions from Android apps. There are two main techniques for extracting permissions: static analysis and dynamic analysis.

We followed a static approach to extract permissions that involves decompiling the app's APK file using tools such as Apktool, JADX, or Androguard to extract the manifest file, which contains details about the app's rights. The permission declarations are then extracted from the manifest file using XML parsing libraries.These extracted permissions can be further analyzed by comparing them to a predefined list of known dangerous permissions or by looking for anomalous or excessive permissions that an app seeks beyond the scope of its authorized functionality.

In contrast, dynamic analysis involves running the app on a device or emulator and observing its behavior during runtime using tools like DroidBox, TaintDroid, or MobSF. During runtime, dynamic analysis tools record the permissions that the app seeks, either through the uses-permission> tag in the manifest file or through runtime permission requests made using the Android Runtime Permissions system.

We had a total of 129 unique permissions from both datasets.

3.3 Feature-Feature Correlation with ENMRS

In order to evaluate the ECFS scores in our dataset, we used the Effective Normalized Mean Residue Similarity (ENMRS) measure, which is an extension of the Normalized Mean Residue Similarity (NMRS) measure.

While Pearson's correlation coefficient is another widely used correlation measure, we opted for NMRS due to its strict confinement to detecting shifting correlation only, rather than scaling correlation. However, both NMRS and Pearson's correlation coefficient are highly sensitive to outlier or noisy values, which can cause important features to be dropped from the optimal feature subset. To address this limitation, we replaced the object mean with object local means in ENMRS, computed by taking the average of the element and its left and right neighbors. This characteristic is highly required in feature-feature correlation analysis due to the availability of correlation over a subset of homogeneous objects. In our case, we used a single left and right neighbor and decided to use the single neighborhood scheme in our local mean computation. ENMRS between a pair of objects $d1 = [a_1, a_2, ..., a_n]$ and $d2 = [b_1, b_2, ..., b_n]$ can be defined as follows.

$$ENMRS(d_1, d_2) = 1 - \frac{\sum_{i=1}^{n} |a_i - a_{lmean(i)} - b_i + b_{lmean(i)}|}{2 \times \max\left(\sum_{i=1}^{n} |(a_i - a_{lmean(i)})|, \sum_{i=1}^{n} |b_i - b_{lmean(i)}|\right)}$$

where,

$$a_{Imean(\mathrm{i})} = (a_{i-1} + a_i + a_{i+1})/3 \quad \text{if, } 1 < i < n,$$

$$a_{Imean(\mathrm{i})} = (a_i + a_{i+1})/2 \quad \text{if, } i = 1,$$

$$a_{Imean(\mathrm{i})} = (a_{i-1} + a_i)/2 \quad \text{if, } i = n.$$

3.4 Feature-class correlation measure: crRelevance

The crRelevance measure evaluates the ability of a feature to distinguish various class labels, in our case two, i.e., Malware and Normal, and returns a value in the range of [0, 1]. A class range is defined as a range for a feature, in which all objects have the same class label. The class range can be determined by assigning a range of consecutive values for a feature that has the same class label.

The crRelevance measure is based on four definitions that provide the theoretical basis of crRelevance. The first definition is for a feature with values corresponding to n objects or instances in the dataset, a class range can be defined as a range such that all objects in this range have the same class label. The second definition states the cardinality of a class range as the cardinality of the set of all objects for the feature in the given range. The third definition defines the class-cardinality of class A as the cardinality of the set of all objects having the class label A. The fourth definition is the core class range of class A, which is the highest class range for class A.

$crRelevance^{\text{class}}_{\text{f}_\text{i}}(\text{A})$ is defined as follows

$$crRelevance^{\text{class}}_{\text{f}_\text{i}}(\text{A}) = \frac{rcard\big(ccrange(\text{A})\big)}{ccard(\text{A})}$$

For dataset D, the core class relevance of a feature $f_i \in \text{F}$ can be defined as the highest crRelevance for a given class A_i. Mathematically, crRelevance of a feature f_i, $\text{crRelevance}(\text{f}_\text{i})$, for a dataset with n classes $A_1, A_2, \dots, A_n$ can be defined as follows.

$$crRelevance(f_i) = \max_{1 \le j \le n} crRelevance f^{\text{class}}_{f_i}\left(A_j\right)$$

3.5 Proposed feature selection technique: ECFS

The proposed feature selection method uses ENMRS and crRelevance to compute an ECFS value for each pair of features, which ranges from 0 to 1. The method ensures that a high ECFS value corresponds to a high crRelevance score (or featureclass correlation) and a low ENMRS score (or feature-feature correlation) by subtracting the ENMRS value for the pair from 1 and adding it to the average crRelevance score. The constants n_1 and n_2 are multiplied with computed feature-feature and feature-class components to bring the range [0,2] to [0,1] and control their contribution to the ECFS score. The method selects a user-defined number of features by iteratively choosing the next highest unprocessed feature pair with at least one common feature and including the common feature(s) in the selected subset. The selected feature subset is presented as an output. ENMRS between the pair of features is directly computed, while crRelevance of the individual feature is averaged to obtain the crRelevance value of the pair of features. To ensure that a high ECFS value

(which ranges from 0 to 1) corresponds to a high crRelevance score (or feature-class correlation) and low ENMRS score (or feature-feature correlation), we subtract the ENMRS value for the pair from 1 and add it to the average Table 10 Sizes of optimal feature subsets. crRelevance score is used to obtain the final ECFS value. ECFS value of a pair of features f_1, f_2 can be computed as follows.

$$ECFS(f_1, f_2) = \left(n_1 \times \left(1 - ENMRS(f_1, f_2)\right)\right) + \left(n_2 \times avgRelevance(f_1, f_2)\right)$$

where,

n_1 and n_2 are constants such that $n_1 + n_2 = 1$, and

$$avgRelevance(f_1, f_2) = \frac{crRelevance(f_1) + crRelevance(f_2)}{2}$$

The constants $n1$ and $n2$ are multiplied with computed feature-feature and feature-class components in the equation to bring the range [0,2] to [0,1] and to control the contribution of feature-feature and feature-class correlations on the ECFS score. These constants are set such that $n_1 + n_2 = 1$. Setting $n_1 = n_2 = 0.5$ will lead to the equal contribution of these components. Setting the value $n_1 > 0.5$ will lead to a result that will have more contributions from ENMRS,

i.e., feature-feature correlation while setting the value $n_2 > 0.5$ will lead to a result with contribution from crRelevance or feature class correlation.

3.6 Machine Learning Techniques Used

We used the following Machine Learning Techniques to evaluate the efficiency of the ECFS scores for different values of n1 & n2.

- Decision Tree: A decision tree is a supervised machine learning algorithm that is used for both classification and regression tasks. It is a graphical representation of a flowchart-like structure where each internal node represents a test on a feature attribute, each branch

represents the outcome of the test, and each leaf node represents the final decision or the predicted outcome.

The decision tree algorithm starts with the entire dataset at the root node and recursively splits the data based on the feature values to create a tree-like structure. The splitting is done in a way that maximizes the information gain or minimizes the impurity measure at each node.

The information gain is a measure of the reduction in uncertainty after the split. It calculates the difference between the impurity of the parent node and the weighted impurity of the child nodes. The impurity measures commonly used in decision trees include Gini impurity and entropy.

In a classification task, each leaf node represents a class label, and the majority class in the leaf node is assigned to instances that reach that node. For regression tasks, the leaf nodes contain the predicted numerical values.

The decision tree algorithm is advantageous due to its simplicity and interpretability. It can handle both categorical and numerical features, as well as missing values. Decision trees can also capture non-linear relationships between features. However, decision trees are prone to overfitting, especially when the tree becomes too complex. This issue can be mitigated by techniques such as pruning, setting a maximum depth for the tree, or using ensemble methods like random forests.

Decision trees have numerous applications in various domains, including finance, healthcare, marketing, and customer relationship management. They are widely used for tasks such as credit scoring, fraud detection, disease diagnosis, and recommendation systems.

- Support Vector Machine: A Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks. It is a powerful and versatile algorithm that can effectively handle complex datasets.

  The goal of an SVM is to find the best hyperplane that separates the data points of different classes in the feature space. A hyperplane is a decision boundary that maximizes the margin or the distance between the nearest data points of different classes. The data points closest to the hyperplane are called support vectors, hence the name "Support Vector Machine."

The SVM algorithm can handle linearly separable data by finding a linear hyperplane. However, it can also handle non-linearly separable data by using kernel functions. Kernel functions transform the original feature space into a higher-dimensional space where the data points can be linearly separated. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

In the case of classification, once the hyperplane is determined, new data points can be classified based on which side of the hyperplane they fall on. SVM aims to maximize the margin between the classes, which leads to better generalization and improved performance on unseen data.

For regression tasks, SVM aims to find a hyperplane that best fits the data points, while considering a certain tolerance or error margin. The predicted value for a new data point is determined based on its position relative to the hyperplane.

- **Logistic Regression**: Logistic regression is a statistical model used for binary classification problems, where the goal is to predict a binary outcome or assign an observation to one of two classes. It is a popular and widely used algorithm in machine learning and statistics.

  The key idea behind logistic regression is to model the relationship between a set of independent variables (also known as features or predictors) and the probability of an event occurring. The dependent variable, or the outcome variable, is binary and takes one of two values (e.g., 0 or 1, true or false, yes or no). The logistic regression model estimates the probability that an observation belongs to the positive class (1) given its feature values.

  The logistic regression model uses the logistic function (also known as the sigmoid function) to map the output of a linear combination of the features to a value between 0 and 1. The logistic function has an S-shaped curve that smoothly transitions between the two extremes, representing the probability of the positive class.

- **Random Forest**: Random Forest is a popular machine learning algorithm used for both classification and regression tasks. It belongs to the ensemble learning methods, which combine multiple individual models to make predictions. Random Forest is known for its effectiveness, versatility, and robustness. Random Forest has several advantages. Firstly, it can handle high-dimensional data and large feature sets effectively. It can

capture complex relationships between features and the target variable and handle non-linear decision boundaries. Secondly, Random Forest is less prone to overfitting compared to a single decision tree due to the ensemble of trees and the random feature selection. Moreover, it can handle missing values and maintain good performance even with noisy or irrelevant features. Random Forests are widely used in various applications such as classification, regression, feature importance ranking, and anomaly detection. They are known for their robustness, scalability, and ability to provide insights into feature importance.

- **K-Nearest Neighbor Classifier (KNN)**: K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for both classification and regression tasks. It is a non-parametric algorithm that makes predictions based on the similarity of a new observation to its k nearest neighbors in the training data.

  KNN has several notable characteristics. It is a lazy learning algorithm, meaning it does not explicitly build a model during the training phase. Instead, it stores the entire training dataset and performs computations at the time of making predictions. KNN also assumes that nearby points in the feature space have similar target values. Therefore, it can work well when there is a local structure or clustering in the data.

  One of the advantages of KNN is its simplicity and ease of implementation. It can handle both numerical and categorical data, and it can adapt to changes in the training data without the need for retraining the model. However, KNN can be computationally expensive, especially for large datasets, as it requires calculating distances for each prediction. Additionally, choosing the right value of k and selecting appropriate distance metrics are crucial for obtaining good results with KNN.

- **Gaussian Naive Bayes**: Gaussian Naive Bayes is a supervised machine learning algorithm based on the principles of Bayes' theorem. It is primarily used for classification tasks and is particularly suited for handling continuous numerical features.

  Gaussian Naive Bayes has several advantages. It is computationally efficient and performs well even with a small amount of training data. It can handle both numerical and continuous features, making it suitable for a wide range of applications. Additionally, it provides interpretable results by estimating the class probabilities and allowing the examination of feature importance. However, Gaussian Naive Bayes has limitations. The

assumption of feature independence may not hold in certain scenarios, leading to suboptimal results. It also assumes a Gaussian distribution for the features, which may not be appropriate for all types of data. In situations where feature interactions or non-Gaussian distributions are significant, other algorithms such as decision trees or support vector machines may be more appropriate.

- **Perceptron**: The perceptron is a basic building block of artificial neural networks and is one of the oldest and simplest types of artificial neurons. It is a binary classification algorithm used for supervised learning tasks. The perceptron algorithm is suitable for linearly separable datasets, where a single straight line can separate the two classes. If the dataset is not linearly separable, the perceptron may not converge or produce accurate results. In such cases, more advanced neural network architectures, such as multi-layer perceptrons (MLPs) or deep neural networks, can be employed. Perceptrons have historically played a significant role in the development of neural networks and machine learning. Although they are relatively simple compared to modern neural network architectures, they form the foundation for understanding more complex algorithms.

- **SGD Classifier**: SGD Classifier is a fast and scalable algorithm that can handle large datasets with high-dimensional features. It is particularly useful for online learning and can adapt to changes in the data. In conclusion, each of these machine learning algorithms has its strengths and weaknesses, and the choice of algorithm depends on the specific problem and the characteristics of the data. It is important to understand the underlying assumptions, limitations, and trade-offs of each algorithm before applying it to real-world problems.

CHAPTER 4 RESULTS AND DISCUSSION

4.1 RESULTS

As discussed in the above section, we can apply different values of n1 and n2 with a constraint that their sum leads up to one. Hence, we have used different combinations of n1 and n2 and we have summarized the results from each of the combinations in the sub-sections described below.

1. n1=0.1 and n2=0.9

From Table I we conclude that for n1 =0.1 and n2=0.9, the highest accuracy i.e 89.21% is obtained with Random Forest and the lowest accuracy i.e 70.16 % is obtained by Perceptron. As n1 > n2, the accuracy scores are more inclined towards the crRelevance value of the ECFS score.

Table 1 ML Accuracy Results for n1=0.1 and n2=0.9

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy(%)
0.1	0.9	Decision Tree	0.8762 0.8744 0.8675 0.8819 0.8788 0.8650 0.8788 0.8625 0.8681 0.8681	87.21
0.1	0.9	SVM	0.8100 0.8197 0.8125 0.8184 0.8150	81.51
0.1	0.9	Logistic Regression	0.7881 0.8038 0.8125 0.7994 0.8019 0.7838 0.8000 0.8038 0.7925 0.8031	79.88
0.1	0.9	Random Forest	0.8956 0.8888 0.8988 0.8919 0.8938 0.8806 0.9038 0.8844 0.8888 0.8950	89.21
0.1	0.9	KNeighbors Classifier	0.8694 0.8669 0.8669 0.8675 0.8744 0.8575 0.8681 0.8631 0.8706 0.8738	86.78
0.1	0.9	Gaussian NB	0.6844 0.7006 0.7013 0.7044 0.6913 0.6819 0.7075 0.7131 0.7006 0.7038	69.88
0.1	0.9	Perceptron	0.7394 0.7400 0.7369 0.7463 0.7356 0.7713 0.5388 0.7550 0.5031 0.7500	70.16
0.1	0.9	SGD Classifier	0.7881 0.8044 0.8131 0.7963 0.8044 0.7825 0.8006 0.8019 0.7944 0.8056	79.91

2. n1=0.2 and n2=0.8

From Table II we conclude that for n1 =0.2 and n2=0.8, the highest accuracy i.e 87.90% is obtained with Random Forest and the lowest accuracy i.e 68.08 % is obtained by Perceptron. As n1 > n2, the accuracy scores are more inclined towards the crRelevance value of the ECFS score.

Table 2 ML Accuracy Results for n1=0.2 and n2=0.8

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.2	0.8	Decision tree	0.8581 0.8638 0.8713 0.8581 0.8650 0.8613 0.8463 0.8606 0.8488 0.8738	86.07
0.2	0.8	SVM	0.7778 0.7922 0.7850 0.7859 0.7900	78.62
0.2	0.8	Logistic Regression	0.7694 0.7581 0.7894 0.7744 0.7844 0.7688 0.7594 0.7881 0.7544 0.7931	77.39
0.2	0.8	Random Forest	0.8831 0.8856 0.8925 0.8694 0.8769 0.8769 0.8669 0.8769 0.8713 0.8906	87.90
0.2	0.8	KNeighbors Classifier	0.8563 0.8631 0.8756 0.8488 0.8563 0.8400 0.8463 0.8588 0.8394 0.8738	85.58
0.2	0.8	Gaussian NB	0.7094 0.6919 0.7169 0.6931 0.7169 0.6944 0.6950 0.7131 0.6938 0.7256	70.50
0.2	0.8	Perceptron	0.6275 0.5019 0.5275 0.6813 0.7394 0.7325 0.7269 0.7625 0.7313 0.7775	68.08
0.2	0.8	SGD Classifier	0.7713 0.7619 0.7888 0.7681 0.7844 0.7813 0.7575 0.7731 0.7519 0.7944	77.33

3. n1=0.3 and n2= 0.7

From Table III we conclude that for n1 =0.3 and n2=0.7, the highest accuracy i.e 88.54% is obtained by Random Forest and the lowest accuracy i.e 70.91 % is obtained by Gaussian NB. As n1 > n2 , the accuracy scores are more inclined towards the crRelevance value of the ECFS score.

Table 3 ML Accuracy Results for n1=0.3 and n2=0.7

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.3	0.7	Decision tree	0.8656 0.8662 0.8563 0.8613 0.8631 0.8613 0.8606 0.8500 0.8456 0.8544	85.84
0.3	0.7	SVM	0.8016 0.8050 0.8000 0.7991 0.7938	79.99
0.3	0.7	Logistic Regression	0.7594 0.7813 0.7806 0.7825 0.7688 0.7769 0.7894 0.7625 0.7813 0.7725	77.55
0.3	0.7	Random Forest	0.8831 0.8894 0.9013 0.8881 0.8919 0.8838 0.8819 0.8869 0.8731 0.8750	88.54
0.3	0.7	KNeighbors Classifier	0.8588 0.8662 0.8713 0.8538 0.8656 0.8588 0.8613 0.8588 0.8506 0.8494	85.94
0.3	0.7	Gaussian NB	0.6938 0.7106 0.7344 0.7150 0.7019 0.6994 0.7213 0.7069 0.7019 0.7063	70.91
0.3	0.7	Perceptron	0.7406 0.7863 0.7656 0.6881 0.7688 0.7531 0.7731 0.7525 0.6613 0.7388	74.28
0.3	0.7	SGD Classifier	0.7669 0.8019 0.7925 0.7938 0.7550 0.7925 0.7925 0.7725 0.7850 0.7863	78.39

4. n1=0.4 and n2=0.6

From Table IV we conclude that for n1 =0.4 and n2=0.6, the highest accuracy i.e 89.91% is obtained by Random Forest and the lowest accuracy i.e 62.14 % is obtained by Perceptron. As n1 > n2 , the accuracy scores are more inclined towards the crRelevance value of the ECFS score.

Table 4 ML Accuracy Results for n1=0.4 and n2=0.6

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.4	0.6	Decision tree	0.8856 0.8681 0.8788 0.8762 0.8738 0.8712 0.8819 0.88 0.8831 0.8725	87.71
0.4	0.6	SVM	0.8281 0.8281 0.8203 0.8153 0.8278	82.39
0.4	0.6	Logistic Regression	0.7962 0.7938 0.7844 0.7875 0.7831 0.7831 0.8006 0.775 0.7925 0.7863	78.82
0.4	0.6	Random Forest	0.905 0.8994 0.9062 0.8988 0.8962 0.8906 0.8994 0.8981 0.9025 0.8944	89.91
0.4	0.6	KNeighbors Classifier	0.8731 0.8813 0.88 0.8688 0.8681 0.8763 0.8869 0.865 0.8656 0.8688	87.34
0.4	0.6	Gaussian NB	0.7219 0.7113 0.7056 0.7238 0.7106 0.7256 0.7375 0.7088 0.725 0.7188	71.89
0.4	0.6	Perceptron	0.595 0.78 0.7625 0.7644 0.4369 0 .6 0.4988 0.4981 0.7794 0.4988	62.14
0.4	0.6	SGD Classifier	0.8044 0.7956 0.7894 0.7663 0.7856 0.7913 0.8113 0.7863 0.7956 0.8119	79.38

5. n1=0.5 and n2=0.5

From Table V we conclude that for n1 =0.5 and n2=0.5, the highest accuracy i.e 87.80% is obtained by the Random Forest and the lowest accuracy i.e 71.79 % is obtained by Gaussian NB. As n1 = n2, the accuracy scores are balanced towards the crRelevance and ENMRS values of the ECFS score.

Table 5 ML Accuracy Results for n1=0.5 and n2=0.5

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.5	0.5	Decision Tree	0.8556 0.8431 0.8506 0.8437 0.8594 0.8681 0.8513 0.8475 0.8650 0.8363	85.21
0.5	0.5	SVM	0.7981 0.7903 0.7984 0.8034 0.7934	79.68
0.5	0.5	Logistic Regression	0.7812 0.7750 0.7750 0.7700 0.7681 0.7800 0.8006 0.7644 0.7669 0.7725	77.54
0.5	0.5	Random Forest	0.8788 0.8775 0.8713 0.8575 0.8825 0.8931 0.8913 0.8781 0.8856 0.8644	87.80
0.5	0.5	KNeighbors Classifier	0.8306 0.8431 0.8263 0.8194 0.8419 0.8419 0.8513 0.8250 0.8506 0.8306	83.61
0.5	0.5	Gaussian NB	0.7188 0.7188 0.7250 0.7138 0.7194 0.7156 0.7425 0.7019 0.7131 0.7100	71.79
0.5	0.5	Perceptron	0.7325 0.5006 0.7594 0.7775 0.7719 0.7844 0.7888 0.7550 0.7663 0.7581	73.94
0.5	0.5	SGD Classifier	0.7863 0.7863 0.7719 0.7688 0.7675 0.7856 0.8044 0.7669 0.7694 0.7788	77.86

6. n1=0.6 and n2=0.4

From Table VI we conclude that for n1 =0.6 and n2=0.4, the highest accuracy i.e 90.96% is obtained by Random forest and the lowest accuracy i.e 72.19 % is obtained by Gaussian NB. As n1 < n2 , the accuracy scores are more inclined towards the ENMRS value of the ECFS score.

Table 6 ML Accuracy Results for n1=0.6 and n2=0.4

Ml Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.6	0.4	Decision Tree	0.8875 0.9 0.9075 0.8919 0.8819 0.8938 0.88 0.8831 0.89 0.8938	89.09
0.6	0.4	SVM	0.8528 0.8519 0.8422 0.8519 0.8553	85.08
0.6	0.4	Logistic Regression	0.8256 0.8013 0.8238 0.8125 0.805 0.7994 0.8069 0.8113 0.81 0.795	80.91
0.6	0.4	Random Forest	0.9113 0.91 0.9231 0.915 0.8969 0.9063 0.9013 0.9069 0.9125 0.9125	90.96
0.6	0.4	KNeighbors Classifier	0.8925 0.8888 0.885 0.8931 0.8806 0.8894 0.8875 0.8794 0.8806 0.8819	88.59
0.6	0.4	Gaussian NB	0.7481 0.7013 0.7281 0.7081 0.7194 0.7081 0.745 0.7175 0.7275 0.7156	72.19
0.6	0.4	Perceptron	0.8331 0.5775 0.8156 0.81 0.7781 0.7675 0.8056 0.7763 0.7938 0.77	77.28
0.6	0.4	SGD Classifier	0.83 0.8069 0.8138 0.8169 0.7919 0.8075 0.8125 0.8138 0.8238 0.7906	81.08

7. n1=0.7 and n2=0.3

From Table VII we conclude that for n1 =0.7 and n2=0.3, the highest accuracy i.e 91.64% is obtained by Random forest and the lowest accuracy i.e 72.64 % is obtained by Gaussian NB. As n1 < n2, the accuracy scores are more inclined towards the ENMRS value of the ECFS score.

Table 7 ML Accuracy Results for n1=0.7 and n2=0.3

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.7	0.3	Decision tree	0.9000 0.8981 0.8894 0.8894 0.9069 0.8969 0.8881 0.8856 0.9081 0.8875	89.50
0.7	0.3	SVM	0.8538 0.8547 0.8634 0.8588 0.8638	85.89
0.7	0.3	Logistic Regression	0.8138 0.7944 0.8038 0.8244 0.8269 0.8213 0.8206 0.8000 0.8294 0.8138	81.48
0.7	0.3	Random Forest	0.9150 0.9175 0.9038 0.9113 0.9294 0.9275 0.9144 0.9056 0.9219 0.9175	91.64
0.7	0.3	KNeighbors Classifier	0.8894 0.8925 0.8813 0.8781 0.9025 0.9019 0.9025 0.8875 0.9006 0.8938	89.30
0.7	0.3	Gaussian NB	0.7244 0.7063 0.7250 0.7219 0.7413 0.7331 0.7263 0.7125 0.7375 0.7363	72.64
0.7	0.3	Perceptron	0.7425 0.7756 0.7619 0.5519 0.7975 0.8500 0.7981 0.8319 0.8656 0.5463	75.21
0.7	0.3	SGD Classifier	0.8056 0.7975 0.8088 0.8281 0.8263 0.8288 0.8250 0.7994 0.8300 0.8106	81.60

8. n1=0.8 and n2=0.2

From Table VIII we conclude that for n1 =0.8 and n2=0.2, the highest accuracy i.e 91.96% is obtained by Random Forest and the lowest accuracy i.e 70.76 % is obtained by Perceptron. As n1 < n2, the accuracy scores are more inclined towards the ENMRS value of the ECFS score.

Table 8 ML Accuracy Results for n1=0.8 and n2=0.2

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.8	0.2	Decision*tree*	0.8994 0.9156 0.8931 0.8912 0.9019 0.8969 0.9094 0.8950 0.9019 0.8931	89.975
0.8	0.2	SVM	0.8628 0.8641 0.8663 0.8741 0.8713	86.7688
0.8	0.2	Logistic Regression	0.8088 0.8356 0.8244 0.8163 0.8294 0.8281 0.8250 0.8269 0.8319 0.8163	82.4250
0.8	0.2	Random Forest	0.9169 0.9150 0.9169 0.9163 0.9288 0.9194 0.9244 0.9163 0.9225 0.9200	91.9625
0.8	0.2	KNeighbors classifier	0.8938 0.8963 0.9000 0.8925 0.8988 0.8931 0.9100 0.8956 0.9075 0.8975	89.8500
0.8	0.2	Gaussian NB	0.7150 0.7363 0.7500 0.7219 0.7494 0.7425 0.7419 0.7363 0.7469 0.7306	73.7063
0.8	0.2	Perceptron	0.5588 0.8050 0.8113 0.7213 0.7375 0.7175 0.7988 0.8013 0.7281 0.3969	70.7625
0.8	0.2	SGD Classifier	0.8081 0.8381 0.8100 0.8100 0.8363 0.8256 0.8238 0.8219 0.8306 0.8156	82.2000

9. n1=0.9 and n2=0.1

From Table IX we conclude that for n1 =0.9 and n2=0.1, the highest accuracy i.e 92.25% is obtained by Random Forest and the lowest accuracy i.e 71.20 % is obtained by Perceptron. As n1 < n2, the accuracy scores are more inclined towards the ENMRS value of the ECFS score.

Table 9 ML Accuracy Results for n1=0.9 and n2=0.1

ML Accuracy				
n1	n2	ML Model	Accuracy Scores	Accuracy (%)
0.9	0.1	Decision tree	0.9013 0.9056 0.9000 0.9100 0.9063 0.9069 0.9094 0.9019 0.8913 0.8969	90.2938
0.9	0.1	SVM	0.8831 0.8797 0.8747 0.8813 0.8559	87.4938
0.9	0.1	Logistic Regression	0.8188 0.8319 0.8263 0.8369 0.8269 0.8344 0.8313 0.8256 0.8200 0.8188	82.7063
0.9	0.1	Random Forest	0.9113 0.9256 0.9256 0.9225 0.9319 0.9325 0.9256 0.9206 0.9194 0.9100	92.2500
0.9	0.1	KNeighbors Classifier	0.8925 0.9000 0.9156 0.9094 0.9063 0.9188 0.9019 0.9094 0.9088 0.8913	90.5375
0.9	0.1	Gaussian NB	0.7388 0.7344 0.7381 0.7538 0.7381 0.7413 0.7450 0.7281 0.7388 0.7313	73.8750
0.9	0.1	Perceptron	0.8500 0.7988 0.8388 0.5619 0.6325 0.8544 0.6206 0.8506 0.5619 0.5506	71.2000
0.9	0.1	SGD Classifier	0.8163 0.8256 0.8250 0.8363 0.8200 0.8213 0.8294 0.8219 0.8175 0.8131	82.2625

4.2 CONCLUSION

From Table X we conclude our thesis by evaluating that the highest accuracy i.e 92.25 % was achieved by Random Forest ML technique for the values of n1=0.9 and n2=0.1.

As n1 > n2, our accuracy results were more inclined towards the ENMRS values of the ECFS scores. Also in Table X, we interpret the pattern as for higher n1 values i.e for the feature-feature correlation (ENMRS) factor, the accuracy increases. And the accuracy decreases for higher n2 values meaning for feature-class correlation (crRelevance) scores. Thus, for higher n1 value and lower n2 value of the ECFS score the ML techniques have better accuracy. The preliminary results based on the work we done were satisfactory but need more evaluation on evaluating for various values of n1 and n2 in future work.

Table 10 ML Accuracy for different value of n1 & n2

Highest Accuracy									
n1 & n2	0.1 & 0.9	0.2 & 0.8	0.3 & 0.7	0.4 & 0.6	0.5 & 0.5	0.6 & 0.4	0.7 & 0.3	0.8 & 0.2	0.9 & 0.1
Accuracy (%)	89.21	87.89	88.54	89.90	87.80	90.95	91.63	91.96	92.25

REFERENCES

1. https://androzoo.uni.lu
2. https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/
3. E. Amer, "Permission-Based Approach for Android Malware Analysis Through Ensemble-Based Voting Model," 2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), Cairo, Egypt, 2021, pp. 135-139, doi: 10.1109/MIUCC52538.2021.9447675.
4. M. İbrahim, B. Issa and M. B. Jasser, "A Method for Automatic Android Malware Detection Based on Static Analysis and Deep Learning," in IEEE Access, vol. 10, pp. 117334-117352, 2022, doi: 10.1109/ACCESS.2022.3219047.
5. C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang and H. Kinawi, "Android Malware Detection Based on Factorization Machine," in IEEE Access, vol. 7, pp. 184008-184019, 2019, doi: 10.1109/ACCESS.2019.2958927.
6. J. Qiu et al., "Cyber Code Intelligence for Android Malware Detection," in IEEE Transactions on Cybernetics, vol. 53, no. 1, pp. 617-627, Jan. 2023, doi: 10.1109/TCYB.2022.3164625.
7. U. Haq, T. A. Khan and A. Akhunzada, "A Dynamic Robust DL-Based Model for Android Malware Detection," in IEEE Access, vol. 9, pp. 74510-74521, 2021, doi: 10.1109/ACCESS.2021.3079370.
8. J. Qiu et al., "A3CM: Automatic Capability Annotation for Android Malware," in IEEE Access, vol. 7, pp. 147156-147168, 2019, doi: 10.1109/ACCESS.2019.2946392.
9. M. M. Alani and A. I. Awad, "PAIRED: An Explainable Lightweight Android Malware Detection System," in IEEE Access, vol. 10, pp. 73214-73228, 2022, doi: 10.1109/ACCESS.2022.3189645.
10. S. Khalid and F. B. Hussain, "Evaluating Dynamic Analysis Features for Android Malware Categorization," 2022 International Wireless Communications and Mobile Computing (IWCMC), Dubrovnik, Croatia, 2022, pp. 401-406, doi: 10.1109/IWCMC55113.2022.9824225.
11. S. Seneviratne, R. Shariffdeen, S. Rasnayaka and N. Kasthuriarachchi, "Self-Supervised Vision Transformers for Malware Detection," in IEEE Access, vol. 10, pp. 103121-103135, 2022, doi: 10.1109/ACCESS.2022.3206445.

12. M. Upadhayay, A. Sharma, G. Garg and A. Arora, "RPNDroid: Android Malware Detection using Ranked Permissions and Network Traffic," 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), London, United Kingdom, 2021, pp. 19-24, doi: 10.1109/WorldS451998.2021.9513992.
13. C. Li et al., "Backdoor Attack on Machine Learning Based Android Malware Detectors," in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 5, pp. 3357-3370, 1 Sept.-Oct. 2022, doi: 10.1109/TDSC.2021.3094824.
14. S. Kumar, D. Mishra, B. Panda and S. K. Shukla, "AndroOBFS: Time-tagged Obfuscated Android Malware Dataset with Family Information," 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR), Pittsburgh, PA, USA, 2022, pp. 454-458, doi: 10.1145/3524842.3528493.
15. L. N. Vu and S. Jung, "AdMat: A CNN-on-Matrix Approach to Android Malware Detection and Classification," in IEEE Access, vol. 9, pp. 39680-39694, 2021, doi: 10.1109/ACCESS.2021.3063748.
16. G. Canfora, F. Martinelli, F. Mercaldo, V. Nardone, A. Santone and C. A. Visaggio, "LEILA: Formal Tool for Identifying Mobile Malicious Behaviour," in IEEE Transactions on Software Engineering, vol. 45, no. 12, pp. 1230-1252, 1 Dec. 2019, doi: 10.1109/TSE.2018.2834344.
17. M. Yousefi-Azar, L. Hamey, V. Varadharajan and S. Chen, "Byte2vec: Malware Representation and Feature Selection for Android," in The Computer Journal, vol. 63, no. 1, pp. 1125-1138, Jan. 2020, doi: 10.1093/comjnl/bxz121.
18. G. Suarez-Tangil, J. E. Tapiador, F. Lombardi and R. D. Pietro, "Alterdroid: Differential Fault Analysis of Obfuscated Smartphone Malware," in IEEE Transactions on Mobile Computing, vol. 15, no. 4, pp. 789-802, 1 April 2016, doi: 10.1109/TMC.2015.2444847.
19. T. Eom, H. Kim, S. An, J. S. Park and D. S. Kim, "Android Malware Detection Using Feature Selections and Random Forest," 2018 International Conference on Software Security and Assurance (ICSSA), Seoul, Korea (South), 2018, pp. 55-61, doi: 10.1109/ICSSA45270.2018.00023.
20. Xiaohan Zhang and Zhengping Jin, "A new semantics-based android malware detection," 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2016, pp. 1412-1416, doi: 10.1109/CompComm.2016.7924936.

21. S. Dissanayake, S. Gunathunga, D. Jayanetti, K. Perera, C. Liyanapathirana and L. Rupasinghe, "An Analysis on Different Distance Measures in KNN with PCA for Android Malware Detection," 2022 22nd International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, 2022, pp. 178-182, doi: 10.1109/ICTer58063.2022.10024079.
22. M. Hassan and I. Sogukpinar, "Android Malware Variant Detection by Comparing Traditional Antivirus," 2022 7th International Conference on Computer Science and Engineering (UBMK), Diyarbakir, Turkey, 2022, pp. 507-511, doi: 10.1109/UBMK55850.2022.9919458.
23. S. Amenova, C. Turan and D. Zharkynbek, "Android Malware Classification by CNN-LSTM," 2022 International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan, 2022, pp. 1-4, doi: 10.1109/SIST54437.2022.9945816.
24. T. Mantoro, D. Stephen and W. Wandy, "Malware Detection with Obfuscation Techniques on Android Using Dynamic Analysis," 2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED), Sukabumi, Indonesia, 2022, pp. 1-6, doi: 10.1109/ICCED56140.2022.10010359.
25. J. B. S and K. R, "A state-of-the-art Analysis of Android Malware Detection Methods," 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2022, pp. 851-855, doi: 10.1109/ICOEI53556.2022.9777170.
26. H. Bai, N. Xie, X. Di and Q. Ye, "FAMD: A Fast Multifeature Android Malware Detection Framework, Design, and Implementation," in IEEE Access, vol. 8, pp. 194729-194740, 2020, doi: 10.1109/ACCESS.2020.3033026.
27. M. Awais, M. A. Tariq, J. Iqbal and Y. Masood, "Anti-Ant Framework for Android Malware Detection and Prevention Using Supervised Learning," 2023 4th International Conference on Advancements in Computational Sciences (ICACS), Lahore, Pakistan, 2023, pp. 1-5, doi: 10.1109/ICACS55311.2023.10089629.
28. T. Islam, S. Rahman, M. Hasan, A. Rahaman, I. Jabiullah,“Evaluation of N-Gram Based Multi-Layer Approach to Detect Malware in Android,Procedia Computer Science,”Volume 171,2020,Pages 1074-1082.
29. J. Mishra, S. K. Sahay, H. Rathore and L. Kumar, "Duplicates in the Drebin Dataset and Reduction in the Accuracy of the Malware Detection Models," 2021 26th IEEE

Asia-Pacific Conference on Communications (APCC), Kuala Lumpur, Malaysia, 2021, pp. 161-165, doi: 10.1109/APCC49754.2021.9609892.

30. R. Goyal, A. Spognardi, N. Dragoni and M. Argyriou, "SafeDroid: A Distributed Malware Detection Service for Android," 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 2016, pp. 59-66, doi: 10.1109/SOCA.2016.14.
31. Z. Ren, H. Wu, Q. Ning, I. Hussain, B. Chen,“End-to-end malware detection for android IoT devices using deep learning,Ad Hoc Networks,”Volume 101,2020,102098.
32. S. R. Tuan Mat, M. Razak, M. Kahar, J. Arif, A. Firdaus,"A Bayesian probability model for Android malware detection,"ICT Express,Volume 8, Issue 3,2022,Pages 424-431.
33. K. Kong, Z. Zhang, Z. Yang, Z. Zhang,”FCSCNN: Feature centralized Siamese CNN-based android malware identification,”Computers & Security,Volume 112,2022,102514,ISSN 0167-4048.
34. A. Arora, S. K. Peddoju and M. Conti, "PermPair: Android Malware Detection Using Permission Pairs," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1968-1982, 2020, doi: 10.1109/TIFS.2019.2950134.
35. A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," Journal of Information Security and Applications, Volume 58,2021.
36. Juliza Mohamad Arif, Mohd Faizal Ab Razak, Sharfah Ratibah Tuan Mat, Suryanti Awang, Nor Syahidatul Nadiah Ismail, Ahmad Firdaus,Android mobile malware detection using fuzzy AHP,Journal of Information Security and Applications,Volume 61,2021,102929,ISSN 2214-2126.
37. T. Lei, Z. Qin, Z. Wang, Q. Li and D. Ye, "EveDroid: Event-Aware Android Malware Detection Against Model Degrading for IoT Devices," in IEEE Internet of Things Journal, vol. 6, no. 4, pp. 6668-6680, Aug. 2019, doi: 10.1109/JIOT.2019.2909745.
38. R. Srivastava, R. P. Mishra, V. Kumar, H. K. Shukla, N. Goyal and C. Singh, "Android Malware Detection Amid COVID-19," 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), Moradabad, India, 2020, pp. 74-78, doi: 10.1109/SMART50582.2020.9337105.
39. J. B. S and K. R, "A state-of-the-art Analysis of Android Malware Detection Methods," 2022 6th International Conference on Trends in Electronics and

Informatics (ICOEI), Tirunelveli, India, 2022, pp. 851-855, doi: 10.1109/ICOEI53556.2022.9777170.

40. K. Aktaş and S. Sen, "Android malware detection based on runtime behaviour," 2018 26th Signal Processing and Communications Applications Conference (SIU), Izmir, Turkey, 2018, pp. 1-4, doi: 10.1109/SIU.2018.8404768.
41. P. Xiong, X. Wang, W. Niu, T. Zhu and G. Li, "Android malware detection with contrasting permission patterns," in China Communications, vol. 11, no. 8, pp. 1-14, Aug. 2014, doi: 10.1109/CC.2014.6911083.
42. X. Li, J. Liu, Y. Huo, R. Zhang and Y. Yao, "An Android malware detection method based on AndroidManifest file," 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), Beijing, China, 2016, pp. 239-243, doi: 10.1109/CCIS.2016.7790261.
43. J. Garcia, M. Hammad and S. Malek, "[Journal First] Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware," 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 2018, pp. 497-497, doi: 10.1145/3180155.3182551.
44. D. V. Ng and J. -I. G. Hwang, "Android malware detection using the dendritic cell algorithm," 2014 International Conference on Machine Learning and Cybernetics, Lanzhou, China, 2014, pp. 257-262, doi: 10.1109/ICMLC.2014.7009126.
45. F. M. Darus, N. A. A. Salleh and A. F. Mohd Ariffin, "Android Malware Detection Using Machine Learning on Image Patterns," 2018 Cyber Resilience Conference (CRC), Putrajaya, Malaysia, 2018, pp. 1-2, doi: 10.1109/CR.2018.8626828.
46. S. Iqbal and M. Zulkernine, "SpyDroid: A Framework for Employing Multiple Real-Time Malware Detectors on Android," 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), Nantucket, MA, USA, 2018, pp. 1-8, doi: 10.1109/MALWARE.2018.8659365.
47. J. Wang, B. Li and Y. Zeng, "XGBoost-Based Android Malware Detection," 2017 13th International Conference on Computational Intelligence and Security (CIS), Hong Kong, China, 2017, pp. 268-272, doi: 10.1109/CIS.2017.00065.
48. P. Zhang, S. Cheng, S. Lou and F. Jiang, "A Novel Android Malware Detection Approach Using Operand Sequences," 2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC), Shanghai, China, 2018, pp. 1-5, doi: 10.1109/SSIC.2018.8556755.

49. M. Masum and H. Shahriar, "Droid-NNet: Deep Learning Neural Network for Android Malware Detection," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 5789-5793, doi: 10.1109/BigData47090.2019.9006053.
50. N. Kumari and M. Chen, "Malware and Piracy Detection in Android Applications," 2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR), CA, USA, 2022, pp. 306-311, doi: 10.1109/MIPR54900.2022.00061.
51. H. Haidros Rahima Manzil and M. Naik S, "DynaMalDroid: Dynamic Analysis-Based Detection Framework for Android Malware Using Machine Learning Techniques," 2022 International Conference on Knowledge Engineering and Communication Systems (ICKES), Chickballapur, India, 2022, pp. 1-6, doi: 10.1109/ICKECS56523.2022.10060106.

13% Overall Similarity

Top sources found in the following databases:

- 9% Internet database
- Crossref database
- 5% Submitted Works database
- 8% Publications database
- Crossref Posted Content database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	link.springer.com Internet	5%
2	statista.com Internet	<1%
3	Huo, Yuchong, Ping Jiang, Yuan Zhu, Shuang Feng, and Xi Wu. "Optima... Crossref	<1%
4	fortunebusinessinsights.com Internet	<1%
5	University of Essex on 2023-04-25 Submitted works	<1%
6	Pallabi Borah, Hasin A. Ahmed, Dhruba K. Bhattacharyya. "A statistical ... Crossref	<1%
7	hotelnear.in Internet	<1%
8	University of Essex on 2023-01-18 Submitted works	<1%