DESIGN AND DEVELOPMENT OF MALWARE DETECTION MODELS FOR ANDROID SMARTPHONES

A Thesis Submitted
In Partial Fulfillment of the Requirements for the
Degree of

DOCTOR OF PHILOSOPHY

in

Mathematics

by

Yash Sharma

(Roll No. 2K21/PHDAM/08)

Under the Supervision of

Dr. Anshul Arora

Department of Applied Mathematics
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-110042, India

October, 2025

CANDIDATE'S DECLARATION

I, **Yash Sharma**, hereby certify that the work which is being presented in the thesis entitled **"Design and Development of Malware Detection Models for Android Smartphones**" in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy** in Mathematics, submitted in the Department of Applied Mathematics, Delhi Technological University is an authentic record of my own work carried out during the period from 22^{nd} June 2021 to 15^{th} April 2025 under the supervision of Dr. Anshul Arora.

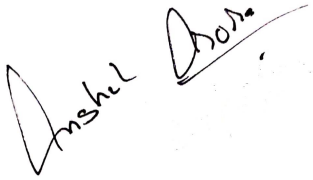
The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Yash Sharma

Roll No: 2K21/PHDAM/08

CERTIFICATE

Certified that **Mr. Yash Sharma** (2K21/PHDAM/08) has carried out their research work presented in this thesis entitled **"Design and Development of Malware Detection Models for Android Smartphones"** for the award of **Doctor of Philosophy** from Department of Applied Mathematics, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.



Dr. Anshul Arora
Supervisor
Department of Applied Mathematics
Delhi Technological University
Delhi.

ACKNOWLEDGEMENT

I express my sincere gratitude to my supervisor Dr. Anshul Arora, for his motivation and unconditional support during my research. It has been a great pleasure to work under his guidance, and I am deeply thankful for his invaluable assistance in making this work possible.

I am thankful to DRC Chairperson, Prof. Sangita Kansal, Head of the department, Prof. R. Srivastava and all DRC members for extending their support and providing all the facilities necessary for my research. I am also thankful to the Dean PG and her office staff for their prolonged support.

I extend my gratitude to my fellow research scholars, who have consistently engaged in discussions, provided support, and taken the time to read my work. I am also grateful to CSIR, Government of India, for providing a fellowship that made my Ph.D. work possible.

Finally, I am deeply indebted to my family for their unconditional support, patience, and encouragement. Without their love and belief in me, I would not be where I am today.

Above all, I express my heartfelt gratitude to the Lord Hanuman for guiding me on the right path and giving me the strength and perseverance to complete this journey.

Date: October 19, 2025 **YASH SHARMA**

Place: Delhi, India.

This Thesis is dedicated to
Lord Hanuman,
My Teachers,
&
My Family Members.

Abstract

This thesis addresses significant challenges in Android malware detection by proposing innovative solutions that enhance detection accuracy, optimize feature selection, and address the limitations of the existing approaches. The research begins with a thorough review of the current state of Android malware detection, highlighting the critical need for effective feature-ranking mechanisms to overcome the problem of overlapping features, such as permissions and intents, between benign and malicious applications. This review identified a major gap in the existing literature—while many studies applied feature ranking algorithms, few achieved both optimal feature selection and high detection accuracy. To fill this gap, we developed two static analysis-based models: *PHIGrader* and *PHIAnalyzer*. *PHIGrader* utilizes a *frequency-based Multi-Criteria Decision-Making (MCDM)* approach to rank the most commonly used static features, namely permissions, intents, and hardware components. In contrast, *PHIAnalyzer* employs a frequency-based *Chi-Square* statistical test to evaluate the effectiveness of combining the above-mentioned three features. Both models demonstrated improved accuracy in detecting malware and provided a more refined selection of features. However, static analysis alone often proves to be insufficient in detecting more sophisticated, runtime-dependent malware.

Building on the limitations identified in static analysis, the thesis transitions to dynamic analysis, specifically focusing on *CorrNetDroid*, a novel dynamic analysis-based malware detection system. This model ranks network traffic features using two key statistical measures, *crRelevance* and *Normalized Mean Residue Similarity (NMRS)*, to assess feature-class and feature-feature correlations, respectively. By applying these rankings, *CorrNetDroid* efficiently reduces the feature set while maintaining high detection accuracy. The model successfully addresses the challenge of detecting runtime malware. However, certain malware types, such as SMS-based malware, operate silently in the background without generating network traffic, underscoring the need for a comprehensive solution that combines static and dynamic analysis.

To address these limitations, the thesis introduces *AndroV-Rank*, a hybrid analysis frame-

work that combines static permissions with dynamic system calls for more robust malware detection. The *VIKOR* ranking method is employed to rank and select the most discriminative features, leading to a refined set of just 65 features, which improves both classification accuracy and efficiency compared to traditional static or dynamic analysis models. This hybrid approach effectively overcomes the challenges posed by standalone methods, as it can detect malware that relies on both static and dynamic behavior. Building on this concept, we then propose *PattMatch*, an instance-based pattern-matching classifier that utilizes *Average Weighted Pattern Scoring (AWPS)* and *Attribute Score-based Ranking (ASR)* to predict malware class labels with exceptional accuracy. This model further improves upon hybrid analysis by addressing the complexities of machine learning algorithms and achieving superior performance in both balanced and imbalanced datasets, with a remarkably high accuracy of 99.93% using only 10 attributes.

Finally, the thesis extends its scope to malware multicategory classification, where two models are developed to classify Android malware into four distinct categories: *Adware, Fraudware Trojans, Ransomware, and Spyware*. The first model relies on dynamic analysis, utilizing system calls for classification, while the second, *AndroMultiCat*, adopts a hybrid approach that combines static and dynamic features to improve classification performance. Both models demonstrate significant improvements in classification accuracy and efficiency, with the hybrid approach yielding superior results. The research concludes with a summary of the findings, highlighting the contributions of the proposed models in advancing Android malware detection, while also discussing potential future directions for the field, including the exploration of more sophisticated ranking algorithms and the integration of additional behavioral features to further enhance detection capabilities.

Contents

CANDIDATE'S DECLARATION i

CERTIFICATE BY THE SUPERVISOR iii

Acknowledgement v

Abstract ix

List of Abbreviations xxiii

1 Introduction 1

1.1 Preliminaries . . . 3
1.2 Android Malware . . . 4
1.3 Research Gaps . . . 7
1.4 Problem Statement . . . 8
1.5 Objectives and Contributions . . . 8
1.6 Organization of the thesis . . . 11

2 Literature Review 13

2.1 Static Detection Model . . . 14
2.1.1 Permissions-based . . . 16
2.1.2 Permissions with Manifest file components . . . 20
2.1.3 API calls based Detection . . . 22
2.1.4 Limitations of Static Analysis . . . 25
2.2 Dynamic Detection Model . . . 25
2.2.1 OS-based detection . . . 26
2.2.2 Network Traffic-Based Detection . . . 28
2.2.3 Limitations of Dynamic Analysis . . . 30
2.3 Hybrid Detection Model . . . 31
2.3.1 Limitations of Hybrid Analysis . . . 33
2.4 Multi-Category Malware Detection Model . . . 33

2.5 Summary . . . 35

3 *PHIGrader*: Evaluating the effectiveness of Manifest file components in Android malware detection using Multi Criteria Decision Making techniques 37

3.1 Introduction . . . 38
3.2 System Design . . . 42
3.2.1 Dataset . . . 43
3.2.2 Feature Extraction . . . 44
3.2.3 Feature Representation . . . 44
3.2.4 Features Ranking . . . 45
3.2.5 Machine Learning and Deep Learning Classifiers . . . 52
3.2.6 Proposed Malware Detection Algorithm . . . 53
3.3 Feature Ranking Results . . . 55
3.3.1 Allotting Weights To The Features . . . 55
3.3.2 Features Ranking . . . 57
3.4 Detection Results on *DATASET-1* . . . 62
3.4.1 Detection Results with *TOPSIS* . . . 62
3.4.2 Detection Results with *EDAS* . . . 65
3.4.3 Detection Results with *WASPAS* . . . 69
3.4.4 Comparison with other feature ranking techniques . . . 73
3.4.5 Comparison with other statistical tests . . . 75
3.5 Detection Results on *DATASET-2* . . . 78
3.5.1 Detection Results with *TOPSIS* . . . 79
3.5.2 Detection Results with *EDAS* . . . 81
3.5.3 Detection Results with *WASPAS* . . . 85
3.6 Discussion . . . 89
3.6.1 Comparison with other related works . . . 89
3.6.2 Limitations . . . 89
3.7 Conclusion and Future work . . . 90

4 *PHIAnalyzer*: A novel Android malware detection system using ranked Manifest file components 93

4.1 Introduction . . . 94
4.1.1 Drawbacks of existing approaches . . . 94
4.1.2 Objectives and Need of Proposed Approach . . . 95
4.2 System Design . . . 97
4.2.1 Data Acquisition and Representation . . . 97
4.2.2 Features Ranking . . . 98
4.2.3 Machine Learning and Deep Learning Classifiers . . . 99

4.2.4 Proposed Malware Detection Algorithm . . . 100
4.3 Results and Discussion . . . 101
4.3.1 Allotting Weights To The Features . . . 102
4.3.2 Features Ranking . . . 103
4.3.3 Detection Results on *DATASET-1* . . . 104
4.3.4 Detection Results on *DATASET-2* . . . 117
4.3.5 Comparison with other related works . . . 122
4.3.6 Limitations . . . 122
4.4 Conclusion and Future Work . . . 123

5 *CorrNetDroid*: Android Malware Detector leveraging a Correlation-based Feature Selection for Network Traffic features 125
5.1 Introduction . . . 125
5.2 System Design . . . 130
5.2.1 Dataset Collection . . . 131
5.2.2 Traffic Split . . . 132
5.2.3 Features Aggregation . . . 132
5.2.4 Feature Selection . . . 132
5.2.5 Proposed Detection Algorithm . . . 136
5.2.6 Machine Learning and Deep Learning Classifiers . . . 137
5.3 Results and Discussion . . . 138
5.3.1 Features Ranking . . . 138
5.3.2 Detection Results on *Testing Dataset* . . . 140
5.3.3 Comparison with other statistical tests . . . 141
5.3.4 Comparison of *NMRS* with other Correlation Measures . . . 145
5.3.5 Comparsion with other related works . . . 147
5.4 Limitations . . . 149
5.5 Conclusion and Future Work . . . 150

6 Hybrid Android Malware Detection leveraging Static Permissions and Dynamic System Calls 151
6.1 Introduction . . . 151
6.2 Proposed Hybrid Model - I . . . 153
6.2.1 Dataset Accumulation . . . 153
6.2.2 Features Ranking . . . 154
6.2.3 Machine Learning and Deep Learning Classifiers . . . 156
6.2.4 Proposed Malware Detection Algorithm . . . 156
6.3 Results and Discussion: Hybrid Model - I . . . 157
6.3.1 Feature Ranking using *VIKOR* . . . 157

6.3.2 Detection results on the Testing Dataset . . . 158
6.4 Proposed Hybrid Model - II . . . 160
6.4.1 Dataset Accumulation . . . 161
6.4.2 Methodology . . . 163
6.5 Results and Discussions: Hybrid Model - II . . . 169
6.5.1 *Attribute Score-based Ranking (ASR)* results . . . 169
6.5.2 Classification results on *Testing* dataset . . . 171
6.5.3 Comparison with other classifiers . . . 175
6.5.4 Comparison with other related works . . . 177
6.6 Conclusion and Future Work . . . 179

7 Android Malware Multi-Category Classification via Highly Discriminative Feature Ranking 181
7.1 Introduction . . . 181
7.2 Proposed Multi-Category Detection model-1 . . . 183
7.2.1 Data collection . . . 183
7.2.2 Feature Ranking . . . 184
7.2.3 Machine Learning and Deep Learning Classifiers . . . 185
7.2.4 Proposed Malware Multi-Category Classification Algorithm . . . 185
7.3 Results and Discussion: Multi-Category Model - I . . . 186
7.3.1 *ReliefF* Ranking Results . . . 186
7.3.2 Classification Results on the Testing Dataset . . . 186
7.4 Proposed Multi-Category Detection model - II . . . 187
7.4.1 Dataset Accumulation . . . 189
7.4.2 Feature Ranking . . . 190
7.4.3 Machine Learning and Deep Learning Classifiers . . . 191
7.4.4 Proposed Malware Multi-Category classification Algorithm . . . 191
7.5 Results and Discussion: Multi-Category Detection model - II . . . 191
7.5.1 Discrimination Score-based Ranking results . . . 192
7.5.2 Classification results on the Testing dataset . . . 193
7.6 Conclusion and Future Work . . . 196

8 Conclusion, Future Scope, and Social Impact 197
8.1 Conclusion . . . 197
8.2 Future Scope . . . 200
8.3 Social Impact of the proposed research . . . 202

Bibliography 204

List of Tables

1.1 Threats posed by Android malware 7

3.1 Top 20 most frequently requested permissions from both normal and malware datasets with their corresponding frequency. 39

3.2 Top 20 most frequently requested intents from both normal and malware datasets with their corresponding frequency. 39

3.3 Top 20 most frequently requested hardware components from both normal and malware datasets with their corresponding frequency. 40

3.4 Top 10 normal dominant and malware dominant permissions with their corresponding weights 56

3.5 Top 10 normal dominant and malware dominant intents with their corresponding weights 56

3.6 Top 10 normal dominant and malware dominant hardware components with their corresponding weights 57

3.7 Top 10 permissions ranked using *TOPSIS* 58

3.8 Top 10 intents ranked using *TOPSIS* 58

3.9 Top 10 hardware components ranked using *TOPSIS* 59

3.10 Top 10 permissions ranked using *EDAS* 59

3.11 Top 10 intents ranked using *EDAS* 60

3.12 Top 10 hardware components ranked using *EDAS* 60

3.13 Top 10 permissions ranked using *WASPAS* 61

3.14 Top 10 intents ranked using *WASPAS* 61

3.15 Top 10 hardware components ranked using *WASPAS* 62

3.16 Compiled Detection results (in %) on applying the proposed algorithm on *DATASET -1* 72
3.17 Top 10 features ranked using PCA 73
3.18 Top 10 features ranked using ECCD 73
3.19 Comparison of best detection results (in %) from *MCDM* techniques with Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on permissions 74
3.20 Comparison of best detection results (in %) from *MCDM* techniques with Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on intents 75
3.21 Comparison of best detection results (in %) from *MCDM* techniques with Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on hardware components 75
3.22 Top 10 features ranked using Mutual Information 76
3.23 Top 10 features ranked using Pearson Correlation Coefficient 76
3.24 Top 10 features ranked using T-Test 76
3.25 Comparison of best detection results (in %) from *MCDM* techniques with Mutual Information, Pearson Coefficient, and T-Test on permissions 77
3.26 Comparison of best detection results (in %) from *MCDM* techniques with Mutual Information, Pearson Coefficient, and T-Test on intents 78
3.27 Comparison of best detection results (in %) from *MCDM* techniques with Mutual Information, Pearson Coefficient, and T-Test on hardware components . . 78
3.28 Compiled Detection results (in %) on applying the proposed algorithm on *DATASET -2* 88
3.29 Comparison of proposed work with the existing literature based on malware detection using permissions, intents or hardware components. 90

4.1 Top 10 permissions along with their corresponding weights. 102
4.2 Top 10 intents along with their corresponding weights. 102
4.3 Top 10 hardware components along with their corresponding weights. 103
4.4 Top 10 permissions with their corresponding F-scores 104

4.5 Top 10 intents with their corresponding F-scores 104
4.6 Top 10 hardware components with their corresponding F-scores 104
4.7 Detection results with proposed approach considering only permissions 106
4.8 Detection results with proposed approach considering only intents 106
4.9 Detection results with proposed approach considering only hardware components 107
4.10 Detection results with proposed approach considering the combination of permission and intents 108
4.11 Detection results with proposed approach considering the combination of intents and hardware components 108
4.12 Detection results with proposed approach considering the combination of permissions and hardware components 109
4.13 Detection results with proposed approach considering the combination of permissions, intents and hardware components 110
4.14 Comparison of proposed frequency-based *Chi-Square* test in terms of detection accuracy upon using different combinations of features. 111
4.15 Detection results considering all features for *DATASET-1* without applying the proposed approach 111
4.16 Top 10 permissions, intents and hardware components ranked using Mutual Information 112
4.17 Top 10 permissions, intents, and hardware components ranked using Correlation Coefficient 112
4.18 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on permissions 113
4.19 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on intents 113
4.20 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on hardware components 114
4.21 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of permission and intents 115

4.22 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of intents and hardware components . 115

4.23 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of permissions and hardware components 116

4.24 Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of permissions, intents and hardware components 116

4.25 Detection results with proposed approach considering only permissions 117

4.26 Detection results with proposed approach considering only intents 118

4.27 Detection results with proposed approach considering only hardware components 119

4.28 Detection results with proposed approach considering the combination of permissions and intents 119

4.29 Detection results with proposed approach considering the combination of intents and hardware components 120

4.30 Detection results with proposed approach considering the combination of permissions and hardware components 121

4.31 Detection results with proposed approach considering the combination of permissions, intents and hardware components 122

4.32 Comparison of proposed work with the existing literature based on malware detection using permissions and intents. 123

5.1 Some traffic features and their range for malware and normal mobile traffic on smartphones 127

5.2 List of network traffic features 133

5.3 Traffic features ranked using *crRelevance* and their correspoding difference between category scores 139

5.4 20 traffic feature pairs, top and bottom each, ranked using *NMRS* and their corresponding correlation scores 139

5.5 Detection results when we apply our *NMRS*-based proposed algorithm on *crRelevance* feature ranking . . . 141

5.6 Traffic features ranked using various statistical tests . . . 144

5.7 Comparsion of *NMRS*-based proposed algorithm on *crRelevance* rankings with various statistical tests when we apply the same algorithm applied on them . . . 145

5.8 Top 10 Traffic feature pairs ranked using Pearson's correlation coefficient and their corresponding correlation scores . . . 146

5.9 Detection results when we apply Pearson's correlation coefficient-based algorithm on *crRelevance* feature ranking . . . 147

5.10 Traffic features ranked based on their deviation from normal traffic behavior . . . 148

5.11 Detection results when we implement other related work . . . 148

5.12 Comparison of proposed work with the existing literature based on malware detection using TCP flows . . . 149

6.1 Top 10 features ranked according to their *preference scores* using VIKOR for the static and dynamic category datasets . . . 157

6.2 Top 30 features ranked according to their *preference scores* using *VIKOR* for the hybrid category datasets . . . 158

6.3 Compiled detection results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets . . . 159

6.4 Top 10 permissions ranked with *ASR* along with their corresponding attribute scores and rank weights . . . 170

6.5 Top 10 system calls ranked with *ASR* along with their corresponding attribute scores and rank weights . . . 170

6.6 Top 10 permissions and system calls ranked with *ASR* along with their corresponding attribute scores and rank weights . . . 170

6.7 Classification results obtained by applying the proposed classifier to the static category dataset . . . 171

6.8 Classification results obtained by applying the proposed classifier to the dynamic category dataset . . . 172

6.9 Classification results obtained by applying the proposed classifier to the hybrid category dataset 173

6.10 Compiled detection results to compare the performance of the proposed classifier with other related works across the three dataset categories 178

6.11 Comparison of proposed work with the existing literature based on hybrid malware detection models 179

7.1 Details of the dataset including malware categories and the families comprising each category. 184

7.2 Top 10 system calls ranked in order of their *ReliefF* Score 186

7.3 Classification Accuracy Across Iterations of Feature Reduction 187

7.4 Top 10 features ranked according to their Discrimination scores for the static and dynamic category 192

7.5 Top 20 features ranked according to their Discrimination scores for the Hybrid category 193

7.6 Compiled classification results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets 194

List of Figures

1.1 Android platform architecture 3

1.2 Number of Android Malware Recorded Every Year from 2010- 2024. 5

1.3 Overview of the proposed works of this thesis 9

2.1 Taxonomy of Android malware detection techniques 14

2.2 Snapshot of permissions requested by WhatsApp Messenger app 15

2.3 Snapshot of intents requested by WhatsApp Messenger app 16

2.4 Snapshot of hardware components requested by Microsoft Edge Web Browser app 16

2.5 Snapshot of TCP flows from the network traffic of CRIDEX malware application. 26

3.1 *PHIGrader* System Design 43

3.2 Detection results with *TOPSIS* using permissions 63

3.3 Detection results with *TOPSIS* using intents 64

3.4 Detection results with *TOPSIS* using hardware components 65

3.5 Detection results with *EDAS* using permissions 67

3.6 Detection results with *EDAS* using intents 67

3.7 Detection results with *EDAS* using hardware components 68

3.8 Detection results with *WASPAS* using permissions 70

3.9 Detection results with *WASPAS* using intents 71

3.10 Detection results with *WASPAS* using hardware components 72

3.11 Detection results with *TOPSIS* using permissions 80

3.12 Detection results with *TOPSIS* using intents 81

3.13 Detection results with *TOPSIS* using hardware components 82
3.14 Detection results with *EDAS* using permissions 83
3.15 Detection results with *EDAS* using intents 84
3.16 Detection results with *EDAS* using hardware components 85
3.17 Detection results with *WASPAS* using permissions 86
3.18 Detection results with *WASPAS* using intents 87
3.19 Detection results with *WASPAS* using hardware components 88

5.1 *CorrNetDroid* System Design 131

6.1 Detection results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets 158
6.2 *PattMatch* System Design 162
6.3 Classification results obtained by applying the proposed classifier to the static category dataset 173
6.4 Classification results obtained by applying the proposed classifier to the dynamic category dataset 174
6.5 Classification results obtained by applying the proposed classifier to the hybrid category dataset 174
6.6 Comparison of our proposed classifier with other ML and DL algorithms when applied to the static, dynamic, and hybrid category datasets 176

7.1 *AndroMultiCat* System Design 188
7.2 Classification results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets 194

List of Abbreviations

Abbreviation	Description
AAPT2	Android Asset Packaging Tool
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
APK	Android Application Package
APIs	Application Programming Interfaces
ARFO	Adaptive Red Fox Optimization
ART	Android Runtime
ASR	Attribute Score-Based Ranking
AWPS	Average Weighted Pattern Scoring
BC	Bagging Classifier
BOAWFS	Bat Optimization Algorithm for Wrapper-Based Feature Selection
CIL	Class Imbalanced Learning
CNN	Convolutional Neural Network
CPPM	Contrast Permission Pattern Mining
DBN	Deep Belief Network
DNN	Dense Neural Network
DL	Deep Learning
DNS	Domain Name System
DT	Decision Trees
EDAS	Evaluation Based on Distance from Average Solution
ECCD	Entropy-Based Category Coverage Difference
ELM	Extreme Learning Machine
EMFO	Enhanced Moth Flame Optimized

List of Abbreviations (continued)

Abbreviation	Description
eSNN	Evolving Spiking Neural Network
GNN	Graphical Neural Network
HHO	Harris Hawks Optimization
HTTP	Hypertext Transfer Protocol
IWD	Intelligent Water Drop Algorithm
KCD	Keywords Correlation Distance
LR	Logistic Regression
LSTM	Long Short-Term Memory
MCDM	Multi-Criteria Decision-Making
MLP	Multilayer Perceptron
MSer	Multi-Head Squeeze and Excitation Residual Block
MSB	Mean Square Between
MSW	Mean Square Within
NB	Naive Bayes
NIS	Negative Ideal Solution
NMRS	Normalized Mean Residue Similarity
NLP	Natural Language Processing
OS	Operating System
PCA	Principal Component Analysis
PDME	Program Dissimilarity Measure Based on Entropy
PIS	Positive Ideal Solution
PL-SAE	Pseudo-Label Stacked Auto-Encoder
POPNet	Prototypical Nets
PSA	Particle Swarm Optimization
RBFN	Radial Basis Function Network
RF	Random Forest
SA	Simulated Annealing
SBRBM	Subspace-Based Restricted Boltzmann Machine
SEL	Stacking Ensemble Learning
SFS	Sequential Forward Selection
SMOTE	Synthetic Minority Oversampling Technique

List of Abbreviations (continued)

Abbreviation	Description
SOM	Self-Organizing Map
SSA	Social Spider Algorithm
STG	State Transition Graphs
TAN	Tree Augmented Naive Bayes
TCP	Transmission Control Protocol
TCN	Temporal Convolutional Network
TF-IDFCF	Term Frequency-Inverse Document Frequency Class Frequency
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution
UI	User Interface
UDP	User Datagram Protocol
VP	Voted Perceptron
WASPAS	Weighted Aggregated Sum Product Assessment
WI	Weighted Mutual Information
WPM	Weighted Product Model
WSM	Weighted Sum Model
XGBoost	Extreme Gradient Boosting

Chapter 1

Introduction

Early telephones, dated back to the late 19th century, were primarily used for business communications and connecting individuals over vast distances, revolutionizing the concept of communication. The first generation of phones was simple, designed only to transmit voice. They were bulky, wired devices with limited functionality. Over time, these devices evolved to include features such as rotary dials and push buttons, yet their core purpose remained unchanged for decades: voice communication [1]. Phones of that era were confined to fixed locations, often in homes or offices, with no portability. Thus, the utility of phones was predominantly in fixed, professional, or emergency scenarios.

The advent of smartphones in the early 21st century marked a transformative shift, turning the humble phone into a powerful multifunctional device. A smartphone is a mobile device that combines telecommunication features with advanced computing capabilities, offering internet access, touchscreen navigation, and support for apps and multimedia. The introduction of smartphones changed not only the design and capabilities of phones but also the way people interact with technology and the world around them. With the inclusion of internet connectivity, cameras, GPS, and mobile applications, the smartphone became an indispensable tool for personal, social, and professional use [2]. What started as a communication device rapidly evolved into a pocket-sized computer, capable of handling a wide range of tasks from banking to entertainment, from social networking to navigation. This transformation has drastically altered everyday life, making smartphones essential for billions of users worldwide. Statistics

indicate as of 2025, a little over 4.88 billion people use smartphones around the world[1]. Today, people are reliant on their phones for a variety of functions that extend far beyond communication—these devices now serve as our planners, wallets, and even personal assistants.

Among the diverse smartphone operating systems (OS) available, Android stands as the most dominant. While competitors such as Apple's iOS, Microsoft's Windows Phone, and BlackBerry's OS had their market shares, Android, developed by Google, has emerged as the clear leader [3]. Statistics from leading industry reports demonstrate that Android commands a substantial portion of the global mobile market. For instance, recent studies suggest that over 70% of mobile devices globally run on Android, leaving iOS as a distant second[2]. This dominance is even more pronounced in emerging markets, where Android's affordability and flexibility make it the preferred choice for both consumers and manufacturers. Other operating systems, such as Windows Phone and BlackBerry OS, have gradually lost relevance, leaving Android and iOS as the primary contenders in the smartphone industry.

Several factors contribute to Android's popularity. Firstly, Android's open-source nature allows manufacturers to modify the operating system, leading to a wide range of devices catering to different price points and user preferences[3]. This flexibility has enabled Android to be implemented in everything from high-end flagship phones to low-cost, budget-friendly models. Furthermore, Android offers a vast ecosystem of applications through the Google Play Store, providing users with access to millions of apps across various categories, from productivity tools to entertainment[4]. Its customizable interface allows users to tailor their devices according to personal preferences, a feature that has appealed to a large audience. Additionally, Android's integration with Google's suite of services, such as Gmail, Google Drive, and Google Maps, further enhances its utility, making it a highly functional and user-friendly platform[5]. Android's multitasking capabilities, wide range of hardware support, and constant updates ensure that it remains the preferred choice for a large portion of the global population[6].

[1] https://backlinko.com/smartphone-usage-statistics
[2] https://gs.statcounter.com/os-market-share/mobile/worldwide
[3] https://source.android.com/
[4] https://www.appbrain.com/stats/google-play-store-stats
[5] https://developers.google.com/services
[6] https://developer.android.com/about/versions

1.1 Preliminaries

The Android operating system, depicted in Figure 1.1, adopts a layered architecture, based on the Linux kernel, arranged as follows: Linux kernel, hardware abstraction layer (HAL), native libraries, Android Runtime (ART), Java API framework, and the application layer[7]. This hierarchical structure is designed to provide flexibility and stability across different device types.

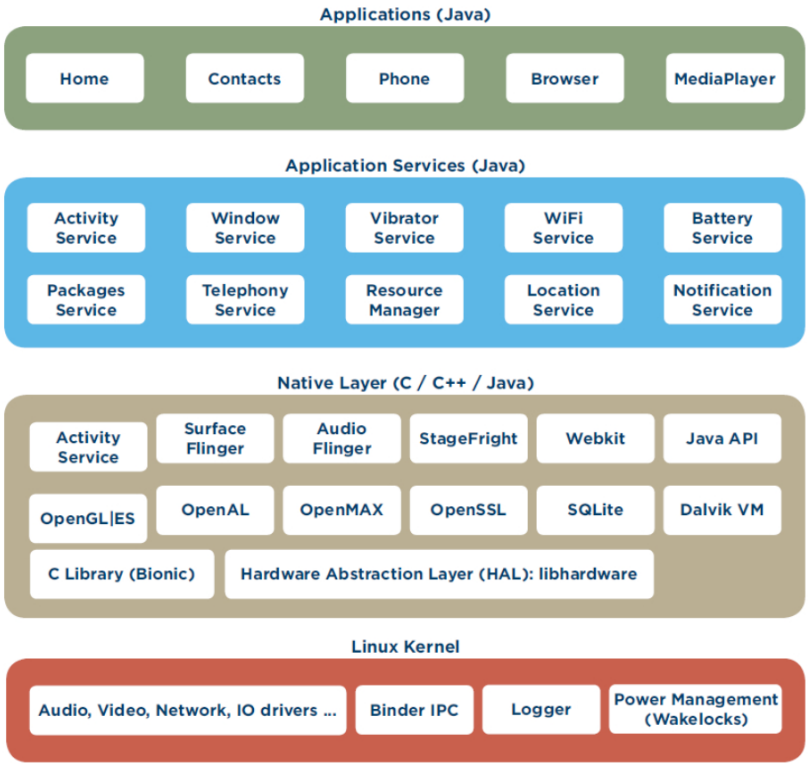


Figure 1.1: Android platform architecture

At the core of Android is the Linux kernel, responsible for managing low-level tasks such as process management, memory handling, and hardware interaction. By leveraging the robust foundation of Linux, Android benefits from proven security and resource management capabilities [4]. The Android Runtime (ART) replaced the Dalvik Virtual Machine from Android 5.0 onwards, improving performance through ahead-of-time (AOT) compilation. This enhances app execution efficiency, particularly for devices with limited resources[8]. ART is responsible for optimized memory usage, garbage collection, and thread management. The Java API framework serves as an interface between application code and underlying system components, simplifying the development of Android apps. This layer provides developers with

[7] https://developer.android.com/guide/platform
[8] https://developer.android.com/guide/platform

reusable modules for common tasks such as user interface (UI) design and inter-application communication.

At the top, the application layer comprises apps written in Java or Kotlin, which utilize Android's extensive APIs to interact with device hardware and system resources. These apps are packaged in Android application package (APK) files and deployed within the Android ecosystem. Permissions, services, and activities are configured in the critical *AndroidManifest.xml* file, which ensures the proper allocation of system resources while maintaining strict security protocols[9]. Android applications are versatile and range from utilities to complex games, benefiting from Android's layered system design and resource management.

1.2 Android Malware

Malware, short for malicious software, refers to any program or code designed to perform unauthorized or harmful actions on computing devices. Android malware specifically targets devices running the Android operating system, exploiting its vulnerabilities for malicious purposes, leading to data breaches, financial loss, and unauthorized access.

The first Android malware, *AndroidOS.FakePlayer*, emerged in 2010, disguising itself as a media player app [5]. Once installed, it sent premium-rate SMS messages without user consent, marking the start of a wave of malware targeting Android devices. Since then, cybercriminals have developed increasingly sophisticated methods to compromise Android systems, affecting both personal data and financial transactions.

One of the most notable incidents is the outbreak of the *Judy* adware in 2017. Embedded in over 40 apps on the Google Play Store, *Judy* bypassed security mechanisms and was downloaded more than 36 million times. It generated fraudulent ad clicks and harvested sensitive data, including credit card details and passwords, becoming one of the most impactful Android malware incidents[10].

Another significant malware, *ExpensiveWall*, was discovered in 2017. Hidden inside wallpaper apps, it sent fraudulent premium SMS messages, charging users for fake services. These apps, downloaded over a million times, led to considerable financial damage for users[11].

Looking at the global scale of Android malware, it has grown exponentially over the past

[9] https://developer.android.com/guide/topics/manifest/manifest-intro

[10] https://www.forbes.com/sites/thomasbrewster/2017/massive-google-android-malware-expensivewall/?sh=7a91664c477f

[11] https://portal.av-atlas.org/malware

decade. As cleary depicted by Figure 1.2, in 2012, there were 22,088 known variants, and by the end of 2024, the number skyrocketed to 35,386,293[12]. This dramatic rise highlights the need for advanced Android security solutions.

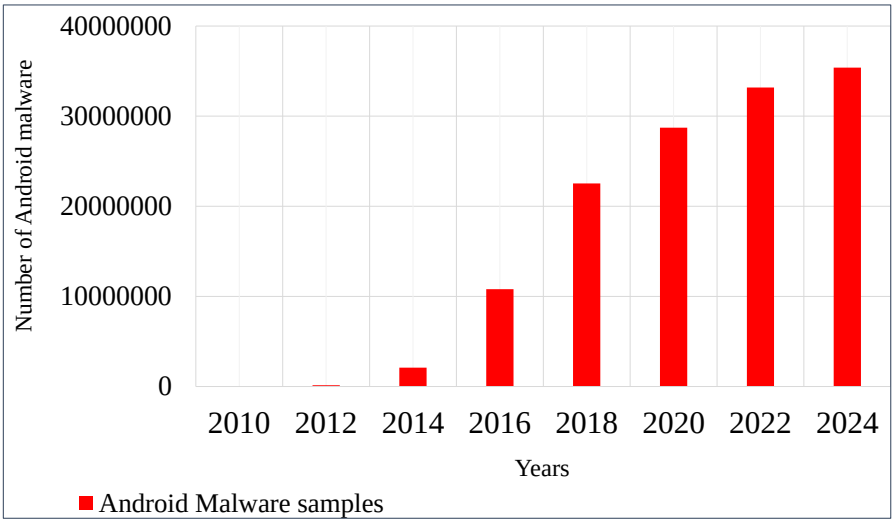


Figure 1.2: Number of Android Malware Recorded Every Year from 2010- 2024.

Real-time threat analysis further highlights the necessity of improving Android's security defenses. Google has made efforts to address these challenges by introducing services like *Bouncer*, which offers a dynamic emulated environment to detect malicious applications before they reach the Play Store [6]. However, while *Bouncer* has mitigated some risks, it has proven to be insufficient in analyzing all potential vulnerabilities. Sophisticated malware often uses dynamic code loading techniques—where malicious code is downloaded during updates, evading detection. Moreover, malware that detects the presence of emulated environments can easily bypass *Bouncer*'s safeguards. Consequently, attackers continue to find ways to circumvent security measures, making the Android platform a prime target for malware attacks. This is one of the reasons for a large number of malware attacks on the Android platform. Additionally, several other factors fuel this increase:

- The vast global Android user base often stores sensitive information on their devices, providing an opportunity for malware developers to profit from identity theft.
- Android's open-source kernel policy allows malware creators to gain insights into potential vulnerabilities within the architecture.

[12]https://portal.av-atlas.org/malware

- The availability of app markets, particularly third-party ones, offers a convenient medium for distributing malicious software.
- The structural similarity between desktop operating systems and Android makes it easier for attackers to transition from targeting desktops to mobile devices.

Next, we outline the various methods through which malware infiltrates smartphones. Below are key entry points for Android malware, explained concisely:

- **App markets:** App markets serve as a convenient entry point for attackers to disseminate malware. A significant proportion of Android malware is introduced into mobile devices through these platforms. Attackers often utilize a repackaging technique, where they decompile legitimate applications, embed malicious components, and then reassemble the app. Notable malware families such as *jSMSHider*, *DroidDream*, and *BgServ* have exploited this repackaging method to compromise the Android ecosystem.
- **Phishing Links:** Cybercriminals use deceptive links, often embedded in emails or messages, to trick users into granting access or downloading malicious software. A common example is the *Svpeng* banking trojan, which gains access to financial credentials through fake websites.
- **Drive-by Downloads:** Simply visiting a compromised website can trigger automatic downloads of malicious software without the user's knowledge.
- **Network Attacks:** Malware can infiltrate devices via insecure public Wi-Fi networks, exploiting vulnerabilities in data transmission. An example is the *HummingBad* malware, which roots the device and generates fraudulent ad revenue.

Table 1.1 summarizes the various threats posed by smartphone malware to the users and the devices [7]. The threats include system damage, financial loss to the users, and information leakage from the device, etc. Apart from them, mobile devices can also be misused by malware developers for the purpose of cyberbullying and sending spam messages on Online Social Networks (OSNs).

Table 1.1: Threats posed by Android malware

Threats		Malware Example
System Damage	Disable system functions (e.g., block calling service)	Fakebank
	Change system configuration (e.g., wallpaper)	ExpensiveWall
Financial Loss	Send SMS / MMS	FakePlayer, HippoSMS
	Dialing premium numbers	BaseBridge, BeanBot
Information Leakage	Privacy breach	BaseBridge
	Stealing banking information	EventBot
Remote Control	Mobile botnet	ADRD, AnserverBot

1.3 Research Gaps

In this section, we describe the research gaps in the existing literature for Android malware detection.

1. Features such as permissions, intents and hardware components are found to be overlapping in normal and malware datasets. Hence some feature ranking using a ranking-based algorithm is essential to identify the distinguishing features to obtain relatively good accuracy. Unfortunately, several works in the literature missed the critical aspect of feature ranking and, thus, failed to obtain the best features. Though some researchers applied a ranking algorithm to choose the best set of features, at the same time were unable to produce optimum accuracy.

   Research Question 1 : How to design and develop a static detection model with ranked manifest file components such as permissions, intents, and hardware components, to identify the best set of static features that can give relatively better accuracy.

2. The static analysis aims to investigate malware without executing the application but by collecting basic information about the app, such as manifest file components or its source code. However, it has been observed that some stealthier malware rely on the runtime events, i.e., calling a phone, sending SMS, etc., to get activated; hence, such malicious apps may evade static detection.

3. In case of dynamic analysis, network traffic has been extensively used for intrusion detection on desktop systems with high accuracy [8] [9], however, it is not that explored in case of mobile malware.

4. Network traffic flows have high similarity in case of normal and malware traffic in mobile environment. For instance, values of the feature *Flow duration* ranges from 0 to 1537.4 in normal mobile traffic and 0 to 1687.6 in malware traffic and *Time interval between packets received* ranges from 0 to 181.5 in normal mobile traffic and 0 to 378.2 in malware traffic. Such overlapping ranges in network traffic again highlight the importance of ranking the traffic features to identify the distinguishing ones, which has not been explored much in the existing literature.

   Research Question 2 : How to design and develop a dynamic Android malware detector using ranked network traffic features to detect stealthier Android malware samples that may evade static detection?

5. Not all malware samples generate network traffic; for instance, malware might only send SMS in the background without generating any network traffic. Hence, network traffic-based detection mechanisms cannot detect such samples, and we can argue that static and dynamic analysis have their limitations. We aim to target these limitations with our proposed objectives.

   Research Question 3 : How to combine static and dynamic analysis and propose a hybrid Android malware detection model that combines static with dynamic features such as static permissions with dynamic system calls.

1.4 Problem Statement

We aim to develop efficient Android malware detection models utilizing traditional analysis techniques—static, dynamic, and hybrid—while addressing their limitations to accurately classify testing applications as benign or malicious. Additionally, we incorporate novel feature ranking/selection methods to extract features with optimal class-distinguishing capabilities, ensuring superior detection/classification accuracy. Figure 1.3 provides a brief yet complete idea of the proposed models of this theis.

1.5 Objectives and Contributions

1. To review the existing related work proposed in the field of Android malware detection.

   Contribution 1 - Our review of existing research on Android malware detection classifies studies into three primary categories based on their analysis techniques: static, dynamic,

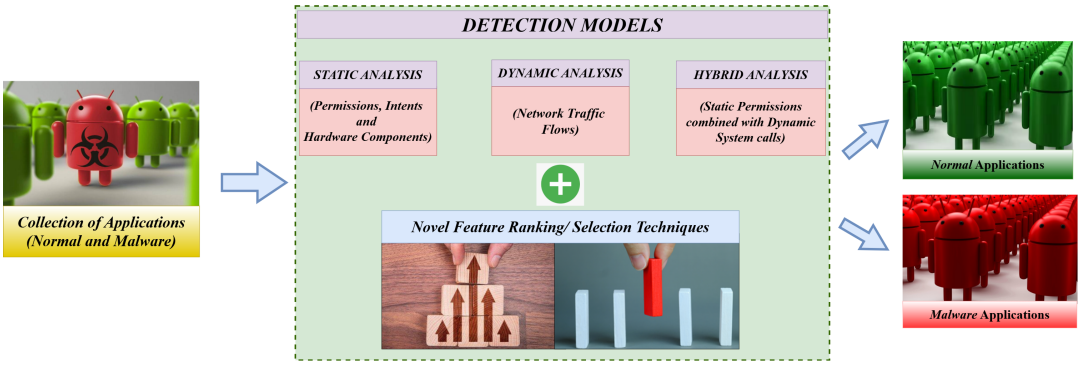


Figure 1.3: Overview of the proposed works of this thesis

and hybrid detection models. For each model, we address the limitations present in current Android malware detection methods. Furthermore, we highlight significant studies that have advanced multi-category classification approaches within the field of Android malware detection.

2. To design and develop a static detection model with ranked manifest file components such as permissions, intents, etc., to identify the best set of static features that can give relatively better accuracy.

   Contribution 2 - We propose *PHIGrader*, an advanced Android malware detection system that ranks and assesses three critical static feature types—permissions, intents, and hardware components—using frequency-based *multi-criteria decision-making (MCDM)* techniques. By leveraging these ranked features with machine learning and deep learning classifiers, *PHIGrader* achieves optimal detection performance. Specifically, when the top 46 features are selected using the *TOPSIS* method, *PHIGrader* attains an impressive 99.10% detection accuracy, outperforming models based on any single feature type or alternative *MCDM* techniques.

   With a belief that further combining these features leverages their unique strengths and mitigates their individual limitations, creating a more comprehensive detection framework, we introduce *PHIAnalyzer* that explores seven distinct combinations of these three feature types to identify the most optimal feature subset achieving higher detection accuracy. This system ranks the features and their combinations using a frequency-based test *Chi-square*, followed by a novel detection algorithm applied to the ranked features. The result is an efficient model achieving 98.49% accuracy using only 12 features—a balanced combination of the top six permissions and top six intents—yielding better accuracy than any single feature set or other feature combinations.

3. To design and develop a dynamic Android malware detector using ranked network traffic features to detect stealthier Android malware samples that may evade static detection.

   Contribution 3 - We propose a dynamic analysis-based Android malware detection system, termed as *CorrNetDroid*, in which we rank the traffic features using two statistical measures, namely *crRelevance* and *Normalized Mean Residue Similarity (NMRS)*, to find feature-class and feature-feature correlation respectively. We further incorporate them into our novel detection algorithm with an aim to select the best subset of features. The experimental results highlight that our *NMRS*-based proposed detection algorithm on *crRelevance* rankings can effectively reduce the feature set while detecting Android malware with 99.50% accuracy on considering two network traffic features, namely *Packet_size_received* and *Time_interval_between_ packets_received*.

4. To combine static and dynamic analysis in order to propose a hybrid Android malware detection model that utilizes static with dynamic features such as static permissions with dynamic system calls.

   Contribution 4 - Our proposed model introduces *PattMatch*, an instance-based pattern-matching classifier that employs *Average Weighted Pattern Scoring* (*AWPS*) and *Attribute Score-based Ranking (ASR)* to accurately predict class labels by matching the patterns of test samples with training patterns. Additionally, by reducing the feature set to the most relevant attributes, we achieve optimal classification accuracy. Experimental results show our model outperforms both static and dynamic approaches, achieving 99.93% accuracy with just 10 attributes.

   Unlike traditional machine learning and deep learning classifiers, our classifier demonstrated superior detection accuracy on the same dataset highlighting the robustness and efficiency of the proposed hybrid detection model.

1.6 Organization of the thesis

Chapter 1: This chapter provides an overview of the emergence of Android as an operating system, followed by a discussion on Android malware, including its historical background, infection methods, and associated risks. Additionally, it presents the motivation behind selecting the research problem, identifying relevant gaps in existing literature. The chapter concludes by outlining the technical contributions of our proposed approach.

Chapter 2: This chapter reviews existing research on Android malware detection, categorized into static, dynamic, and hybrid detection models. For each model, we highlight current limitations in detecting Android malware. Additionally, we review multi-category classification methods developed by other researchers that identify specific malware types using static and dynamic features, focusing on notable studies that have advanced this approach in Android malware detection. The chapter concludes with insights and suggestions for future research directions.

Chapter 3: This chapter introduces *PHIGrader*, an innovative Android malware detection system that ranks and assesses the effectiveness of the most commonly used static features—permissions, intents, and hardware components—using a frequency-based *multi-criteria decision-making (MCDM)* approach. This system incorporates a novel detection algorithm with individual feature rankings involving various machine learning and deep learning classifiers to detect malware. The proposed system, as an output, gives the best set of features as well as the feature type.

Chapter 4: In this chapter, we present *PHIAnalyzer*, a novel Android malware detection system that ranks permissions, intents, and hardware components using a frequency-based *Chi-square* test. The detection algorithm then evaluates seven possible feature combinations—permissions alone, intents alone, hardware components alone, as well as all pairwise combinations to identify the best set of features achieving higher detection accuracy. Our experiments demonstrate that the proposed frequency-based *Chi-square* ranking is better than other various statistical tests when applied to the same datasets.

Chapter 5: In this chapter, we propose a network traffic-based Android malware detection system, termed as *CorrNetDroid*, in which we rank the traffic features using two statistical measures, namely *crRelevance* and *Normalized Mean Residue Similarity (NMRS)*, to find feature-class and feature-feature correlation respectively. We further incorporate them into our novel

detection algorithm with an aim to select the best and inversely correlated features. The experimental results highlight that our *NMRS*-based proposed detection algorithm on *crRelevance* rankings can effectively reduce the feature set while detecting Android malware with remarkable detection accuracy.

Chapter 6: This chapter firstly presents *AndroV-Rank*, a robust hybrid analysis framework for Android malware detection that integrates static permissions and dynamic system calls to extract a refined set of class-distinguishing features. By employing the *VIKOR* method for feature ranking, our approach not only enhances detection accuracy but also streamlines the feature set to a mere 65 attributes while maintaining remarkably high detection accuracy.

Building upon this foundation and addressing the limitations of machine learning classifiers also, we further propose *PattMatch*, an instance-based pattern-matching classifier that employs *Average Weighted Pattern Scoring* (*AWPS*) and *Attribute Score-based Ranking (ASR)* to accurately predict class labels by matching the patterns of test samples with training patterns without the dependance over conventional classifiers. Additionally, by reducing the feature set to the most relevant attributes, we achieve optimal classification accuracy. Unlike traditional machine and deep learning algorithms, our classifier demonstrated superior detection accuracy, highlighting the robustness and efficiency of the proposed hybrid detection model.

Chapter 7: This chapter introduces two models designed for multi-category classification of Android malware into four distinct categories: *Adware, Fraudware Trojans, Ransomware, and Spyware*. Both models aim to achieve high classification accuracy while minimizing the number of features required. The first model is based on dynamic analysis and utilizes system calls for the classification task. In contrast, the second model, *AndroMultiCat*, adopts a hybrid analysis approach, combining static permissions with dynamic system calls for enhanced performance.

Chapter 8: In this chapter, we summarize our findings, discuss the main conclusions, analyze the contributions of this thesis and the achieved results. We also present a discussion on the open research problems in this field and the future work that we plan to focus on.

Chapter 2

Literature Review

This chapter reviews existing research on Android malware detection, as summarized in Figure 2.1. The related work is organized into three main categories based on the analysis techniques employed: static, dynamic, and hybrid detection models. Section 2.1 covers research focused on static detection models, followed by a discussion of dynamic detection models in Section 2.2, and hybrid detection models in Section 2.3. For each detection model, we also outline the limitations of current approaches to Android malware detection. Binary classification, which differentiates between benign and malicious samples, serves as an essential initial step. However, multi-category classification, which further identifies the specific type of malware is also an additional step taken up by many researchers using a range of static and dynamic features. Section 2.4 reviews notable studies that have contributed to advancing multi-category classification in Android malware detection. The chapter concludes with insights and future directions in Section 2.5.

.

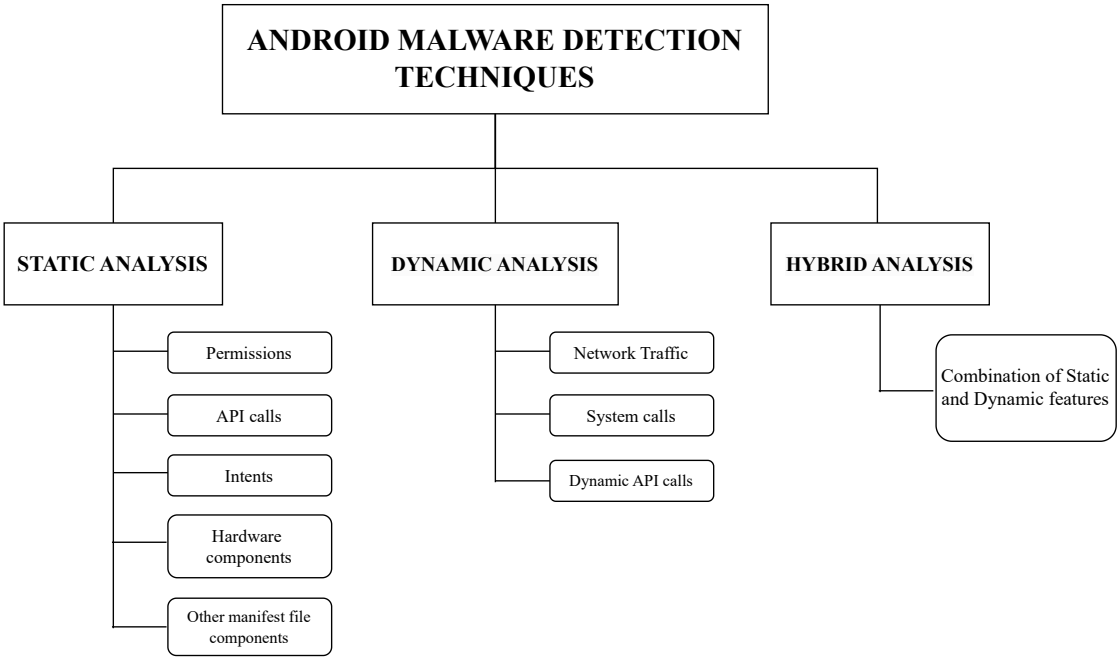


Figure 2.1: Taxonomy of Android malware detection techniques

2.1 Static Detection Model

The static analysis investigates malware without real code or instructions being executed. It provides basic information about app functionality and collects technical indicators from *AndroidManifest.xml* and other resource files. In other words, it can be defined as a source code review of an Android application file. Several reverse engineering tools like *Apktool*[1] or *AAPT2* [2] can be used to decompile an apk and extract the required features. Features that can perform static analysis of applications are called static features. Some commonly used examples of static features are explained in brief below, with Android permission being the most popular.

1. Permissions- Android permissions are security features that regulate an application's access to certain device resources or sensitive data. These permissions are declared in the *AndroidManifest.xml* file and are crucial for maintaining user privacy and ensuring the security of the Android ecosystem. Permission is declared using the <*uses-permission*>tag within the manifest file. For example, as shown in Figure 2.2, which is a snapshot of the *AndroidManifest.xml* file of the *WhatsApp Messenger* app, requires permissions such as { *READ_PHONE_STATE, READ_PHONE_NUMBERS, RECEIVE_SMS, VIBRATE and AUTHEN-*

[1]https://apktool.en.lo4d.com/windows
[2]https://developer.Android.com/studio/command-line/aapt2

TICATE_ACCOUNTS } to execute on Android smartphones.

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
```

Figure 2.2: Snapshot of permissions requested by WhatsApp Messenger app

Permissions in Android play an integral part in creating a secure and privacy-respecting ecosystem. They empower users with control, enable developers to build trustworthy applications, and contribute to the overall security of the Android platform. Some permissions fall under the category of install-time permissions, i.e., they are automatically granted upon the installation of the app, whereas some permissions are known as runtime permissions, which are further requested at runtime. Install-time permissions permit the app limited access to restricted data or actions that can affect the user to a minimal amount. Install-time permissions can be further divided into the following types:

- Normal permissions - These permissions present minimal risk to the user's privacy and the functionality of other apps.
- Signature permissions - These permissions are granted by the permission check system only when the requesting app is signed by the same certificate as the one that declared the permission.

Runtime permissions, often addressed as dangerous permissions, are requested at runtime by the application to request access to view restricted data or perform any prohibited action by presenting a runtime permission request prompt.

2. API calls- Application Programming Interfaces (APIs) act as a medium for one program to interact with another, and an API call or request can be defined as a message sent to a server asking an API to provide a service or information. After traveling from a client to an API endpoint and being received by the server, the request is processed, and the response is executed.

3. Intent: An intent is a messaging object that a developer can use to request an action from another app component. For example, as shown in Figure 2.3, which is the snapshot of the *AndroidManifest.xml* file of the *WhatsApp Messenger* app, which requires intents such as *REQUEST* and *DEFAULT* to execute on Android smartphones. The three main

fundamental use cases of intent are starting an activity, starting a service, and delivering a broadcast.

```
<intent-filter>
  <action android:name="com.whatsapp.instrumentation.REQUEST"/>
  <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

Figure 2.3: Snapshot of intents requested by WhatsApp Messenger app

4. Hardware components- The hardware components, declared using the <*uses-feature*>tag, allow the declaration of the hardware components that an app needs. For example, as shown in Figure 2.4, which is a snapshot of the *AndroidManifest.xml* file of the *Microsoft Edge Web Browser* app, hardware components such as { *location.gps, camera, microphone and touchscreen* } are required to execute on Android smartphones.

```
<uses-feature android:name="android.hardware.location.gps" android:required="false"/>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.microphone" android:required="false"/>
<uses-feature android:name="android.hardware.touchscreen" android:required="false"/>
```

Figure 2.4: Snapshot of hardware components requested by Microsoft Edge Web Browser app

Other static features include activities, services, broadcast receivers [10], opcode sequences [11], libraries, content providers, Resource names, .dex codes [12], source codes, and other metadata [13]. Among these, permissions are the most extensively used, followed by API calls and intents. These related studies can be further classified into three categories: permissions-based detection, permissions with manifest file components-based detection, and API call-based detection.

2.1.1 Permissions-based

In this section, we review the works that have used entirely permissions for malware analysis or detection. The authors in [14] worked on improving an existing frequency-based method by adding the criteria of adding the class frequency of a particular feature and named it *Term Frequency-Inverse Document Frequency Class Frequency* (TF-IDFCF). Finally, they used multiple ML classifiers to verify the working of their proposed model. Şahın et al. [15] presented a method that relies on the utilization of permissions. The researchers presented two linear regression classifiers and conducted a comparative analysis of the detection outcomes between their proposed classifiers and conventional machine learning classifiers. Furthermore, they

implemented two ensemble classifiers utilizing the bagging technique. The authors in [16] proposed a permission identification model called *SigPid*. Initially, permissions frequently and rarely requested by malware were identified through a pruning process. After normalizing the data, they calculated each permission's rate based on its support value, ranging from -1 to 1. Two ranked lists were created—one in ascending order, the other in descending order. The top values from both lists were used to compute metrics such as TPR, FPR, recall, precision, and F-measure. This process was repeated, considering the top three values and refining the metrics until the optimal set of permissions was identified. The work proposed in [17] analyzed a permission-based Android malware system in which they proposed a permission weight approach, namely Relevance Frequency. They applied their proposed approach to various machine learning algorithms and concluded by comparing the results of their study with the existing or previous methods.

Talha et al. [18] introduced APK Auditor, a permission-based system for Android malware detection. It comprises three main elements: an APK auditor client, a signature database, and a central server. The client submits requests to assess application trustworthiness. The signature database manages permissions, services, and receivers, while the central server computes the malware score based on these permissions. The authors in [19] extracted a set of 123 dynamic permissions from 11000 Android applications. These collected apk packages were made to run with the emulator bluestack. Finally, permissions were extracted by running a Java code and were divided into safe and unsafe permissions. Ultimately, they evaluated the performance of machine learning classifiers on the dataset. In [20], the AppPerm analyzer assessed manifest and code permissions separately. Manifest permissions were retrieved from the *AndroidManifest.xml* file, while code permissions were identified from the decoded APK source code. The authors constructed a feature vector and evaluated six score types. Thresholds were set based on accuracy, sensitivity, and specificity, classifying apps below the threshold as benign and those at or above it as malware.

The authors in [21] proposed an Android malware detection model based on improved Naïve Bayes classification. They determined the value of Pearson Correlation Coefficient "r" and deleted the permissions whose value "r" was less than the threshold "ρ" and derived the new permission set. Further, they got the improved detection model by clustering based on information theory. The authors in [22] discussed an approach based on sequence alignment. This work took a DNA element as permission and determined permission patterns for normal and malicious samples. It is a technique related to bioinformatics used to identify similarities

between applications by evaluating a similarity score and setting up a threshold. In [23], the authors described a monitoring tool to keep track of permission requests from various applications. The monitoring system took the help of the Broadcast Receiver and intent object to detect update events. The app's name and installation time were saved along with the requested permissions; hence, when a new file was generated asking for a new set of permissions, they could be easily identified. Ultimately, they used the pattern of permission sets for known malware applications to match up with the testing dataset permission sets to classify applications as malicious.

Ilham et al. [24] described a novel approach based on permissions. The authors applied filter feature selection algorithms and machine learning algorithms to classify applications in WEKA. In [25], the authors claimed to introduce a new approach called permission maps (Perm- maps) that could combine information related to the Android permissions with their severity level. In the end, (Convolutional Neural Network) CNN techniques were used to classify several malware types. Xiong et al. [26] utilized the dominant permission patterns in either malware dataset or clean dataset to work as a weak classifier in the proposed Enclamald, an ensemble classifier. The permission patterns defined in both datasets were also used but only with significant differences in their support degrees. An unknown application is fed to the classifier, and after computing the score of the weak classifier with a discrimination coefficient, the application is categorized into normal or malware. The authors in [27] introduced a two-layered malware security and detection model by improving the Random Forest Algorithm in the first layer after submitting the fuzzy sets. In the second layer, they mined the sensitive cluster of permissions to analyze the fuzzy sets using the Apriori Algorithm. The work proposed in [28] mainly used a couple of feature extraction algorithms called Sequential Forward selection (SFS) and Principal component Analysis (PCA) to identify the type of permissions and took down the malicious application detection by limiting the permissions that seemed dangerous using the centralized algorithm.

Amer [29] worked on creating an ensemble model based upon multiple machine learning classifiers to train and test the given data. They were subsequently categorizing the apps as malware or benign. The authors emphasized the efficiency of their model as their robustness feature, and it outperformed the previous works in terms of accuracy. The authors in [30] used feature reduction techniques such as Information gain, Relief, and Gain Ratio to take only the most influential set of permissions out of the entire collection. Further, to detect malware from the used dataset, supervised classifiers were used. Sirisha et al. [31] built a sequential neural

network model for training and later tested the permission data with three hidden layers to classify the applications. First, a threshold was set in the sigmoid output in the output layer. Then, all the applications exceeding that threshold were considered malware.

The authors in [32] focused on risk scoring the applications using the Naïve Bayes method and its advanced modifications and mixture models. According to the authors, their approach can be used as a feedback mechanism for the developers as they might get an idea of which permission to keep or lose to make their app less risky. To report their results, they used the Radius of Curvature Curves to compare a randomly selected app with a particular risk value being used as an indicator of a malicious app. They concluded the paper by stating that Naïve Bayes with Informative Priors works best while ranking the apps and risk scoring. In [33], the authors introduced a framework that utilized Natural Language Processing (NLP) techniques called WHYPER, which reads application descriptions to inform the user why the application needs particular permission. For this, they performed analysis over snapshots of the Android application's descriptions, parsed them using an NLP parser, and produced the Annotated description with the help of a semantic engine. Samra et al. [34] worked on making clusters of two categories of Android applications, namely business, and tool, using the K-means Clustering algorithm. The clustering algorithm uses permissions as features extracted from the XML files of the applications. The detection results indicated that their clustering technique could efficiently detect malware.

The authors in [35], after extracting permissions as features from various applications, used Information gained to select k best features. Then on the extracted features, they further applied the K-means clustering algorithm, classified it using a decision tree, and concluded the paper by showing their detection results after using several machine learning algorithms. In [36], the authors extracted permissions from various applications, studied their frequency, and observed that the chances of malware asking for a single permission are comparatively higher than the normal apps. Further, they applied various machine learning algorithms with different values of k in k fold cross validation to note down the accuracy values and called their whole approach Permission Usage to detect Malware in Android (PUMA). Moonsamy et al. [37] emphasized the importance of promoting the utility of "Used" as well as "Required" permissions. They used the Biclustering method in the first step to visualize the permissions, later to use the rare yet unique as well as frequently asked permissions; they proposed the Contrast Permission Pattern Mining (CPPM) method in which they reduced the dataset to contrasting permissions pattern by taking the support score for each feature. Finally, they selected the

permissions with the maximum support difference between the normal and malicious datasets. The authors in [38] presented the Appguard system, which has proven helpful in customizing security policies on untrusted applications. Whenever a new app gets installed, its proposed model asks the client to secure it, and then it installs a new modified app after rewriting its policies and deleting the old app. In [39], the authors divided the permission into four groups: Android, custom, dangerous, and all permissions. Further, they used this division to calculate eight permission pair scores, four each for normal and malicious. In the end, per-pair scores for normal and malware apps were used to classify the app as benign or dangerous. Finally, they concluded their paper by stating the detection results with machine learning algorithms such as Random Forest and Stacking Ensemble Learning (SEL).

2.1.2 Permissions with Manifest file components

Few studies analyzed, in addition to the permissions, other manifest file components as well, such as intents, hardware components, opcode sequences, .dex code etc. In this section, we review the works that have used manifest file components for malware analysis or detection in combination with permissions.

The work done by the authors in [40] was threefold. In the first step, they filtered out the most relevant features from the entire set using mutual information gain. Then, they did the same for another feature set with top code-based features. Finally, they repeated this step with a combination of permissions and code-based features. Their conclusion proved that the hybrid features provided the best detection accuracy. Seyfari and Meimandi [41] tackled the issue of many features by employing an ensemble approach that combined the Simulated Annealing (SA) algorithm with fuzzy logic. This approach, which was assessed using conventional machine learning classifiers, exhibited efficacy in exploring the solution neighborhood.

In a different study, [42] the authors built a detection system utilizing the LSSVM learning approach, incorporating ten unique feature selections and ranking techniques. Chaudhary and Masood [43] employed a comparative methodology by utilizing both the entire dataset and a diminished set obtained by the Chi-square feature reduction technique. By utilizing the Convolutional Neural Networks (CNN) method with the entire dataset of permissions and intents, researchers noticed improved performance and decreased overhead while using the smaller dataset as opposed to the complete one. Rahima Manzil and Naik [44] introduced a

novel method for feature selection, where they utilized Hamming distance and threshold techniques to identify the most relevant set of permissions and intents. Afterward, they utilized various classifiers, including machine learning, deep learning, and ensemble learning, to identify Android malware. Bai et al. [45] chose permissions and opcode sequences to feed into their CatBoost classifier for malware detection and family classification. In the first step, they extracted the permissions, followed by using the N-gram algorithm on the opcode sequences. Furthermore, for the feature selection method, the FCBF algorithm was used before the testing phase. Alazab et al. [46] innovated an algorithm based on scoring and grouping techniques to identify similar repackaged applications by comparing the frequency distributions of API calls and permissions between two applications. They even used feature selection techniques, such as information gain, along with machine learning classifiers to choose the best features and obtain the highest accuracy.

Bhat and Dutta [47] extracted permissions and various other features in the data collection step. To reduce the feature set, infrequent features or common features in both classes were removed because they were insufficiently informative. Moreover, they ranked the features based on their information gain scores to choose the optimum set. Finally, they used machine learning classifiers to showcase their detection results. Song et al. [48] matched the dangerous permissions, their combinations, and other malicious features with permissions requested by unknown applications to generate a detection report and submit it to the user. Based on these results, they built a threat degree threshold model for detecting malicious behavior.

Dehkordy and Rasoolzadegan [49] addressed the issue of balancing the dataset before detection using techniques such as the synthetic minority oversampling technique (SMOTE), random undersampling, and a hybrid method involving both to balance the huge dataset of ten types of features. In addition, they reduced the dataset using frequency ranking-based methods and further used machine learning classifiers for detection after balancing the dataset. Nguyen et al. [50] extracted 12 types of features, categorized them into three groups, and further reduced the feature set using a support vector machine (SVM), deep neural network (DNN), and analysis of variance (ANOVA). After reducing the set, the authors fed the group results individually as multiple inputs to the DNN to later combine into one final DNN to learn the abstract of each feature vector before making the final decision.

Firdaus et al. [51] took a different approach by extracting permissions and rare features such as directory paths and telephony. Furthermore, after ranking and reducing the feature set using information gain and frequency-based methods, the results were fed to various bio-inspired

artificial neural network classifiers, namely multilayer perceptron (MLP), voted perceptron (VP), and radial basis function network (RBFN). Varsha et al. [52] extracted permissions, opcode, and various other manifest file features to build a detection model. The authors used various feature ranking and selection techniques, such as naive Bayes (NB), weight calculation, entropy-based Category Coverage Difference (ECCD), and Weighted Mutual Information (WI), to reduce the feature set, choose the most relevant features, and fed them to various machine learning classifiers.

Sun et al. [53] proposed a non-parametric learning framework using the positive and unlabeled (PU) learning method to learn and detect malware after removing irrelevant features using frequency ratio criteria and PCA techniques. Finally, they compared the results of their PU learning approach with those of other machine learning classifiers and approaches. Rathore et al. [54] underscored the significance of feature reduction and ranking by employing an extensive array of feature sets. They proposed a robust feature reduction method that employs diverse classifiers and feature sets, encompassing permissions, intents, opcode sequences, and mutually exclusive and merged feature spaces. Despite a reduction of up to 90% in feature size, this impacted the original detection accuracy to some extent, but concurrently, it effectively streamlined test and training times.

2.1.3 API calls based Detection

In this section, we review the works that have used API calls, within the Java source code of the app, to detect Android malware. Almahmoud et al. [55] moved forward with their approach by performing a static analysis of four features, permissions, API calls, monitoring system events, and permission rates. They aimed to compare the detection results of recurrent neural networks with those of traditional machine learning classifiers. Mahindru and Sangal [56] opted for Artificial Neural Networks (ANN), more especially self-organizing maps (SOMs), as their classifier of choice. The researchers employed six feature selection strategies to identify malware behavior, utilizing permissions, API calls, user rating, and the quantity of user-downloaded apps as the features. Taheri et al. [57] calculated hamming distance between features, performed static analysis, and built four new KNN-based classifiers. The features extracted were permissions, API calls, and intents. They used the random forest as a feature selection algorithm and concluded their work by comparing it with various state-of-the-art techniques such as mixed and separate solutions, the program dissimilarity measure based on

entropy (PDME), and FalDroid algorithms. Mohamed et al. [58] proposed an Android malware detection system that uses only the most common permissions and API calls and feeds them to machine learning classifiers. Sheen et al. [59] proposed a multi-feature collaborative decision fusion method to club the decisions predicted by various classifiers and for various features such as permissions and API calls. The authors also used several feature ranking methods, such as chi-square, relief, and information gain, to reduce the feature set before the testing phase.

Mahindru and Sangal [60] extracted features such as permissions and API calls to build a detection system using various machine learning algorithms, including supervised, unsupervised, semi-supervised, and hybrid learning classifiers. The authors used ten distinct feature selection and ranking techniques to deal with the dimensionality issue and reduced the feature set. Taheri et al. [61] proposed a couple of defense methods, particularly for adversarial attacks, using robust-NN and C4N CNN algorithms and feature sets including API and permissions. Mahindru and Sangal [62] extracted features such as permissions and API calls, rating, and the number of users downloading the app to build a detection system using various unsupervised machine learning algorithms. The authors used ten distinct feature selection and ranking techniques to deal with the dimensionality issue and reduced the feature set.

Xie et al. [63] proposed an analysis-based approach to fingerprint Android malware families to describe their different behaviors. For this, they extracted permissions, API calls, and hardware components, ranked them based on Fisher score and frequency-based methods, and chose the top 20 features to be used for fingerprinting. Lastly, they used an SVM machine learning classifier to check the efficiency of their proposed approach. AlJarrah et al. [64] extracted permissions, API calls, and contextual information, tackled the dimensionality problem using Information Gain, and fed the features to various machine learning algorithms. Mahesh and Hemalatha [65] combined the CNN classifier with an Adaptive Red Fox Optimization (ARFO) technique to propose a new approach for malware detection. To conduct their research, they extracted permissions and API calls and reduced the dataset using the Minmax technique.

Keyvanpour et al. [66] mainly used three feature selection techniques on their extracted features, namely, permissions, API calls, intents, and hardware components, and reduced the set with the help of frequency-based, RF weigh-based, and feature group frequency-based methods and further fed the results to various machine learning classifiers. Mahindru and Sangal [67] proposed a static analysis approach that extracts permissions and API call features for detection. The authors further used the t-test and multivariate linear regression stepwise forward

selection and cross-correlation methods to reduce the feature set. Finally, the results were fed to various machine learning and ensemble classifiers, such as radial basis function neural networks, using three different ensemble methods. Sun et al. [68] extracted permissions, API calls, intents, and package names as keyword features and determined the correlation between them using the Keywords Correlation Distance(KCD) technique. The smaller the KCD, the closer the keywords. The extracted features are then utilized by the SVM classifier. Arp et al. [69] performed static analysis using various features such as intents, permissions, hardware components, network addresses, app components, and API calls to build a detection model and address the limitations of static analysis. They proposed a detection system that could provide efficient runtime performance and worked on many malware datasets by mapping features to a joint vector space, where patterns and combinations were analyzed. Zhu et al. [70] opted for static analysis in the development of their detection model. They leveraged permissions, API calls, and hardware features as inputs for their CNN-based multi-head Squeeze and Excitation Residual block (MSer).

İbrahim et al. [71] used various static features, including permissions, API calls, receivers, and services. They proposed new features, such as file size and fuzzy hash values, and processed them using a deep learning model, comparing its efficiency with that of several other machine learning classifiers. Kabakus [72] proposed a neural network-based model that uses one-dimensional data as input for training and testing. The features included intents, API calls, and permissions. Yuan et al. [73] introduced a broad learning approach similar to a flat neural network with two hidden layers for lightweight on-device detection. They used features such as permissions, intent actions, and API calls, outperforming both shallow and some deep learning models in on-device training. The authors in proposed a privacy-preserving framework that leveraged federated learning (FL) to collaboratively train Android malware detection models across distributed user devices without sharing local data. The authors in [74] employed a deep neural network within the Federated Learning framework to improve detection accuracy while addressing the privacy and scalability limitations of centralized systems. Their experimental results indicated that the FL-based approach achieved performance comparable to centralized training while maintaining data confidentiality and adaptability to distributed environments. This research demonstrated the potential of federated learning as a secure and scalable paradigm for Android malware detection.

2.1.4 Limitations of Static Analysis

Static analysis, while widely used in malware detection, has several inherent limitations. Since static analysis focuses on examining the code without executing it, it often misses critical context-dependent actions such as code obfuscation, dynamic code loading, and runtime permissions modifications. Furthermore, static methods struggle with evaluating interdependencies between different application components that interact dynamically, which can limit their effectiveness in detecting complex, multi-stage attacks.

When focusing on permissions, static analysis faces limitations due to the overly broad nature of permission requests. Android applications often request a wide range of permissions that may not be fully utilized, making it difficult to determine the true intent behind the permission usage. This can result in false assumptions regarding an app's behavior. Similarly, analysis of the manifest file is limited, as it only provides a high-level overview of declared components, such as activities, services, and intents, without reflecting how these components are actually used during execution. Malicious apps can declare benign-looking components while hiding harmful actions in less obvious parts of the code. The limitation with API calls based detection is that often if the APIs used are not related to any manifest file component, they act as noise in the detection process. Hence, it may lead to lower detection accuracy.

2.2 Dynamic Detection Model

Dynamic analysis opts for a different approach than static analysis. Instead of examining the code, it relies upon monitoring an application's behavior while it is running over any virtual or real CPU. As the name suggests, dynamic analysis is performed by analyzing the runtime behavior of applications, and the features analyzed during this process are called dynamic features, such as -

1. Network traffic- Network traffic refers to the flow of data across a network, encompassing all the communication that occurs between devices, servers, and other network elements. It includes data packets transmitted through various protocols, such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Hypertext Transfer Protocol (HTTP), Domain Name System (DNS), and others.

   In mobile network traffic, Transmission Control Protocol (TCP) flows play a crucial role in ensuring reliable and efficient communication between devices. TCP is a connection-

oriented protocol that facilitates the orderly and error-checked delivery of data between applications. TCP connections begin with a three-way handshake, where the sender and receiver exchange synchronization (SYN) and acknowledgment (ACK) packets to establish a connection. TCP connections are gracefully terminated using a four-way handshake, involving FIN (finish) and ACK flags. For example, Figure 2.5 is a snapshot of Wireshark-extracted TCP flows from the network traffic of *CRIDEX* malware application. Six flows out of over five thousand are displayed in the figure depicting the duration and movement of packets/bytes from source to destination.

TCP · 5223

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.2.108	54851	93.189.89.83	8080	10	3 kB	54827	5	1 kB	5	2 kB	638830.697094	0.0955	95 kbps	129 kbps
10.0.2.108	54852	62.75.184.70	8080	3	194 bytes	54828	3	194 bytes	0	0 bytes	638831.750195	8.9986	172 bits/s	0 bits/s
10.0.2.108	54853	93.189.89.83	8080	12	3 kB	54829	6	1 kB	6	2 kB	638853.761407	0.1434	68 kbps	94 kbps
10.0.2.108	54854	62.75.184.70	8080	3	194 bytes	54830	3	194 bytes	0	0 bytes	638854.863812	8.9987	172 bits/s	0 bits/s
10.0.2.108	54855	93.189.89.83	8080	12	3 kB	54831	6	1 kB	6	2 kB	638876.865093	0.1433	66 kbps	88 kbps
10.0.2.108	54856	62.75.184.70	8080	3	194 bytes	54832	3	194 bytes	0	0 bytes	638877.968479	9.0069	172 bits/s	0 bits/s

Figure 2.5: Snapshot of TCP flows from the network traffic of CRIDEX malware application.

2. System calls - In Android, system calls are essential functions that allow applications to interact with the underlying operating system kernel. These calls are typically denoted by unique identifiers or numbers, which correspond to specific operations or services provided by the kernel. System calls are found within the Android Runtime (ART) environment and are invoked by applications to perform various tasks such as file I/O operations, network communication, process management, and hardware interaction. For example, the *open()* system call is used by applications to open files, while the *socket()* system call is utilized for network socket creation. System calls allow real-time monitoring of app behavior, enabling the detection of malware activities as they happen.

Researchers have used other dynamic features also such as function call graphs [75], telemetry channels [76] and URL's [77] etc. for Android malware detection. These related studies can be further classified into two categories namely OS-based features, and Network Traffic features, which we explore in detail further.

2.2.1 OS-based detection

In this section, we review the works that have used features like system calls, CPU logs, user interaction, etc., for Android malware detection.

Dimjašević et al. [78] were driven by the goal of demonstrating that semantic information exists in system call sequences. They treated each system call sequence as a sentence in a language, constructing a classifier based on the Long Short-Term Memory (LSTM) language model. The authors in [79] highlighted the fact that feature selection improves the performance of existing approaches. Subsequently, they proposed a two-step approach based on the Rough set and another statistical test to extract the refined form of system calls. Chew et al. [80] focused on presenting a ransomware detection technique based on behaviors observed in the system calls performed by the malware. They then improved upon their initial approach to detect crypto-ransomware in real-time using a 2-layer token-based finite state machine streaming approach.

In [81], the authors introduced *EnDroid*, an innovative dynamic analysis framework that autonomously extracts various types of dynamic behavior features such as cryptographic operations, system calls, etc. to enhance malware detection. They utilized the chi-square feature selection algorithm to eliminate irrelevant or noisy features and identify crucial ones. Further, their proposed model employed Stacking to achieve effective malware detection. TaintDroid [82] model, based on dynamic taint analysis, analyzed the system calls sequences and tracked the flow of privacy-sensitive information through third-party apps. The authors, with system calls sequences, observed that many normal apps can leak the private information stored on the mobile device. Many systems such as ([83], [84]) are built on the TaintDroid model to detect the privacy leakage from the apps on the Android platform. Yang et al. [85] extended the TaintDroid model to detect the data leaks from the apps and also determine whether the leak is due to user intention or not. All these works focused on analyzing data-leaks from the apps, rather than detecting malicious apps. CopperDroid [86] model analyzed system calls of malware samples and described whether the malicious behavior is initiated from Java, JNI, or native code execution.

Afonso et al. [87] analyzed the combination of the dynamic API calls and system calls to detect malicious apps. CrowDroid [88] model extracted the system calls and made use of partitional clustering techniques to distinguish malicious apps from the normal ones. DroidTrace model [89], based on ptrace, monitored various dynamic features such as system calls sequences, file operations, network connections, etc., for malware detection. [90] analyzed run time traces of the apps like system calls, network traffic behavior, and real-time user inputs like users' interactions with the apps, etc. to evaluate the risk associated with the applications. Jang et al. [91] leveraged volatile memory acquisition to detect malicious Android applications.

2.2.2 Network Traffic-Based Detection

In the literature, several authors have focused on Android malware detection using network traffic features, this section reviews all such related works proposed in the literature.

In their study, Arora and Peddoju [92] opted to gather network traffic features using real smartphones instead of emulators. They highlighted the advantages of this approach, as it allowed them to obtain a comprehensive set of 22 network traffic features, which proved to be more than enough to achieve optimal detection accuracy. In order to further decrease the number of features and identify the most optimal set of characteristics, they employed information gain and statistical methods like chi-square to rank the feature set. The authors of the study [93] conducted their research by analyzing the network flow of applications. They discovered that certain network traffic features have distinct value ranges when comparing malware traffic to normal traffic. As a result, they selected the most significant and distinguishing network traffic features to develop a model for detecting malware.

In their study, Lashkari et al. [94] introduced nine novel flow-based network traffic characteristics. These features were used to develop an Android malware detection model that demonstrated high efficiency by utilizing only a minimal amount of features. The model was able to successfully detect unknown malware instances and accurately classify the type of malware. In order to determine the most optimal set of features from both existing and newly introduced ones, the researchers employed algorithms such as information gain, CFS subset, and SVM.

Shabtai et al. [95] tried to understand the reason behind the deviations in the application's network traffic behavior from the normal flow by observing the network traffic flows. The authors focus on the server side of the system and believe that a strong correlation exists between normal behavior patterns that can be used to detect abnormal activities. To conduct their research , the authors used eight Android devices, real malware, and self-developed malware.

Wang et al. [96] proposed an approach analyzing the text semantics of HTTP request headers and believed that it could be used to distinguish between normal and malware applications. To use the HTTP request header text as features, the authors used a natural language processing algorithm called N-gram and tried to reduce the feature set to only the most influential ones using a statistical chi-square test. For their testing phase, they used the SVM machine learning algorithm and varied the value of N in the N-gram to determine the best set of parameters.

Li et al. [97] aimed at mapping the network traffic features onto a high-dimensional matrix space for few-shot Android malware-encrypted network traffic classification. Simultaneously, to perform feature reduction, they carefully compressed the searching space using a metric learning framework called path optimization prototypical nets (POPNet). Hossain et al. [98] focused mainly on finding a robust solution to deal with the growing ransomware attacks using network traffic features. To deal with the huge dataset problem, they exploited particle swarm optimization to select only the optimal traffic characteristics. Furthermore, they used two machine learning classifiers to verify the detection accuracy of their proposed approach using a reduced set of features. The proposed PSO-assisted feature selection enabled the classifier to significantly improve detection accuracy. The authors in [99] devised a DL-based malware classification system using raw payload and CNNs. They concentrated on the raw payload of malware network traffic, leveraging the byte information to represent the behavioral patterns of malware and further treated the window selection algorithm's filtered (SWS) flows as documents to be processed by NLP methods.

The authors in [100] divided their work into two parts: creating a training model and a real-time detection model. For training , they incorporated several third-party scanning services to observe malicious behavior and train their model. They based their training and real-time detection model on DNS queries and HTTP requests and proved that their model produced better results than the integrated scanning services upon considering only DNS queries alone. Zulkifli et al. [101] directed their approach towards dynamic detection techniques based on network traffic, where traffic flows were extracted during application runtime. For the testing phase, they used the J48 decision tree machine learning algorithm to classify applications as normal or malware for two datasets.

Pang et al. [102] worked on creating an efficient and convenient network traffic collection system. The authors observed that the malware applications produced a negligible amount of network traffic data; hence, to tackle the irregularity issue in the two datasets, they tried to combine imbalance algorithms with the classical machine learning algorithms for their testing methods. Malik and Kaushal [103] proposed a detection system called CREDROID based on pattern-based detection, which analyzes the DNS queries and the information being transmitted to the remote server by an application to classify them as normal or malware. They considered four parameters: putting the app through VirusTotal to calculate an apk score, analyzing the reliability of the website the application is communicating with, the kind of information being sent, and the type of communication channel being used.

Wang et al. [104] analyzed multiple levels of network traffic features and emphasized that combining 2 levels, namely HTTP packet and TCP flow, can be successfully employed to create a lightweight server-based malware detection model. Lastly, on the belief that machine learning can be used to automatically discover the rules by analyzing the data, they applied machine learning algorithms on the training set and performed the testing experiments. Chen et al. [105] extracted a combination of time-related network flow and packet features and further used supervised machine learning algorithms to classify traffic as malware or normal. Mahdavifar et al. [106] emphasized the concern regarding the difficulty in accumulating well-labeled datasets and hence proposed a Deep Neural Network (DNN) based work to perform dynamic analysis for malware category classification. They further compared the detection results of their approach with those of a well-known semi-supervised machine learning technique called label propagation.

Liu et al. [107] suggested a Graphical Neural Network (GNN) model based on mobile network traffic that focused not only on node characteristics but also on edge attributes. After extracting the network traffic information, they updated the hidden state of each node and combined adjacency data with node attributes to construct the GNN model to further classify applications as normal or malware. Hamouda et al. [108] utilized the FDL to propose a deep learning-based malware detection model, in particular a convolutional neural network model to detect Android malware based on network behavior.

2.2.3 Limitations of Dynamic Analysis

Dynamic analysis, while a powerful tool for detecting malware through real-time behavior monitoring, presents several limitations that hinder its universal applicability. One primary shortcoming is its resource-intensive nature. Analyzing programs dynamically requires significant computational resources and time, making it less efficient for large-scale or real-time malware detection. Moreover, sophisticated malware variants can detect when they are being analyzed in a controlled environment, such as virtual machines or emulators, and alter their behavior to avoid detection. Furthermore, not all malware samples generate observable network traffic. Some malicious programs operate in more covert ways, such as by sending text messages or manipulating local files, without establishing connections to remote servers. Consequently, network traffic-based detection mechanisms fail to capture these activities.

At the same time, system call-based detection mechanisms, commonly employed in dynamic

analysis, face challenges in terms of platform and version dependency. The effectiveness of these methods can vary significantly across different operating system versions, making it difficult to generalize detection strategies.

Overall, dynamic analysis, while valuable in detecting real-time malware activity, is hampered by its resource demands, susceptibility to evasion techniques, and limited generalizability across platforms and malware types.

2.3 Hybrid Detection Model

Given the limitations of both static and dynamic analysis, hybrid analysis has emerged as a comprehensive approach to Android malware detection. Hybrid analysis combines the strengths of both static and dynamic methods, providing a more robust and resilient detection framework. Merging the two techniques, hybrid analysis enhances detection coverage, allowing for the identification of both known and zero-day threats while minimizing false positives.

Lastly, we now discuss some studies that have based their research on combining static and dynamic analysis techniques to propose a hybrid detection model. Wang et al. [109] analyzed permission sequences to build a static detection model for text-based binary classification. They further classified malware families by extracting memory features and constructing object reference graphs, demonstrating high-accuracy resistance to obfuscation attacks. Kang et al. [110] tried to overcome the limitations of using just permissions as a part of detection by using other features too such as API calls, Intents, serial number, file hash and system commands to carry out application classification. Further to classify the malware family too, they calculated the similarity score by using Needleman-Wunsch algorithm for the feature strings. The number of malware families are around a several hundred which poses as a limitation for detection to tackle with the zero day malware. Hence, Qiu et al. [111] used the security/privacy-related capabilities for each malware sample, which exists in around dozens only, to annotate the applications and build a feature vector table instead of using the malware family information while incorporating the TF-IDF technique.

The work done by the authors in [40] was threefold. In the first step, they filtered out the most relevant features from the entire set using mutual information gain. Then, they did the same for another feature set with top code-based features. Finally, they repeated this step with a combination of permissions and code-based features. Their conclusion proved that the hybrid features provided the best detection accuracy. The authors in [112] used a variety of

features such as permissions API calls and system commands to evaluate a machine learning based problem utilising the Bayesian classifier to analyse and detect malicious behaviour in unknown applications. To carry out the detection process they ranked the features on the basis of Mutual information score and lastly they computed their work by showcasing their detection results to prove that their approach produced optimum results. Zhu et al. [113] described their approach as a complete machine learning-based random forest approach, extracting four groups of features, namely permissions, permission rate, API calls, and system monitoring events. Finally, they compared the detection results of the RF classifier with those of the SVM classifier. Zhang et al. [114] performed a dynamic taint analysis technique to completely identify both explicit and implicit permissions use points to further build a permission use graph with the behavior profiler module of the proposed system. The profiled permission graphs were later used to capture the behaviors of using permissions inside an application. Yang et al. [115] proposed a method involving static analysis of permission-related API invocations and dynamic exploration to analyze permission-related behavior of the app using a locally exhaustive permission combination strategy that is also capable of simultaneously modifying permission combinations at runtime. The authors first constructed the State Transition Graphs (STGs) using the permissions and further fed them to the dynamic exploration module to implement the breadth-first search for UI exploration. Zhou et al. [116] extracted static features such as permissions, API calls and network addresses to feed the features into GRU model and improve the traditional GRU model to SimGRU which is based on the similarity principle, and propose three different GRU structures namely InputSimGRU, HiddenSimGRU and InputHiddenSimGRU.

Qaisar and Li [117] proposed a hybrid approach to extract and store features in a case base using case-based reasoning, which is a lazy learning approach capable of continuously learning. They used k means clustering to find similarities between features and detect malware behavior. Wang et al. [118] proposed a detection model using an ensemble of string-based and structural-based features, such as permissions, API calls, intents, hardware components, code patterns, and functional call graphs. To showcase the proposed model's detection results, they used various machine learning classifiers with a single feature type and an ensemble of both feature types. Arshad et al. [119] exploited the benefits of both static and dynamic analysis by proposing a hybrid detection model using permissions, hardware components, and various other features, including system calls and network addresses. The authors used both local and remote hosts to detect the malicious behavior of applications using several machine-learning

algorithms. Anupama et al. [120] proposed a hybrid methodology that combined permissions and system calls to create a detection model utilizing a range of machine learning and deep learning classifiers. Later, they aimed to analyze the working of classifiers in the case of adversarial samples. Surendran et al. [121] proposed a hybrid approach considering permissions and API calls from the static feature set and system calls from the dynamic feature set to study the co-dependency between the static and dynamic features. Lu et al. [122] extracted a bunch of resource features and semantic features as a part of static analysis and extracted a variety of dynamic features to build a hybrid deep-learning malware detection model. Due to the binary nature of static features, they were used as input for the Deep belief Network (DBN), and dynamic features were used as input for the Gated Recurrent Unit (GRU).

2.3.1 Limitations of Hybrid Analysis

As discussed earlier, both static and dynamic analysis have their inherent limitations. Since most of the hybrid works extract the OS-based dynamic features such as system calls, CPU logs, user logs, etc., high computation overhead is involved in the processing. Additionally, not all Android malware samples generate network traffic. It has been observed that certain types of malware may transmit text messages discreetly in the background, without producing any noticeable network traffic. Hence, network traffic-based detection mechanisms cannot detect such samples.

2.4 Multi-Category Malware Detection Model

While the threat of Android malware is well-recognized, it extends beyond basic detection. Malware can be categorized into distinct types, such as *Adware, Fraudware Trojans, Spyware, and Ransomware*. Thus, binary classification, which differentiates between benign and malicious samples, serves as an essential initial step. However, multi-category classification, which further identifies the specific type of malware is also an additional step taken up by many researchers using a range of static and dynamic features. This section reviews notable studies that have contributed to advancing multi-category classification in Android malware detection.

Abuthawabeh and Mahmoud [123] extracted the conversation-level network traffic features to build an Android malware detection system that was capable of not only binary and malware categorization but also classification of the detected malware family. For this, after cleaning

the data to remove the unwanted redundancies, they used the ensemble learning technique to choose the best set of features, followed by training and testing the data using the machine learning classifiers.

Feng et al. [124] divided their whole approach into two layers, wherein the first layer they focused mainly on the static features such as permissions, intent, etc. whereas, in the second layer, they focused on the dynamic features, i.e., the network traffic features of the applications. After the extraction of static features in the first layer, they fed them to the fully connected neural network and used the benign output applications for the next layer of extracting network traffic features. Results indicate the second layer which was a combination of convolutional autoencoder and neural network, was capable of classifying applications as normal or malware with the added benefit of detecting their corresponding malware category and family too. Imtiaz et al. [125] put forward a hybrid approach using static features in the first phase of their model and dynamic features in the second phase. They extracted permissions, intents, and API calls and fed the results to their proposed deep-learning artificial neural network model for malware classification and used network traffic flows for malware family classification. Feldman et al. [126] worked on using static features such as permissions, high-priority intent filter, and version numbers from the manifest file to build their detection model to classify nature and app's specific category, namely adware, spyware, and SMS malware. Their experimental results indicate that only permissions requests aren't sufficient to detect malware, hence, they extended their study to network traffic features by analyzing some malware applications such as HGSpy, Simplocker, and a Minimob variant.

The authors in [127] combined permissions with more than a single type of network traffic features to build their binary as well as family classification detection model. They mainly used DNS queries, TCP flows, HTTP packets, and other packet contents in addition to several machine learning classifiers. Ding et al. [128] proposed a hybrid approach combining static and dynamic analysis. For the first part of their approach, they extracted permissions and intents, followed by obtaining the best subset of static features using various feature selection and machine learning methods. For the second part, they shifted to dynamic analysis, converted the network traffic data into Mnist format, and fed it into their Res7LSTM model which combined residual network and a long short-term memory model. Their approach had the added advantage of not only binary classification but category and malware family classification as well.

2.5 Summary

In this chapter, we described the state of the art approaches for malware detection specifically Android-based smartphones. Regarding Android malware detection, we discussed the approaches in three categories namely: Static Detection, Dynamic Detection, and Hybrid Detection. Additionally, we reviewed some notable studies that have contributed to advancing multi-category classification in Android malware detection. We also highlighted the limitations of the existing works in each of the three categories. To overcome the aforementioned limitations, we proposed some solutions which are discussed subsequently in the upcoming chapters. Feature ranking over combinations of permissions and other manifest file components using a ranking-based algorithm is essential to identify the distinguishing features to obtain relatively good accuracy. The next problem addressed in the thesis is on prioritizing the network traffic features, among a large number of features, for Android malware detection. Lastly, we propose some hybrid detection models that efficiently overcome the limitations of standalone static and dynamic analysis methods.

Chapter 3

PHIGrader: Evaluating the effectiveness of Manifest file components in Android malware detection using Multi Criteria Decision Making techniques

---

Among the components in the *AndroidManifest* file, the most significant are permissions, intents, and hardware components. This chapter introduces *PHIGrader*, an Android malware detection system using a frequency-based *Multi-Criteria Decision-Making (MCDM)* approach to rank these static features. The aim is to identify the most effective feature type and feature set for Android malware detection. Section 3.1 outlines the motivation and methodology. Section 3.2 details the proposed methodology, while Section 3.3 discusses feature ranking results. Detection results for two datasets are presented in Sections 3.4 and 3.5. A comparison with existing literature, along with limitations, is covered in Section 3.6. The chapter concludes with future work directions in Section 3.7.

3.1 Introduction

Permissions, intents, and hardware components serve as critical static features within the *AndroidManifest* file, offering significant value for Android malware detection. Permissions regulate app access to sensitive data and system functionalities, making them pivotal in distinguishing benign from malicious applications. Intents capture inter-process communications, revealing app interactions that may signal malicious behavior. Hardware components indicate the device's capabilities the app can leverage, thus suggesting risk levels based on required resources. These features are widely employed in the literature, as their static nature provides insights without requiring app execution, making faster analysis. Although API calls are another frequently used static feature, their limitations arise from contextual ambiguity, where an API call alone may not reliably indicate intent without dynamic contextual analysis. This limitation underscores the advantage of permissions, intents, and hardware components as foundational features in static Android malware detection frameworks.

Motivation: Among all the components present within the manifest file of an Android application, the most important and influential are permissions, intents, and hardware components. These static features have been widely used in the literature for Android malware detection. However, there are many similarities in the feature usage patterns of normal and malicious apps. Tables 3.1 , 3.2 and 3.3 respectively, summarize the top 20 permissions, intents, and hardware components based on their frequency in the normal and malware datasets.

We collected 77,000 normal apps and an equal number of malware apps from *Androzoo*. More details about the dataset are discussed in the upcoming sections. Furthermore, we extracted permissions, intents, and hardware components from the manifest files of the corresponding applications. As shown in Table 3.1, 13 of the top 20 permissions are common in normal and malware datasets. Similarly, Table 3.2 and Table 3.3 highlights that seven out of the top 20 intents and 16 out of the 20 hardware components are common in both datasets. Such similarity in these features across both datasets motivates us to rank the features to propose an efficient detection model with distinguishing features. For instance, the Android operating system has more than 150 permissions; if we use all of them as features, irrelevant features may hamper detection accuracy. Hence, feature ranking is a key process in developing a detection algorithm.

Several related works, such as [70] , [129], and [118], have used static features to frame their Android malware detection models. If we take a closer look at them, we observe that Zhu

Table 3.1: Top 20 most frequently requested permissions from both normal and malware datasets with their corresponding frequency.

PERMISSIONS	**Normal Frequency**	**PERMISSIONS**	**Malware Frequency**
INTERNET	55063	INTERNET	55684
ACCESS_NETWORK_STATE	52391	ACCESS_NETWORK_STATE	55252
WRITE_EXTERNAL_STORAGE	38934	WRITE_EXTERNAL_STORAGE	54759
WAKE_LOCK	32527	ACCESS_WIFI_STATE	53886
ACCESS_WIFI_STATE	28554	READ_PHONE_STATE	53586
RECEIVE	23875	READ_EXTERNAL_STORAGE	46646
READ_EXTERNAL_STORAGE	22516	WAKE_LOCK	44003
VIBRATE	20472	GET_TASKS	43399
ACCESS_FINE_LOCATION	16968	CHANGE_WIFI_STATE	43165
ACCESS_COARSE_LOCATION	16650	ACCESS_COARSE_LOCATION	42425
RECEIVE_BOOT_COMPLETED	16519	VIBRATE	42325
CAMERA	14993	MOUNT_UNMOUNT_FILESYSTEMS	41324
READ_PHONE_STATE	14176	ACCESS_FINE_LOCATION	40720
C2D_MESSAGE	12342	WRITE_SETTINGS	39497
BIND_GET_INSTALL_REFERRER_SERVICE	10593	SYSTEM_ALERT_WINDOW	38594
BILLING	9905	CAMERA	36115
FOREGROUND_SERVICE	9587	CHANGE_NETWORK_STATE	30874
GET_ACCOUNTS	7806	RECEIVE_BOOT_COMPLETED	29441
WRITE_SETTINGS	7258	READ_LOGS	29112
BLUETOOTH	5820	RECORD_AUDIO	27010

et al. [70] chose static analysis to build their detection model using permissions and hardware components. They applied Convolutional Neural Network (CNN)-based multi-Head Squeeze and Excitation Residual block (MSer) on static features for malware detection. Rana and Sung [129] focused on static features such as permissions, intents, and other hardware components to create functions based on features that extract the most useful information to facilitate detection. Consequently, they developed a dictionary of these most useful features to generate a

Table 3.2: Top 20 most frequently requested intents from both normal and malware datasets with their corresponding frequency.

INTENTS	**Normal Frequency**	**INTENTS**	**Malware Frequency**
MAIN	55919	MAIN	55832
LAUNCHER	55902	LAUNCHER	55769
RECEIVE	22667	DEFAULT	45689
DEFAULT	21291	VIEW	35548
VIEW	18922	BROWSABLE	33915
BROWSABLE	17545	USER_PRESENT	33108
BOOT_COMPLETED	16510	PACKAGE_REMOVED	26806
REGISTRATION	8256	BOOT_COMPLETED	26645
ACTION_POWER_DISCONNECTED	7318	PACKAGE_ADDED	21111
ACTION_POWER_CONNECTED	6690	REGISTRATION	16609
LEANBACK_LAUNCHER	6171	REGISTER	14419
TIME_SET	5989	NOTIFICATION_RECEIVED_PROXY	14139
TIMEZONE_CHANGED	5937	PushService	14004
BATTERY_LOW	5798	REPORT	13998
BATTERY_OKAY	5788	PUSH_TIME	13998
DEVICE_STORAGE_LOW	5750	NOTIFICATION_OPENED	13019
DEVICE_STORAGE_OK	5748	MESSAGE_RECEIVED	13016
MEDIA_BUTTON	4932	NOTIFICATION_RECEIVED	12957
QUICKBOOT_POWERON	4730	DaemonService	12488
MY_PACKAGE_REPLACED	4131	CONNECTION	12252

Table 3.3: Top 20 most frequently requested hardware components from both normal and malware datasets with their corresponding frequency.

Hardware components	**Normal frequency**	**Hardware components**	**Malware frequency**
camera	12337	camera	21063
touchscreen	12147	Camera.autofocus	19080
Camera.autofocus	10446	camera.flash	3288
touchscreen.multitouch	8999	nfc.hce	2324
touchscreen.multitouch.distinct	8765	touchscreen	2101
location.GPS	7468	camera.front	2040
location.network	7103	wifi	1694
location	6223	touchscreen.multitouch	1334
screen.landscape	5002	location.GPS	1262
telephony	4725	touchscreen.multitouch.distinct	1256
wifi	4484	microphone	1219
screen.portrait	4136	screen.landscape	1196
sensor.accelerometer	3892	sensor.accelerometer	1111
vulkan	3235	bluetooth_le	855
camera.flash	2892	location.network	825
camera.front	2722	telephony	674
microphone	2216	autofocus	567
bluetooth	2194	location	413
bluetooth_le	1087	camera2.full	297
NFC	811	usb.action.USB_STATE	264

feature vector that could be fed into various classifiers. Wang et al. [118] proposed a detection model using an ensemble of string-based and structural-based features such as permissions, intents, hardware components, and code patterns. To showcase the detection results of the proposed model, various machine learning classifiers with a single feature type and an ensemble of both feature types were used.

None of the above works used the key concept of ranking the features and hence, missed the feature reduction step, which could have enhanced the quality of their results. However, in some works such as [63] and [130], the authors did rank the features and even the combination of features in some cases. Xie et al. [63] proposed an analysis-based approach to fingerprint Android malware families for describing the different behaviors. They extracted permissions, API calls, and hardware components, ranked them on the basis of the Fisher score and frequency-based methods, and chose the top 20 features to be used for fingerprinting. Wang et al. [130] ranked the features using the absolute frequency rate difference between the malware and benign datasets. In particular, they used permissions and hardware components to create a feature vector. However, both studies were implemented on a smaller set of applications compared with the huge dataset in our proposed work. More importantly, our work outperforms both in terms of detection accuracy.

Using static features such as permissions, intents, and hardware components has always been a simple yet effective approach to detect malicious applications. However, it all comes down

to choosing the right set of features and the feature type. Hence, in this chapter, we aim to analyze the effectiveness of the above-mentioned three most commonly used static features in Android malware detection while taking their frequency as weight inputs and further ranking them using a couple of *Multi-Criteria Decision Making* (*MCDM*) techniques. The following research questions emerge considering the proposed detection model based on the ranking of features:

- **RQ1** Where does the need for ranking the features arise and subsequently, what is the significance of feature reduction compared with feeding all the features as inputs at once?
- **RQ2** How to rank features, i.e., how to incorporate feature ranking?
- **RQ3** How to devise a detection approach using the ranked features?
- **RQ4** Which feature among the most commonly used *AndroidManifest* file components gives the best detection accuracy?

We are driven by the goal of answering the research questions mentioned above and at the same time forming an Android malware detector, named *PHIGrader*. We used a frequency-based *Multi-Criteria Decision-Making* (*MCDM*) approach to rank the three most commonly used static feature types, namely permissions, intents, and hardware components. We identified the best feature type and the best feature set for Android malware detection among the commonly used *AndroidManifest* file components. For this purpose, we applied three *MCDM* techniques individually to all three feature types. We attempted to implement the *MCDM* techniques because of their numerous advantages, such as a simple yet quick computing process and the ability to work with a vast dataset such as ours. Moreover, these techniques have a rational and comprehensive logic that works best when a fundamental ranking of alternatives is needed. Furthermore, we have proposed a novel detection algorithm that uses feature rankings formulated from frequency-based *MCDM* techniques and applies various machine learning and deep learning techniques to detect Android malware effectively. The work proposed in this chapter uses a mix of old and recent datasets for evaluation. Our detection results outperform several state-of-the-art techniques proposed in related areas of research. Moreover, our experiments indicate that the proposed frequency-based *MCDM* approach gives us better accuracy than the popularly used feature ranking methods such as Principal Component Analysis (PCA), Entropy-based Category Coverage Difference (ECCD) and also better than

other statistical tests such as mutual information, Pearson correlation coefficient, and T-test, which have been used in [131], when we evaluate them against the same dataset of normal and malware apps.

Contributions: The major contributions of this chapter are as follows:

- Initially, we ranked the three feature sets of permissions, intents, and hardware components individually in order of their frequency difference between the malware and normal training datasets to assign frequency-based weights to each feature.
- Next, we apply three *MCDM* techniques to the three weighted feature sets and rank them according to their preference score.
- We proposed a novel algorithm that uses the individual rankings of permissions, intents, and hardware components described by *MCDM* techniques to develop an efficient Android malware detection system.
- We recognized that the detection results of the proposed approach are better than those of various state-of-the-art techniques existing in the literature for Android malware detection.

3.2 System Design

In this section, we explain our proposed methodology in detail. Figure 3.1 summarizes a brief yet complete idea of our proposed model *PHIGrader*, which is divided mainly into two modules. We refer to the first module as the *Ranking Module*, which includes extracting features from the training dataset and ranking them using three *Multi-Criteria Decision-Making* (*MCDM*) techniques. Such a feature ranking eliminates irrelevant features and filters out only the influential ones. In the *Detection Module*, we propose a novel algorithm that applies machine learning and deep learning classifiers to obtain the best features that can provide higher detection accuracy. The following subsections discuss in detail both modules of the proposed model.

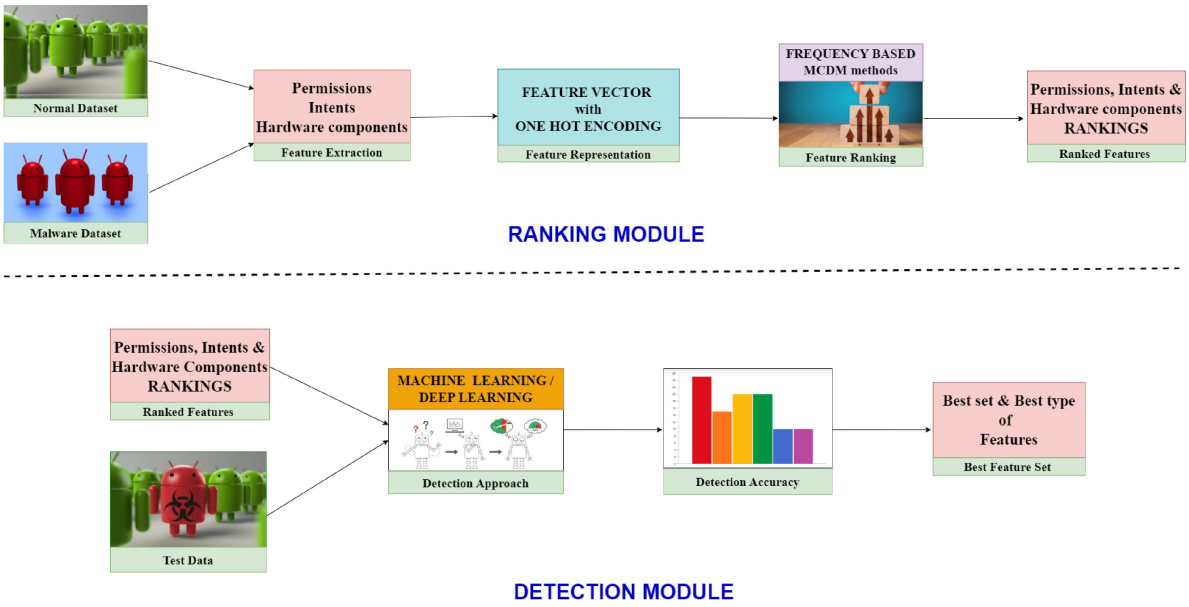


Figure 3.1: *PHIGrader* System Design

Ranking Module

3.2.1 Dataset

To begin with, we needed a vast dataset of normal and malware applications to conduct our research. For this purpose, we downloaded 77,000 normal and 77,000 malware applications from *Androzoo* [132] dated between 2012 and 2020. Of these, we used 56,000 normal apps and 56,000 malware apps in the *Ranking Module*. The remaining 21,000 normal and 21,000 malware apps were used in the *Detection Module*. We term this dataset as *DATASET-1*. In addition, we tested our approach on another unknown dataset containing more recent and stealthier malware samples detected between 2021 and 2022, named *DATASET-2*. The market used by *Androzoo* for normal applications is the Google Play Store, whereas the malware apps are from various sources such as *PlayDrone* , *appchina* , *anzhi*, and *VirusShare*. To create the normal dataset from *Androzoo*, we filtered out those apps that had VirusTotal [1] detection score of zero, i.e., the apps that were detected as malware by none of the antiviruses on VirusTotal. Furthermore, for the malware dataset, we filtered out those applications with a detection score of at least five, i.e., the apps detected as malware by at least five antiviruses on VirusTotal. With regard to the sizes of applications to be used, we settled on a range of APK sizes spanning from 1 byte to 8 GB to include applications of varying sizes and functionalities.

[1] https://www.virustotal.com/gui/home/upload

3.2.2 Feature Extraction

Android OS uses the Android Package Kit (APK) file format, which contains several sub-files and folders that further include essential information such as the application's permissions and. The most commonly used language for writing the source code of an Android application is Java. Subsequently, the Java source codes are compiled and converted into executable Dalvik bytecodes. Among the several important files present inside the bundle, one is the *AndroidManifest.xml* file, which contains three of the most important features used in our detection model: permissions, intents, and hardware components. The process of extracting such information from the kit is called decompilation.

We used the *Android Asset Packaging Tool (AAPT2)* tool [2] to extract the list of permissions, intents and hardware components from normal and malware applications. Finally, these three extracted lists of features, i.e., 129 permissions, 79 intents, and 88 hardware components were further used to generate a feature vector for each application in the feature representation process.

3.2.3 Feature Representation

After extracting the list of features from the applications of our dataset, we create feature vector tables for their representation. The extracted features are represented using the One Hot Encoding method [3] to generate a feature vector for each app in both normal and malware datasets separately. The feature vector developed for each app is of the binary type, with a *1* for the features that the application requests and a *0* for the features that are not present within that app. In this way, we create six separate vector tables, normal and malware, for permissions, intents, and hardware components represented by Pn_{VT} , Pm_{VT} , In_{VT} , Im_{VT} , Hn_{VT} and Hm_{VT}, respectively. For instance, if there are a total of five permissions, say $<P_1, \ldots..P_5>$ and five intents say $<I_1, \ldots..I_5>$ in the system, and any application A_j has permissions P_1, P_2, P_5 and intents I_3, I_4, I_5, then the app A_j is represented as *11001* and *00111* in Pn_{VT} *and* In_{VT} respectively.

We observe that some features have a high frequency in normal or malware datasets. The frequency difference (Δf) between the malware (M) and normal datasets (N) for any feature

[2] https://developer.android.com/studio/command-line/aapt2

[3] https://scikit-learn.org/stable/modules/generated/sklearn. preprocessing.OneHotEncoder.html

(f) can provide valuable insights for feature ranking. Therefore, before applying the *MCDM* techniques to rank the features, we initially assign weights ($w(f)$) to all permissions, intents, and hardware components separately based on their frequency difference in malware and normal datasets. We subtract the frequency count of every feature type separately in the normal dataset from that of the malware set, as highlighted in equation 3.2.1, and sort them in descending order.

$$\Delta f = \text{Freq}_M(f) - \text{Freq}_N(f) \tag{3.2.1}$$

Next, we take the newly assigned weights based on the frequency difference for each feature in the malware and normal datasets.

For instance, if there are N number of features, the top one-third of N, after ranking based on the frequency difference between malware and normal datasets, will be assigned a weight of 1 and considered as malware-dominant features. Similarly, the bottom one-third of N will be assigned a weight of 3 and considered as normal dominant features. The remaining one-third will be given a weight of 2 as they show neutral dominance or preference. Equation 3.2.3 depicts this relationship below -

$$w(f) = \begin{cases} 1 & \text{if } f \text{ is in top one-third of } N \\ 2 & \text{if } f \text{ is in middle one-third of } N \\ 3 & \text{if } f \text{ is in bottom one-third of } N \end{cases} \tag{3.2.2}$$

where N is the total number of ranked features.

After assigning weights to all features, for every occurrence of *1* for any feature, we replace *1* with its corresponding weight in all six vector tables. For instance, consider the same app A_j, which was initially represented as *11001* and *00111* in Pn_{VT} *and* In_{VT} respectively. Suppose the weights for P_1, P_2, *and* P_5 are 1, 2, and 3, respectively, and the weights for I_3, I_4 *and* I_5 are 3, 2, and 1, then A_j is represented as *12003* and *00321* in Pn_{VT} *and* In_{VT} respectively. These vector tables function as decision matrices for our *MCDM* methods.

3.2.4 Features Ranking

Multi-Criteria Decision Making (*MCDM*) is one of the most accurate methods of decision making. *MCDM* considers different qualitative and quantitative criteria that need to be fixed to find the best alternative or choose the best feature. In simple words, *MCDM* deals with structur-

ing and decision making when the data has manifold criteria and the decider needs to find the best alternative according to his/her preferences. The main steps of all *MCDM* problems are as follows: identifying the criteria, determining the weights for the criteria, and ranking the alternatives or features available in order of preference, followed by choosing the best or even opting for a subset from them. Furthermore, the goal of all *MCDM* problems is to define the alternatives or features as nondominant or influential. Several types of *MCDM* techniques work with a similar goal but differ in the complexity level of algorithms, weighting methods for criteria, way of representing preferences evaluation criteria, uncertain data possibility, and finally, data aggregation type. In our study, we used three different *MCDM* techniques to rank the extracted features individually in order of preference to depict the application of *MCDM* in Android malware detection. More information about the techniques used in this chapter is given below:

1. ***TOPSIS*** - *The Technique for Order of Preference by Similarity to Ideal Solution* (*TOPSIS*) [133] is a multi-criteria decision analysis method based on the assumption that the best alternative should have the least geometric distance from the Positive Ideal Solution (PIS) and the longest geometric distance from the Negative Ideal Solution (NIS). *TOPSIS* is used for comparing a set of alternatives by normalizing scores for each criterion, describing the geometric distance between each alternative and the ideal alternative, and finally, giving out the best alternative as the final result. The complete steps in a typical *TOPSIS* application are described below for an *MCDM* problem defined on m alternatives and n decision criteria:

   Step 1 - The normalization method to produce the normalized decision matrix r_{ij}.

$$r_{ij} = x_{ij}^2 / \sqrt{\sum_{1}^{n} x_{ij}^2} \tag{3.2.3}$$

   where x_{ij} is the performance value of alternative i when evaluated in terms of criterion j.
   Step 2 - The weights w_{ij} are assigned to various criteria according to their respective importance or contribution.

$$w_{ij}, \quad j = 1,2 \ldots n \tag{3.2.4}$$

$$\sum_{1}^{n} w_j = 1 \quad j = 1,2 \ldots n \tag{3.2.5}$$

Step 3 - The weighted normalized value v_{ij} can be computed by calculating the product of the normalized decision matrix r_{ij} and the associated weights w_{ij} with the formula shown in equation 3.2.6.

$$v_{ij} = w_{ij} \times r_{ij} \tag{3.2.6}$$

Step 4 - Determine the positive-ideal solution and negative-ideal solution.

$$A^* = \left\{ \left(\max v_{ij} \mid j \in J \right), \left(\min v_{ij} \mid j \in J' \right) \right\} \tag{3.2.7}$$

$$A^- = \left\{ \left(\min v_{ij} \mid j \in J \right), \left(\max v_{ij} \mid j \in J' \right) \right\} \tag{3.2.8}$$

for $J = 1,2,3 \ldots, n$, where J is associated with the benefit criteria. $J' = 1,2,3 \ldots, n$ where J' is associated with the cost criteria.

Step 5 - Calculate the separation measure. The separation of each alternative from the positive ideal is given by equation 3.2.9:

$$S_i^* = \sqrt{\sum_{1}^{n} \left(v_{ij} - v_j^* \right)^2 \; j = 1} \tag{3.2.9}$$

where i=1,2, . . . , m.

Similarly, the separation of each alternative from the negative ideal is given by equation 3.2.10:

$$S_i^- = \sqrt{\sum \left(v_{ij} - v_j^- \right)^2 \; j = 1} \tag{3.2.10}$$

where i=1,2, . . . , m.

Step 6 - Calculate the relative closeness to the ideal solution. The relative closeness of Ai with respect to A * is defined as follows:

$$C_i^* = S_i^- / \left(S_i^* + S_i^- \right), 0 \leq Ci^* \leq 1 \tag{3.2.11}$$

where i=1,2, . . . , m.

We assumed permissions as i (alternatives) and applications as j (criteria). Because each application contributes equally to the decision-making process, we assign equal weights (w_j) to all applications. To ensure that the summation of all weights is 1, each application is assigned a weight of 1 divided by the total number of applications, as highlighted in equation 3.2.12. This means that each application now contributes proportionally to the overall assessment, reflecting their equal significance in the problem domain.

$$w_j = \frac{1}{n}, \quad \sum_{j=1}^{n} w_j = 1 \tag{3.2.12}$$

The larger the C_i^* value (preference score), the better the performance of the alternatives.

$$\text{Performance}(i) \propto C_i^* \tag{3.2.13}$$

In our case, we applied *TOPSIS* on Pn_{VT} and Pm_{VT} vector tables separately using the weights w to compute the normal preference score (Cn_i^* values) and malware preference score (Cm_i^* values) for each permission feature.

$$Cn_i^* = TOPSIS(Pn_{VT}, w), \quad i = 1,2,\ldots,m \tag{3.2.14}$$

$$Cm_i^* = TOPSIS(Pm_{VT}, w), \quad i = 1,2,\ldots,m \tag{3.2.15}$$

Further, we calculate the difference (D_i) between the malware and normal preference scores for each permission.

$$D_i = Cm_i^* - Cn_i^*, \quad i = 1,2,\ldots,m \tag{3.2.16}$$

Note that the permission with the highest difference between malware and normal preference score will be the best and most preferred feature. Hence, in the last step, we rank the permissions in decreasing order of D_i values to obtain the *TOPSIS*-ranked permissions list.

$$\text{Rank}(i) = \text{sort}(D_i, \text{descending}) \tag{3.2.17}$$

We apply all the above-mentioned steps on In_{VT} and Im_{VT} too in order to compute the

TOPSIS-ranked intents list and similarly on Hn_{VT} and Hm_{VT} to compute the *TOPSIS*-ranked hardware components list.

2. ***EDAS*** - *Evaluation based on Distance from Average Solution* (*EDAS*) [134] is another commonly used *MCDM* technique. The output, i.e., the best alternative, of the *EDAS* is determined on the basis of the distances of the alternatives from an average solution. Moreover, as the average solution is determined with the help of the arithmetic mean, the *EDAS* method proves to be quite efficient in dealing with stochastic problems. The highest final normalized score *ASi* gives the best alternative of the proposed ones. The complete steps for an *EDAS* application are as follows:
Step 1 - After developing the decision matrix, determine the average solution according to all criteria as follows:

$$AV = [AV_j]_{1.:n} \tag{3.2.18}$$

where,

$$AV_j = \frac{\sum_{i=1}^{m} x_{ij}}{m} \tag{3.2.19}$$

Step 2 - Calculate the positive distance matrix $[PDA_{ij}]_{m\times n}$ from average and the negative distance matrix $[NDA_{ij}]_{m\times n}$ from the average matrices according to the type of criteria (benefit or cost) as follows:

if j th criterion is beneficial,

$$PDA_{ij} = \frac{\max(0,(x_{ij} - AV_j))}{AV_j}, \tag{3.2.20}$$

$$NDA_{ij} = \frac{\max(0,(AV_j - x_{ij}))}{AV_j}, \tag{3.2.21}$$

if j th criterion is cost,

$$PDA_{ij} = \frac{\max(0,(AV_j - x_{ij}))}{AV_j}, \tag{3.2.22}$$

$$NDA_{ij} = \frac{\max(0,(x_{ij} - AV_j))}{AV_j} \tag{3.2.23}$$

where PDA_{ij} and NDA_{ij} denote the positive and negative distance of i th alternative from the average solution in terms of j th criterion, respectively.

Step 3 - Determine the weighted sum of PDA and NDA (SP_i and SN_i) for all alternatives as follows:

$$SP_i = \sum_{j=1}^{n} w_j PDA_{ij} \tag{3.2.24}$$

$$SN_i = \sum_{j=1}^{n} w_j NDA_{ij} \tag{3.2.25}$$

where w_j is the weight of j th criterion.

Step 4 - Normalize the values of SP and SN for all alternatives, shown as follows:

$$NSP_i = \frac{SP_i}{\max_i (SP_i)}, \tag{3.2.26}$$

$$NSN_i = 1 - \frac{SN_i}{\max_i (SN_i)}, \tag{3.2.27}$$

Step 5 - Calculate the appraisal score or preference score (AS) for all alternatives as follows:

$$AS_i = \frac{1}{2} (NSP_i + NSN_i), \tag{3.2.28}$$

where $0 \leq AS_i \leq 1$.

The larger the AS_i value (preference score), the better the performance of the alternatives.

$$\text{Performance}(i) \propto AS_i \tag{3.2.29}$$

In our case, we applied *EDAS* on Pn_{VT} and Pm_{VT} vector tables separately using the weights w to compute the normal preference score (ASn_i values) and malware preference score (ASm_i values) for each permission feature while taking the same assumptions as *TOPSIS*, i.e., permissions as i (alternatives) and applications as j (criteria).

$$ASn_i = EDAS(Pn_{VT}, w), \quad i = 1, 2, \ldots, m \tag{3.2.30}$$

$$ASm_i = EDAS(Pm_{VT}, w), \quad i = 1, 2, \ldots, m \tag{3.2.31}$$

Further, we calculate the difference (D_i) between the malware and normal preference

scores for each permission.

$$D_i = ASm_i - ASn_i, \quad i = 1,2,\ldots,m \tag{3.2.32}$$

Note that the permission with the highest difference between malware and normal preference score will be the best and most preferred feature. Hence, in the last step, we rank the permissions in decreasing order of D_i values to obtain the *EDAS*-ranked permissions list.

$$\text{Rank}(i) = \text{sort}(D_i, \text{descending}) \tag{3.2.33}$$

We apply all the above-mentioned steps on In_{VT} and Im_{VT} too in order to compute the *EDAS*-ranked intents list and similarly on Hn_{VT} and Hm_{VT} to compute the *EDAS*-ranked hardware components list.

3. ***WASPAS*** - *The Weighted Aggregated Sum Product Assessment (WASPAS)* method is a unique combination of the *Weighted Sum Model (WSM)* and *weighted product model (WPM)*. *WASPAS* integrates the merits of both *WSM* and *WPM*, yet proves to be mathematically simple, due to which it is now widely accepted as an efficient decision-making tool. A ranking of alternatives is performed based on the value of combined optimality criteria computed according to the results of these two models. The method, by making a sensitivity analysis within its functioning, can check the consistency of alternative rankings. The complete steps in a typical *WASPAS* application are described below for a *MCDM* problem defined on m alternatives and n decision criteria.:

   Step 1 - The total relative importance of alternative i (where x_{ij} is it's performance value when it is evaluated in terms of criterion j)
   as per the WSM method, denoted by $Q_i^{(1)}$, is defined as:

$$Q_i^{(1)} = \sum_{j=1}^{n} \bar{x}_{ij} w_j \tag{3.2.34}$$

   where

$$\bar{x}_{ij} = \begin{cases} \frac{x_{ij}}{\max_i x_{ij}} \text{ if } \max_i x_{ij} \text{ is preferable} \\ \frac{\min_{ij} x_{ij}}{x_{ij}} \text{ if } \min_i x_{ij} \text{ is preferable} \end{cases} \tag{3.2.35}$$

   This is defined as the linearization of initial criteria values.

   Step 2 - The total relative importance of alternative i as per the *WPM* method, denoted

by $Q_i^{(2)}$, is defined as:

$$Q_i^{(2)} = \prod_{j=1}^{n} (\bar{x}_{ij})^{w_j}, \tag{3.2.36}$$

where $\bar{x}_{ij}$ is the linearization of initial criteria values as explained above.

Step 3 - The *Weighted Aggregated Sum Product Assessment* (*WASPAS*) method for ranking of alternatives is defined as:

$$Q_i = \lambda \sum_{j=1}^{n} \bar{x}_{ij} w_j + (1-\lambda) \prod_{j=1}^{n} (\bar{x}_{ij})^{w_j}, \tag{3.2.37}$$

where $\lambda = 0, 0.1, 0.2, \ldots, 1$

The larger the Q_i value (preference score), the better the performance of the alternatives. In our case, we applied *WASPAS* on Pn_{VT} and Pm_{VT} vector tables separately with an aim to compute the benign preference score (Q_i values) and malware preference score (Q_i values) for each permission feature. We assumed permissions as i (alternatives) and applications as criteria. Since each application contributes equally to the decision-making process, we give equal weights to all the applications. Moreover, we treat benign applications as benefit criteria whereas malware applications as non-beneficial or cost criteria. Consequently, after following **Steps 1- 3**, we successfully manage to compute two sets of Q_i values in the case of *WASPAS* using permissions, i.e, one for the benign dataset (Benign Preference score) and another for the malware dataset (Malware Preference score). Note that the permission with the highest difference between malware and benign preference score will be the best and most preferred feature. Hence, in the last step, we rank the permissions in decreasing order of difference value between preference scores (Malware Preference score - Benign Preference score) to give out the *WASPAS*-ranked permissions list.

We apply all the above-mentioned steps on In_{VT} and Im_{VT} too in order to compute *WASPAS*-ranked intents list and similarly, on Hn_{VT} and Hm_{VT} to compute *WASPAS*-ranked hardware components list.

3.2.5 Machine Learning and Deep Learning Classifiers

We used several machine learning and deep learning classifiers [135] in our detection approach. We applied ten widely used techniques, namely Decision Trees (DT), Random Forest

Algorithm 1 Proposed Malware Detection Algorithm

Input: $F_{List} \leftarrow$ Ranked feature List
Output: Best set of features with a higher detection rate
$BF_{List} \leftarrow$ Initialized as a copy of F_{List}
$F_i \leftarrow i^{\text{th}}$ ranked feature in F_{List}
$N \leftarrow$ Number of features in F_{List}
$F_{all} \leftarrow$ List of all features from testing dataset (non unique)
$D_{Max} \leftarrow$ Maximum accuracy obtained, initialized to zero.
$D_{Acc} \leftarrow$ Accuracy obtained after each iteration.
for i $\leftarrow$ 1 to N **do**
    Copy F_{N-i+1} in F_{List}
    $F_{all} \backslash \{F_N$ upto $F_{N-i+1}\}$ // Delete all $\{F_N$ upto $F_{N-i+1}\}$ from F_{all}
    Find D_{Acc} using ML algorithms for features present in F_{all}
    if $D_{Acc} > D_{Max}$ **then**
        $D_{Acc} = D_{Max}$
        $BF_{List} \backslash \{F_N$ upto $F_{N-i+1}\}$ // Delete all $\{F_N$ upto $F_{N-i+1}\}$ from BF_{List}
        else exit
    end if
end for
return D_{Max}
return BF_{List}

(RF), Bagging classifier (BC), Gaussian Naive Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM) as machine learning classifiers and Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Artificial Neural Networks (ANN), Dense Neural Network (DNN) as deep learning classifiers.

All experiments with these classifiers were performed using ten-fold cross-validation [136]. The code concludes by printing the cross-validation results, including the accuracy scores for each fold and the mean accuracy across all folds. This provides insights into the model's consistency and overall performance across diverse subsets of the dataset.

Detection Module

3.2.6 Proposed Malware Detection Algorithm

In response to our RQ3, i.e., how to devise a detection approach using the ranked features, this section describes our proposed detection algorithm termed Algorithm 1. As discussed in the previous subsection, we use the feature preference score computed separately for malware and normal datasets from a particular *MCDM* technique to further rank them in order of rele-

vance. The higher the difference between the preference score values, the higher the relevancy. We aim to find the best set of features to provide better detection accuracy. F_{List} represents the ranked features, i.e., permissions, intents, or hardware components in decreasing order of their computed difference.

$$F_{List} = \{f_1, f_2, \ldots, f_N\} \tag{3.2.38}$$

Since, we will need to modify the F_{List} based on the performance of the test set, we introduce another validation list with the name BF_{List} which will be initialized as a copy of F_{List} only. Eventually, after all iterations BF_{List} will give us the best set of features to provide better detection accuracy.

$$BF_{List} = F_{List} \tag{3.2.39}$$

In the first iteration of the algorithm, we select the bottom-ranked feature from F_{List}. We then execute machine learning and deep learning algorithms on the testing data after eliminating the bottom-ranked feature and considering only the rest of the features from the F_{List} and observe the detection accuracy, say D_{Acc}. The maximum accuracy, say D_{Max}, is initialized to zero. At every iteration, we compare D_{Acc} and D_{Max}. If the accuracy at the current iteration, i.e., D_{Acc}, is higher than D_{Max}, we proceed towards the next iteration and we set D_{Max} as D_{Acc} and at the same iteration only, we delete the bottom-ranked feature from the BF_{List} leaving *N-1* features in it. The following equations summarize the above-mentioned procedure.

$$\text{if } D_{Acc} > D_{Max} \tag{3.2.40}$$

$$\text{then } D_{Max} = D_{Acc} \tag{3.2.41}$$

$$\text{and } BF_{List} = \{f_1, f_2, \ldots, f_{N-1}\} \tag{3.2.42}$$

In the next iteration, we select the bottom two ranked features and find the detection accuracy on the testing data by eliminating these two and considering the rest *N-2* features only, i.e., D_{Acc} for the current iteration. Again, we compare the D_{Max} and D_{Acc}, and if D_{Acc} is higher than D_{Max}, we delete the bottom two ranked features from the BF_{List} and proceed to the next iteration to select the bottom three ranked features. The following equations summarize the above-mentioned procedure.

$$\text{if } D_{Acc} > D_{Max} \tag{3.2.43}$$

$$\text{then } D_{Max} = D_{Acc} \tag{3.2.44}$$

$$\text{and } BF_{List} = \{f_1, f_2, \ldots, f_{N-2}\} \tag{3.2.45}$$

The algorithm continues in the same manner and terminates when the detection accuracy does not improve further. At a stage when D_{Acc} is not higher than D_{max}, we return the D_{Max} and BF_{List} containing only the best set of ranked features. The overall computational complexity of the proposed algorithm can be approximated as O(N * (N + M + f(n))), where N is the number of features in the F_{List}, M is the size of the F_{all} list, and f(n) represents the time complexity of the ML algorithms used for training and evaluation.

Algorithm 1 answers research question three, i.e., how to frame a detection approach based on the ranking of features. We describe the results obtained from the proposed approach in the next section.

3.3 Feature Ranking Results

In this section, we present and discuss the feature ranking results obtained using the proposed *PHIGrader* model. We point out that we have separate datasets for training and testing. As described in Section 3.2.1, there are 77,000 applications, each in the normal and malware categories. Of these, we used 56,000 normal apps and 56,000 malware apps in the ranking module. The remaining 21,000 normal and 21,000 malware apps were used in the detection module. We term this dataset *DATASET-1*. In addition, we tested our approach on another unknown dataset containing more recent and stealthier malware samples detected between 2021 and 2022, named *DATASET-2*. In the upcoming subsections, we first discuss the ranking obtained from the three *MCDM* techniques individually, namely *TOPSIS*, *EDAS* and *WASPAS*, after allotting weights to all three feature types. Thereafter, in the subsequent sections, we discuss the detection results of *DATASET-1* and *DATASET-2*.

3.3.1 Allotting Weights To The Features

As discussed in Section 3.2.3, we first assign weights to features based on their frequency difference in the malware and normal training datasets. We note that we have three separate rankings, one each for permissions, intents, and hardware components. Tables 3.4 , 3.5 and 3.6 summarize the top ten normal dominant and malware dominant features along with their assigned weights. As seen from Table 3.4, the normal dominant permission named

RECEIVE is assigned a weight of three because it has the lowest frequency difference between the malware and normal datasets, whereas the malware dominant permission named *MOUNT_UNMOUNT_FILESYSTEMS* had the highest frequency difference between the malware and normal datasets; hence, it was weighted one. Similarly, we can acknowledge the weights of the other top 10 normal dominant and malware dominant permissions from the table.

Table 3.4: Top 10 normal dominant and malware dominant permissions with their corresponding weights

Normal dominant Permissions	**Weights alloted**	**Malware dominant Permissions**	**Weights alloted**
RECEIVE	3	MOUNT_UNMOUNT_FILESYSTEMS	1
BIND_GET_INSTALL_ REFERRER_SERVICE	3	READ_PHONE_STATE	1
C2D_MESSAGE	3	GET_TASKS	1
FOREGROUND_SERVICE	3	CHANGE_WIFI_STATE	1
BILLING	3	SYSTEM_ALERT_WINDOW	1
USE_FINGERPRINT	3	WRITE_SETTINGS	1
READ_GSERVICES	3	CHANGE_NETWORK_STATE	1
USE_BIOMETRIC	3	READ_LOGS	1
UPDATE_SHORTCUT	3	ACCESS_COARSE_LOCATION	1
BROADCAST_BADGE	3	ACCESS_WIFI_STATE	1

As seen from Table 3.5, the normal dominant intent named *RECEIVE* is assigned a weight of three because it has the lowest frequency difference between the malware and normal datasets, whereas the malware dominant intent named *USER_PRESENT* had the highest frequency difference between the malware and normal datasets; hence, it had a weight of one. Similarly, we can acknowledge the weights of the other top 10 normal dominant and malware dominant intents from the table.

Table 3.5: Top 10 normal dominant and malware dominant intents with their corresponding weights

Normal dominant Intents	**Weights alloted**	**Malware dominant Intents**	**Weights alloted**
RECEIVE	3	USER_PRESENT	1
DEVICE_STORAGE_LOW	3	PACKAGE_REMOVED	1
DEVICE_STORAGE_OK	3	DEFAULT	1
LEANBACK_LAUNCHER	3	PACKAGE_ADDED	1
BATTERY_OKAY	3	VIEW	1
BATTERY_LOW	3	BROWSABLE	1
MEDIA_BUTTON	3	REGISTER	1
MY_PACKAGE_REPLACED	3	NOTIFICATION_RECEIVED_ PROXY	1
TIMEZONE_CHANGED	3	PushService	1
QUICKBOOT_POWERON	3	PUSH_TIME	1

Similarly, it can be seen from Table 3.6, the normal dominant hardware component named *touchscreen* is assigned a weight of three because it has the lowest frequency difference between the malware and normal datasets, whereas the malware dominant component named *camera* had the highest frequency difference between the malware and normal dataset, hence, had a

weight of one. Similarly, we can acknowledge the weights of the other top 10 normal-dominant and malware-dominant hardware components from the table.

Table 3.6: Top 10 normal dominant and malware dominant hardware components with their corresponding weights

Normal dominant Hardware components	**Weights alloted**	**Malware dominant Hardware components**	**Weights alloted**
touchscreen	3	camera	1
touchscreen.multitouch	3	Camera.autofocus	1
touchscreen.multitouch.distinct	3	nfc.hce	1
location.network	3	autofocus	1
location.GPS	3	camera.flash	1
location	3	camera2.full	1
telephony	3	usb.action.USB_STATE	1
screen.portrait	3	sensor.stepcounter	1
screen.landscape	3	sensor.stepdetector	1
vulkan	3	camera.setParameters	1

3.3.2 Features Ranking

In response to our RQ2, i.e., how to incorporate feature ranking, this section presents the various techniques chosen by us to rank the features in order of relevance. To obtain the preference score of each feature, we separately applied the three *MCDM* techniques, *TOPSIS*, *EDAS*, and *WASPAS*, to the six vector tables (two for each feature type) developed for permissions, intents, and hardware components. Furthermore, we used the difference between the preference score of the features obtained using the malware and normal datasets to identify the most distinguishing features.

Using the frequency-based *MCDM* approach mentioned above, we answer research question two, i.e., how to rank the features to recognize the most distinguishing and influential ones among them.

Feature ranking using *TOPSIS*

In this section, we discuss the ranking obtained on applying *TOPSIS* over permissions, intents, and hardware components individually. Tables 3.7 , 3.8 and 3.9 summarize the top ten permissions, intents, and hardware components respectively according to the ranking done using the preference score obtained by *TOPSIS*.

Table 3.7 highlights that the permission named *UPDATE_APP_OPS_STATS* is the most distinguishing permission with the highest difference between the malware and normal preference score according to *TOPSIS*. Similarly, we can infer rankings of other permissions based on

their scores from the table. The permission named *INTERNET* had the lowest preference score difference value of -0.9945 amongst all permissions and hence, is the least distinguishing permission.

Table 3.7: Top 10 permissions ranked using *TOPSIS*

Permissions	Malware Preference score	Normal Preference score	Difference
UPDATE_APP_OPS_ STATS	0.995640016	0.994700764	0.000939252
USE_BIOMETRIC	0.999710875	0.999859006	-0.000148131
MAPS_RECEIVE	0.999401252	0.99984433	-0.000443078
READ_OWNER_DATA	0.997612157	0.998838311	-0.001226154
READ_USER_ DICTIONARY	0.997566566	0.999204898	-0.001638332
SEND_DOWNLOAD_ COMPLETED_INTENTS	0.995754181	0.997521968	-0.001767787
QUERY_ALL_ PACKAGES	0.99792082	0.999770741	-0.001849921
RECEIVE_WAP_PUSH	0.996069212	0.998509435	-0.002440223
BIND_GET_INSTALL_ REFERRER_SERVICE	0.996692437	0.999937243	-0.003244807
READ_SYNC_STATS	0.995959198	0.999400858	-0.00344166

Table 3.8 highlights that the intent named *UNREGISTRATION* is the most distinguishing intent with the highest difference between the malware and normal preference score according to *TOPSIS*. Similarly, we can infer rankings of other intents based on their scores from the table. The intent named *MAIN* had the lowest preference score difference value of -0.9362 and hence, is the least distinguishing intent.

Table 3.8: Top 10 intents ranked using *TOPSIS*

Intents	Malware Preference score	Normal Preference score	Difference
UNREGISTRATION	0.932365913	0.002912326	0.929454
ELECTION_RESULT_ V4	0.928431158	0	0.928431
webview	0.932733054	0.007804123	0.924929
PING_V4	0.921531166	0.002528772	0.919002
MEDIA_CHECKING	0.927302209	0.008587014	0.918715
COCKROACH	0.922930032	0.005779095	0.917151
action	0.921980242	0.006800472	0.91518
ACTION_RICHPUSH_CALLBACK	0.91729704	0.004018638	0.913278
ACTION_VIEW_ DOWNLOADS	1	0.087523518	0.912476
ELECTION	0.919571962	0.007494131	0.912078

In a similar manner, Table 3.9 highlights that the hardware component named *faketouch. multitouch. jazzhand* is the most distinguishing hardware component with the highest difference between the malware and normal preference score according to *TOPSIS*. Similarly, we can infer rankings of other hardware components based on their scores from the table. The hardware component named *camera* had the lowest preference score difference value of 0.0470 amongst all hardware components and hence, is the least distinguishing one.

Table 3.9: Top 10 hardware components ranked using *TOPSIS*

Hardware components	Malware Preference score	Normal Preference score	Difference
faketouch.multitouch. jazzhand	1	0.000760768	0.999239
sensor.ambient_ temperature	1	0.000760768	0.999239
sensor.heartrate.ecg	1	0.000760768	0.999239
sensor.relative_humidity	1	0.000760768	0.999239
type.automotive	1	0.000760768	0.999239
portrait	1	0.001923007	0.998077
BLUETOOTH_ADMIN	1	0.002172686	0.997827
sensor.heartrate	1	0.002221286	0.997779
sensor. ACCELEROMETER	0.996798316	0	0.996798
type.watch	0.99626415	0	0.996264

Feature ranking using *EDAS*

In this section, we discuss the ranking obtained on applying *EDAS* over permissions, intents and hardware components individually. Tables 3.10 , 3.11 and 3.12 summarize the top ten permissions, intents, and hardware components respectively according to the ranking done using the preference score obtained by *EDAS*. Table 3.10 highlights that the permission named *READ_OWNER_DATA* is the most distinguishing permission with the highest difference between the malware and normal preference score according to *EDAS*. Similarly, we can infer rankings of other permissions based on their scores from the table. The permission named *INTERNET* had the lowest preference score difference value of -0.9949 amongst all permissions and hence, is the least distinguishing permission.

Table 3.10: Top 10 permissions ranked using *EDAS*

Permissions	Malware Preference score	Normal Preference score	Difference
READ_OWNER_DATA	0.988485224	0.000817483	0.987667742
SEND_DOWNLOAD_ COMPLETED_INTENTS	0.987399804	0.000148769	0.987251036
WRITE_OWNER_DATA	0.988159244	0.001161826	0.986997418
UPDATE_APP_OPS_ STATS	0.986864022	7.69131E-05	0.986787109
READ_USER_ DICTIONARY	0.987964382	0.001516245	0.986448136
DEVICE_POWER	0.987416607	0.001680802	0.985735805
READ_SYNC_STATS	0.987797896	0.002401552	0.985396344
RECEIVE_WAP_PUSH	0.985181748	0.000569173	0.984612576
RECEIVE_MCS_ MESSAGE	0.985653997	0.001409919	0.984244078
QUERY_ALL_ PACKAGES	0.996960967	0.012767359	0.984193608

Table 3.11 highlights that the intent named *SEND_MULTIPLE* is the most distinguishing intent with the highest difference between the malware and normal preference score according to *EDAS*. Similarly, we can infer rankings of other intents based on their scores from the table. The intent named *MAIN* had the lowest preference score difference value of -0.9993 and hence, is the least distinguishing intent.

In a similar manner, Table 3.12 highlights that the hardware component named *faketouch. multitouch. jazzhand* is the most distinguishing hardware component with the highest difference

Table 3.11: Top 10 intents ranked using *EDAS*

Intents	Malware Preference score	Normal Preference score	Difference
SEND_MULTIPLE	0.994820911	0.008634884	0.986186
webview	0.986137754	0.000184527	0.985953
MESSAGE_CLICKED	0.986222594	0.000342747	0.98588
MESSAGE_ARRIVED	0.986208426	0.000342747	0.985866
ELECTION_RESULT_ V4	0.984945278	0	0.984945
DATE_CHANGED	0.985482814	0.001746364	0.983736
action	0.983561978	0.000130295	0.983432
BATTERY_CHANGED	0.984495175	0.001219489	0.983276
MEDIA_CHECKING	0.983434556	0.000225724	0.983209
COCKROACH	0.982778387	8.94315E-05	0.982689

between the malware and normal preference score according to *EDAS*. Similarly, we can infer rankings of other hardware components based on their scores from the table. The hardware component named *camera* had the lowest preference score difference value of -0.5659 amongst all hardware components and hence, is the least distinguishing one.

Table 3.12: Top 10 hardware components ranked using *EDAS*

Hardware components	Malware Preference score	Normal Preference score	Difference
faketouch.multitouch. jazzhand	1	2.56585E-05	0.999974342
sensor.ambient_ temperature	1	2.56585E-05	0.999974342
sensor.heartrate.ecg	1	2.56585E-05	0.999974342
sensor.relative_humidity	1	2.56585E-05	0.999974342
type.automotive	1	2.56585E-05	0.999974342
type.watch	0.999961715	0	0.999961715
sensor. ACCELEROMETER	0.999948631	0	0.999948631
BLUETOOTH_ADMIN	1	5.26423E-05	0.999947358
portrait	1	6.60416E-05	0.999933958
sensor.heartrate	1	7.62598E-05	0.99992374

Feature ranking using *WASPAS*

In this section, we discuss the ranking obtained on applying *WASPAS* over permissions, intents, and hardware components individually. Tables 3.13 , 3.14 and 3.15 summarize the top ten permissions, intents, and hardware components respectively according to the ranking done using the preference score obtained by *WASPAS*.

Table 3.13 highlights that the permission named *READ_OWNER_DATA* is the most distinguishing permission with the highest difference between the malware and normal preference score according to *WASPAS*. Similarly, we can infer rankings of other permissions based on their scores from the table. The permission named *INTERNET* had the lowest preference score difference value of -0.0125 amongst all permissions and hence, is the least distinguishing permission.

Table 3.14 highlights that the intent named *MESSAGE_ARRIVED* is the most distinguishing intent with the highest difference between the malware and normal preference score according

Table 3.13: Top 10 permissions ranked using *WASPAS*

Permissions	**Malware Preference score**	**Normal Preference score**	**Difference**
READ_OWNER_DATA	0.996748521	0.000165107	0.996583414
READ_USER_ DICTIONARY	0.996582966	0.00031986	0.996263106
SEND_DOWNLOAD_ COMPLETED_INTENTS	0.9955652	3.78087E-05	0.995527391
UPDATE_APP_OPS_ STATS	0.995327441	1.53057E-05	0.995312135
RECEIVE_WAP_PUSH	0.99538727	0.000113697	0.995273572
QUERY_ALL_ PACKAGES	0.998449011	0.003227937	0.995221074
WRITE_OWNER_DATA	0.995397982	0.000315789	0.995082193
READ_SYNC_STATS	0.995485714	0.000564408	0.994921306
DEVICE_POWER	0.994571291	0.000457666	0.994113625
WRITE_CALL_LOG	0.994184062	0.00017898	0.994005082

to *EDAS*. Similarly, we can infer rankings of other intents based on their scores from the table. The intent named *BATTERY_CHANGED* had the lowest preference score difference value of 0.2245 and hence, is the least distinguishing intent.

Table 3.14: Top 10 intents ranked using *WASPAS*

Intents	**Malware Preference score**	**Normal Preference score**	**Difference**
MESSAGE_ARRIVED	0.999450188	9.7658E-05	0.99935253
ELECTION_RESULT_ V4	0.998899696	0	0.998899696
SCREEN_ON	0.998930943	0.000607229	0.998323714
webview	0.998205727	4.08075E-05	0.99816492
HOME	0.99806392	0.001219068	0.996844852
PUSH_TIME	0.996837919	0.000104389	0.996733531
NOTIFICATION_ OPENED	0.996030541	5.25256E-05	0.995978015
PushService	0.993311291	0.000104389	0.993206902
DATE_CHANGED	0.993531221	0.000346466	0.993184755
action	0.992930463	2.93376E-05	0.992901125

In a similar manner, Table 3.15 highlights that the hardware component named *faketouch. multitouch. jazzhand* is the most distinguishing hardware component with the highest difference between the malware and normal preference score according to *WASPAS*. Similarly, we can infer rankings of other hardware components based on their scores from the table. The hardware component named *sensor.compass* had the lowest preference score difference value of -0.0004 amongst all hardware components and hence, is the least distinguishing one.

In the subsequent sections, we present the detection results obtained using the proposed model.

Table 3.15: Top 10 hardware components ranked using *WASPAS*

Hardware components	**Malware Preference score**	**Normal Preference score**	**Difference**
faketouch.multitouch. jazzhand	1	2.04694E-06	0.999998
sensor.ambient_ temperature	1	2.04694E-06	0.999998
sensor.heartrate.ecg	1	2.04694E-06	0.999998
sensor.relative_humidity	1	2.04694E-06	0.999998
type.automotive	1	2.04694E-06	0.999998
BLUETOOTH_ADMIN	1	5.81445E-06	0.999994
portrait	1	6.34124E-06	0.999994
sensor.heartrate	1	7.67137E-06	0.999992
faketouch.multitouch. distinct	1	1.90371E-05	0.999981
touchscreen.multitouch. jazz-hand	1	2.04244E-05	0.99998

3.4 Detection Results on *DATASET-1*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over *DATASET-1*. To check the efficiency of the three most commonly used features present in the *AndroidManifest* file, we performed three experiments, considering 1) permissions, 2) intents, and 3) hardware components, by applying the three *MCDM* techniques individually. We will discuss these results in the upcoming subsections, followed by a comparison of our proposed work with other statistical tests.

3.4.1 Detection Results with *TOPSIS*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over the *DATASET-1* while using the rankings given by *TOPSIS*. Figures 3.2 , 3.3 and 3.4 summarize the detection results when we consider permissions, intents, and hardware components for detection, respectively. We note that in the figures mentioned above, we do not mention the names of all the ranked features because the accuracy upon eliminating them lies within similar ranges to the mentioned ones.

Detection Results with permissions

Figure 3.2 summarizes the detection results when we rank the permissions using the *TOPSIS* technique and further apply the proposed detection algorithm. The figure can be understood as follows. While simultaneously considering all permissions without using the *TOPSIS* ranking,

we achieve 74.64% accuracy with the DT classifier. In the first iteration, on eliminating the least ranked permission named *INTERNET* from the *DATASET-1*, we observe that we get 75.46% accuracy with several machine learning classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked permissions, i.e., *INTERNET* and *ACCESS_NETWORK_STATE* from the *DATASET-1*. In this iteration, we obtain an accuracy of 76.47% with DT and RF classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked permissions and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.2, we achieved the highest detection accuracy of 98.01% with DT classifier upon eliminating 114 permissions out of the total lot of 129, i.e., upon considering only the top 15 permissions namely {*UPDATE_APP_OPS_STATS* , *USE_BIOMETRIC* , *MAPS_RECEIVE* , *READ_OWNER_DATA* , *READ_USER_ DICTIONARY* , *SEND_DOWNLOAD_COMPLETED_INTENTS* , *QUERY_ALL_ PACKAGES* , *RECEIVE_WAP_PUSH* , *BIND_GET_INSTALL_REFERRER_SERVICE* , *READ_SYNC_STATS* , *MESSAGE* , *WRITE_CALL_LOG* , *BAIDU_LOCATION_SERVICE* , *WRITE_OWNER_DATA*, and *BADGE_COUNT_READ* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 98.01% when we apply the proposed Algorithm 1 to permissions.

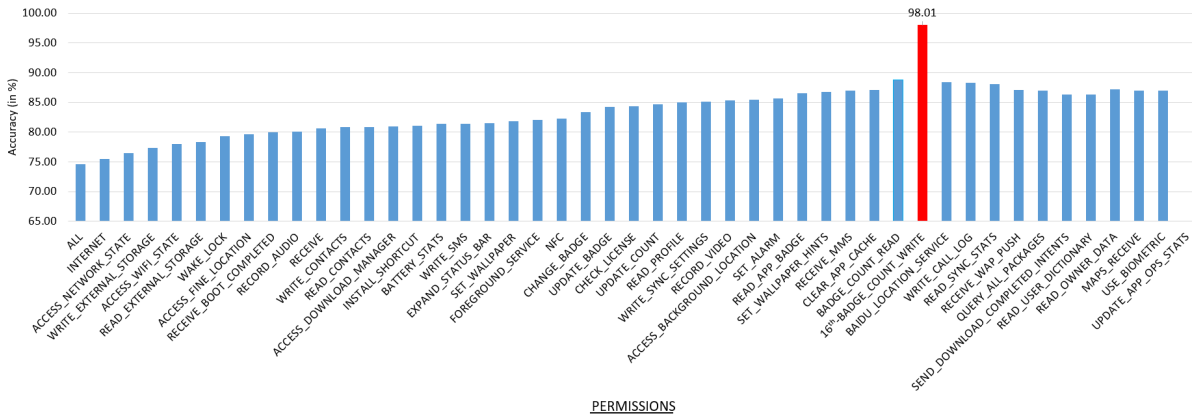


Figure 3.2: Detection results with *TOPSIS* using permissions

Detection Results with intents

Next, we apply the proposed detection algorithm (Algorithm 1) with intents ranked by *TOPSIS*. The algorithm provides the best intents with higher accuracy as an output. Figure 3.3 can be understood as follows. While simultaneously considering all intents without using the *TOPSIS* ranking, we achieve 67.19% accuracy with several machine learning classifiers. In the first it-

eration, after eliminating the least ranked intent named *MAIN* from the *DATASET-1*, we observe that we obtain the same 67.19% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked intents, i.e., *MAIN* and *LAUNCHER* from the *DATASET-1*. In this iteration, we obtain an accuracy of 68.79% with DT, RF, and NB classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked intents and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.3, we achieved the highest detection accuracy of 99.10% with DT classifier upon eliminating 33 intents out of the total lot of 79, i.e., upon considering the top 46 intents, some top ranked intents being { *UNREGISTRATION* , *ELECTION_RESULT_V4* , *WEBVIEW* , *PING_V4* , *MEDIA_CHECKING*.....*PUSH_TIME* , *PUSHSERVICE* , *REPORT* , *NOTIFICATION_RECEIVED_PROXY* , *REGISTER* }, the highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 99.10% when we apply the proposed Algorithm 1 to the intents.

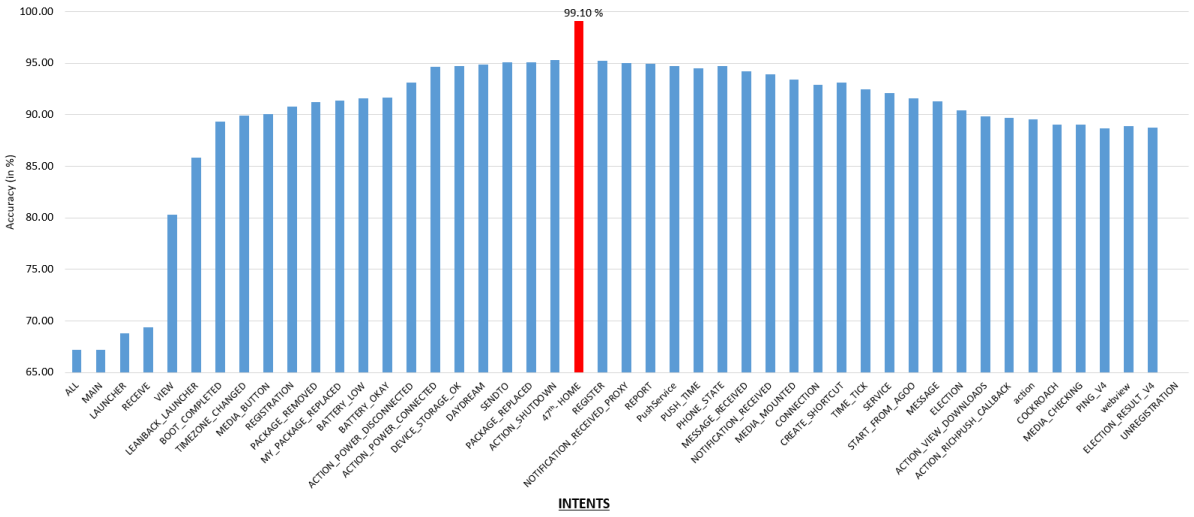


Figure 3.3: Detection results with *TOPSIS* using intents

Detection Results with hardware components

Next, we apply the proposed detection algorithm (Algorithm 1) with the hardware components ranked by *TOPSIS*. The algorithm provides the best hardware components with higher accuracy as an output. Figure 3.4 can be understood as follows. While considering all the hardware components simultaneously without using the *TOPSIS* ranking, we achieve 71.84% accuracy with several machine learning classifiers. In the first iteration, after eliminating the

least ranked hardware component named *camera* from the *DATASET-1*, we observe that we get 73.93% accuracy with the RF and BC classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked hardware components, i.e., *camera* and *Camera.autofocus* from the *DATASET-1*. In this iteration, we obtained an accuracy of 75.55% with DT, RF, and BC classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked hardware components and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.4, we achieved the highest detection accuracy of 91.67% with NB and LR classifiers upon eliminating 69 hardware components out of the total lot of 88, i.e., upon considering only the top 19 hardware components namely {*faketouch.multitouch.jazzhand* , *sensor.ambient_temperature* , *sensor.heartrate.ecg* , *sensor.relative_humidity* , *type.automotive* , *portrait* , *BLUETOOTH_ADMIN* , *sensor.heartrate* , *sensor.ACCELEROMETER* , *type.watch* , *sensor.hifi_sensors* , *faketouch.multitouch.distinct* , *touchscreen.multitouch.jazzhand* , *camera.capability.manual_post_processing* , *camera.capability.manual_sensor* , *READ_EXTERNAL_STORAGE* , *RECORD_AUDIO* , *camera.external* and *opengles.aep* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 91.67% when we apply the proposed Algorithm 1 to hardware components.

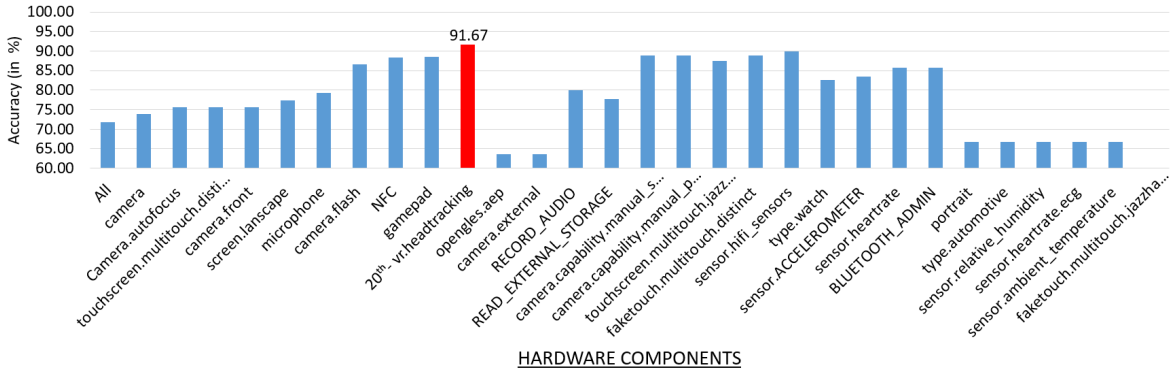


Figure 3.4: Detection results with *TOPSIS* using hardware components

3.4.2 Detection Results with *EDAS*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over *DATASET-1* using the *EDAS* rankings. Figure 3.5 , 3.6, and 3.7 summarize the detection results when we consider permissions, intents, and hardware components for

detection. We note that in the figures mentioned above, we do not mention the names of all the ranked features because the accuracy upon eliminating them lies within similar ranges to the mentioned ones.

Detection Results with permissions

Figure 3.5 summarizes the detection results when we apply the proposed algorithm to permissions ranked using the *EDAS* technique. The figure can be understood as follows. While considering all permissions simultaneously without using the *EDAS* ranking, we achieve 74.64% accuracy with the DT classifier. In the first iteration, on eliminating the least ranked permission named *INTERNET* from the *DATASET-1*, we observe that we get 75.46% accuracy with several machine learning classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked permissions, i.e., *INTERNET* and *ACCESS_NETWORK_STATE* from the *DATASET-1*. In this iteration, we obtain an accuracy of 76.47% with DT and RF classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked permissions and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.5, we achieved the highest detection accuracy of 87.34% with BC classifier upon eliminating 120 permissions out of the total lot of 129, i.e., upon considering only the top nine permissions namely {*READ_OWNER_DATA* , *SEND_DOWNLOAD_COMPLETED_INTENTS* , *WRITE_OWNER_DATA* , *UPDATE_APP_OPS_STATS* , *READ_USER_DICTIONARY* , *DEVICE_POWER* , *READ_SYNC_STATS* , *RECEIVE_WAP_PUSH* and *RECEIVE_MCS_MESSAGE* }, the highest detection accuracy was achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 87.34% when we apply the proposed Algorithm 1 to permissions.

Detection Results with intents

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked intents using *EDAS*. The algorithm provides the best intents with higher accuracy as an output. Figure 3.6 can be understood as follows. While simultaneously considering all intents without using the *EDAS* ranking, we achieve 67.19% accuracy with several machine learning classifiers. In the first iteration, after eliminating the least ranked intent named *MAIN* from the *DATASET-1*, we observe that we obtain the same 67.19% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked intents, i.e., *MAIN* and *LAUNCHER*

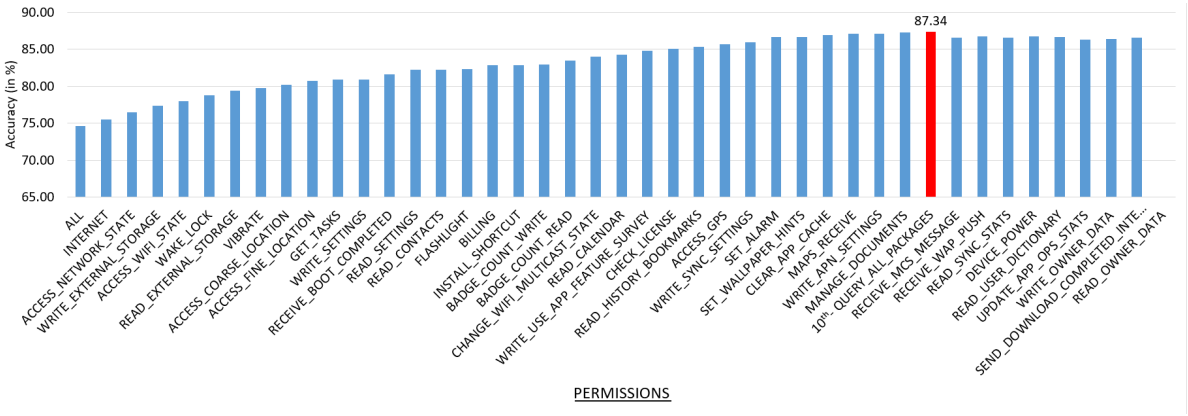


Figure 3.5: Detection results with *EDAS* using permissions

from the *DATASET-1*. In this iteration, we obtain an accuracy of 68.79% with DT, RF, and NB classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked intents and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.6, we achieved the highest detection accuracy of 90.82% with DT classifier upon eliminating 60 intents out of the total lot of 79, i.e., upon considering only the top 19 intents namely {*SEND_MULTIPLE* , *webview* , *MESSAGE_CLICKED* , *MESSAGE_ARRIVED* , *ELECTION_RESULT_V4* , *DATE_CHANGED* , *action* , *BATTERY_CHANGED* , *MEDIA_CHECKING* , *COCKROACH* , *PING_V4* , *WALLPAPER_CHANGED* , *ACTION_VIEW_DOWNLOADS* , *NEW_OUTGOING_CALL* , *SCREEN_ON* , *HEART_BEAT* , *HEADSET_PLUG* , *FEEDBACK* and *MESSAGE* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 90.82% when we apply the proposed Algorithm 1 to the intents.

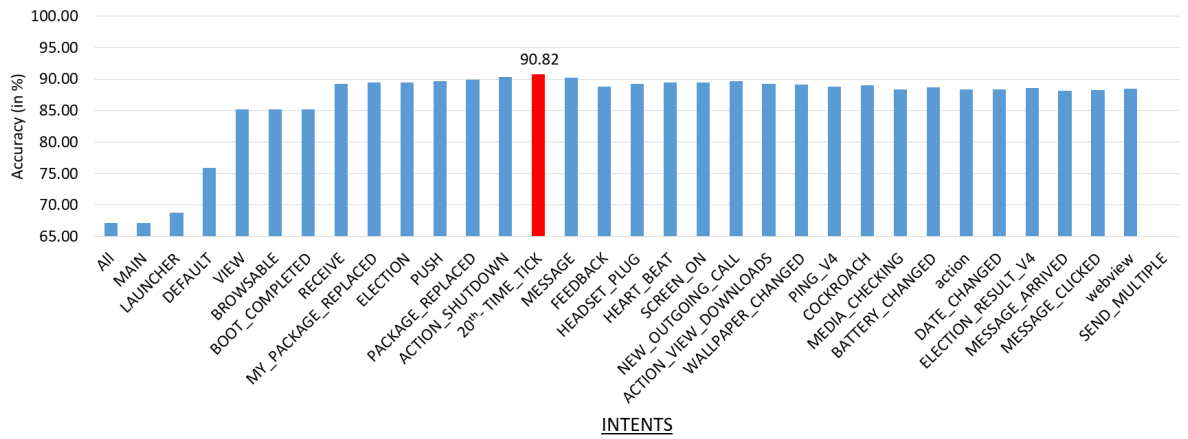


Figure 3.6: Detection results with *EDAS* using intents

Detection Results with hardware components

Next, we apply the proposed detection algorithm (Algorithm 1) with the hardware components ranked by *EDAS*. The algorithm provides the best hardware components with higher accuracy as an output. Figure 3.7 can be understood as follows. While considering all the hardware components simultaneously without using the *EDAS* ranking, we achieve 71.84% accuracy with several machine learning classifiers. In the first iteration, after eliminating the least ranked hardware component named *camera* from the *DATASET-1*, we observe that we get 73.93% accuracy with the RF and BC classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked hardware components, i.e., *camera* and *Camera.autofocus* from the *DATASET-1*. In this iteration, we obtain an accuracy of 75.55% with DT, RF, and BC classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked hardware components and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.7, we achieved the highest detection accuracy of 91.67% with RF, NB, and LR classifiers by eliminating 70 hardware components out of the total lot of 88, i.e., upon considering only the top 18 hardware components, the highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 91.66% when we apply the proposed Algorithm 1 to hardware components.

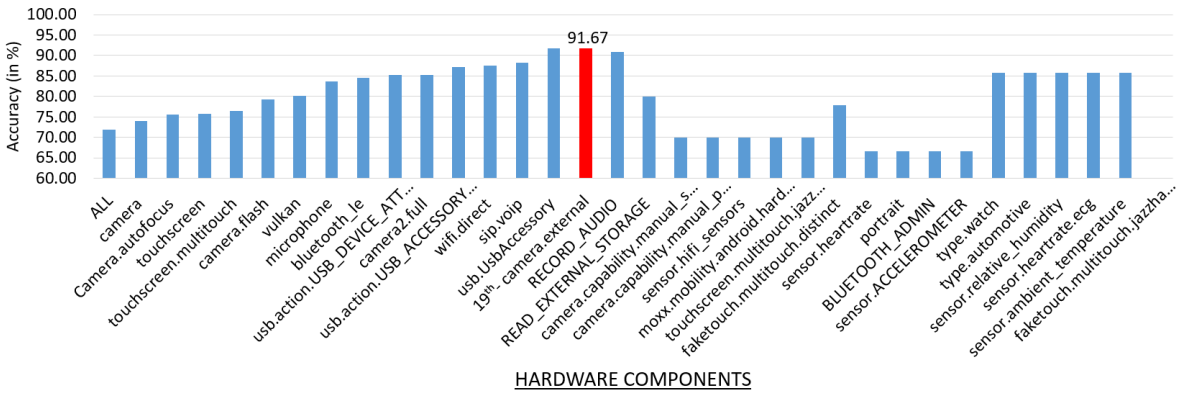


Figure 3.7: Detection results with *EDAS* using hardware components

3.4.3 Detection Results with *WASPAS*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over the *DATASET-1* while using the rankings given by *WASPAS*. Figures 3.8, 3.9 and 3.10 summarize the detection results when we consider permissions, intents, and hardware components for detection respectively. We note that in the figures mentioned above, we don't mention the names of all the ranked features as the accuracy upon eliminating them lie within similar ranges to the mentioned ones.

Detection Results using permissions

Figure 3.8 can be understood as follows. While considering all the permissions simultaneously without utilizing the *WASPAS* ranking, we achieve 74.64% accuracy with the DT classifier. At the first iteration, on eliminating the least ranked permission named *INTERNET* from the *DATASET-1*, we observe that we get 75.46% accuracy with several machine learning classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked permissions, i.e., *INTERNET* and *WRITE_EXTERNAL_STORAGE* from the *DATASET-1*. In this iteration, we get an accuracy of 76.13% with DT and RF classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked permissions and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Figure 3.8, we achieved the highest detection accuracy of 87.73% with DT classifier upon eliminating 111 permissions out of the total lot of 129, i.e., upon considering only the top 18 permissions namely { *READ_OWNER_DATA* , *READ_USER_DICTIONARY* , *SEND_DOWNLOAD_COMPLETED_INTENTS* , *UPDATE_APP_OPS_STATS* , *RECEIVE_WAP_PUSH* , *QUERY_ALL_PACKAGES* , *WRITE_OWNER_DATA* , *READ_SYNC_STATS* , *DEVICE_POWER* , *WRITE_CALL_LOG* , *MESSAGE* , *INSTALL_PACKAGES* , *WRITE_HISTORY_BOOKMARKS* , *MANAGE_DOCUMENTS* , *MAPS_RECEIVE* , *WRITE_MEDIA_STORAGE* , *RECEIVE_MCS_MESSAGE* and *ACTIVITY_RECOGNITION* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we get the highest accuracy of 87.73% when we apply the proposed Algorithm 1 on permissions.

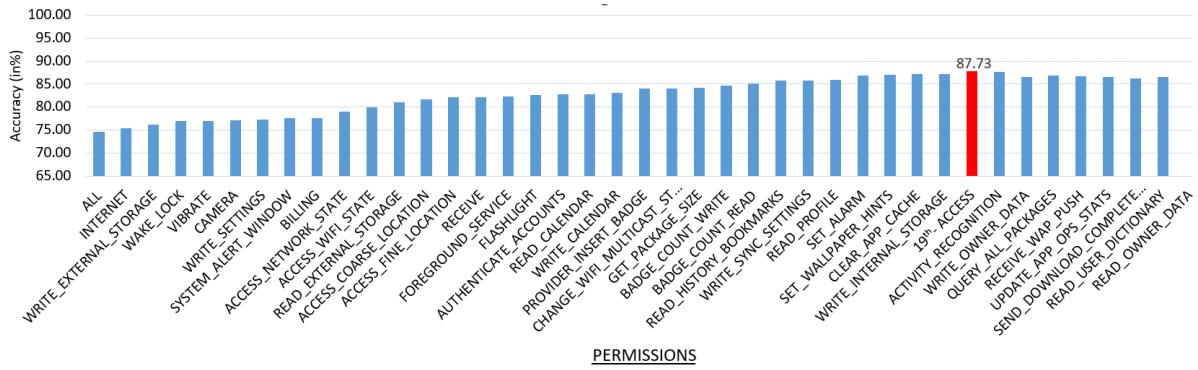


Figure 3.8: Detection results with *WASPAS* using permissions

Detection Results using intents

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked intents by *WASPAS*. The algorithm will give the best intents with higher accuracy as an output. Figure 3.9 can be understood as follows. While considering all the intents simultaneously without utilizing the *EDAS* ranking, we achieve 67.19% accuracy with several machine learning classifiers. At the first iteration, on eliminating the least ranked intent named *BATTERY_CHANGED* from the *DATASET-1*, we observe that we get the same 67.19% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked intents, i.e., *BATTERY_CHANGED* and *MEDIA_REMOVED* from the *DATASET-1*. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked intents and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Figure 3.9, we achieved the highest detection accuracy of 93.75% with DT and RF classifiers upon eliminating 53 intents out of the total lot of 79, i.e., upon considering only the top 26 intents namely {*MESSAGE_ARRIVED* , *ELECTION_RESULT_V4* , *SCREEN_ON* , *webview* , *HOME* , *PUSH_TIME* , *NOTIFICATION_OPENED* , *PushService.....PHONE_STATE* , *REGISTER* , *PACKAGE_REMOVED* , *SERVICE* , *MEDIA_EJECT* and *HEADSET_PLUG* } , highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we get the highest accuracy of 93.75% when we apply the proposed Algorithm 1 on intents.

Detection Results using hardware components

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked hardware components by *WASPAS*. The algorithm will give the best hardware components with higher accuracy

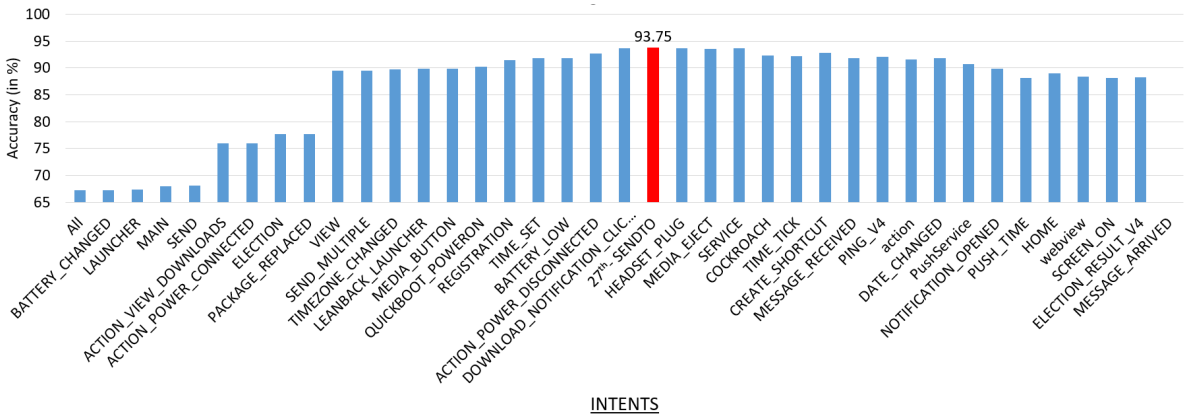


Figure 3.9: Detection results with *WASPAS* using intents

as an output. Figure 3.10 can be understood as follows. While considering all the hardware components simultaneously without utilizing the *WASPAS* ranking, we achieve 71.84% accuracy with several machine learning classifiers. At the first iteration, on eliminating the least ranked hardware component named *sensor.compass* from the *DATASET-1*, we observe that we get the same 71.84% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked hardware components, i.e., *sensor.compass* and *camera.ar* from the *DATASET-1*. In this iteration, we get an accuracy of 72.02% with DT, RF, and BC classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked hardware components and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Figure 3.10, we achieved the highest detection accuracy of 93.75% with several classifiers upon eliminating 66 hardware components out of the total lot of 88, i.e., upon considering only the top 22 hardware components namely {*faketouch.multitouch.jazzhand* , *sensor.ambient_temperature* , *sensor.heartrate.ecg* , *sensor.relative_humidity* , *type.automotive* , *BLUETOOTH_ADMIN* , *portrait* , *sensor.heartrate* , *faketouch.multitouch.distinct.....moxx.mobility.android.hardwareplatform. firebaseinitprovider* , *READ_EXTERNAL_STORAGE* , *RECORD_AUDIO* , *vr.headtracking* , *opengles.aep* , *camera.external* and *biometrics* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we get the highest accuracy of 93.75% when we apply the proposed Algorithm 1 on hardware components.

The compiled detection results when we apply the proposed algorithm to DATASET 1 are summarized in Table 3.16. From the table, we observe that we obtain the highest accuracy

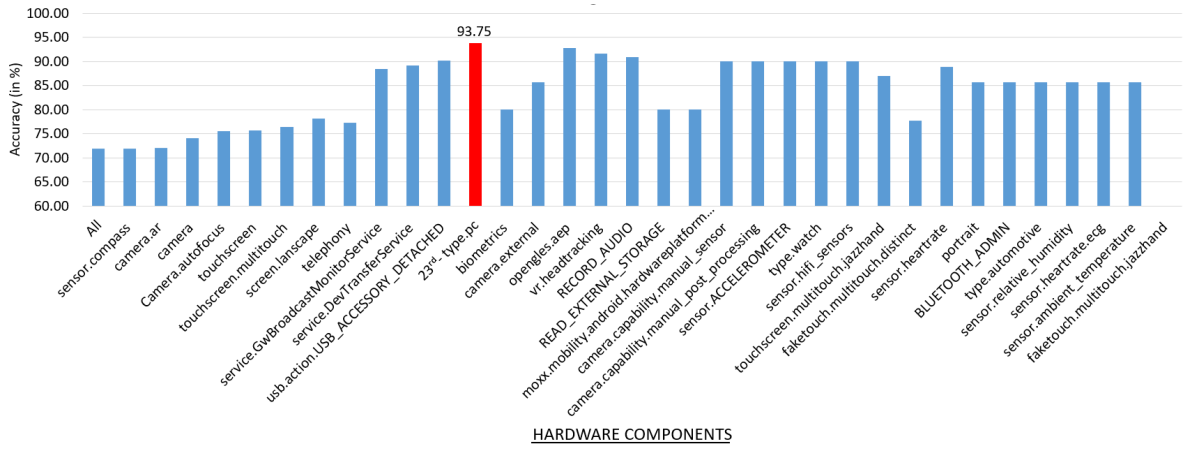


Figure 3.10: Detection results with *WASPAS* using hardware components

of 98.01% on using 15 permissions when we apply the proposed Algorithm 1 to the ranking described by *TOPSIS*. Similarly, the highest accuracy of 99.10% can be achieved using 46 intents when we apply the proposed Algorithm 1 to the ranking described by *TOPSIS*, whereas the ranking given by *WASPAS* results in the highest detection accuracy of 93.75% on using 22 hardware components. **Hence, in response to research question four, we conclude that the *TOPSIS*' top-ranked 46 intents, i.e. intents give the best detection accuracy results amongst the top three most commonly used *AndroidManifest* file features.**

Table 3.16: Compiled Detection results (in %) on applying the proposed algorithm on *DATASET -1*

Feature Ranking Method used	**PERMISSIONS**		**INTENTS**		**HARDWARE COMPONENTS**	
	Number used	**Accuracy (in %)**	**Number used**	**Accuracy (in %)**	**Number used**	**Accuracy (in %)**
TOPSIS	**15**	**98.01**	**46**	**99.10**	19	91.67
EDAS	09	87.34	19	90.82	18	91.67
WASPAS	18	87.73	26	93.75	**22**	**93.75**
No Ranking (All features used)	129	74.64	79	67.19	88	71.84

At the same time, when no feature ranking of any type is used and all the features are fed to the classifiers at once, i.e., on considering the large initial vector of all the permissions, intents, or hardware components simultaneously, we observe that the highest detection accuracy obtained is merely 74.64%, 67.19%, and 71.84% respectively. **Based on the results and the low detection accuracy depicted by Table 3.16, we answer our first research question that feature ranking helps us eliminate irrelevant features that can hamper detection accuracy.**

3.4.4 Comparison with other feature ranking techniques

We applied various *MCDM* techniques to rank the features. However, feature ranking techniques such as Principal Component Analysis (PCA) [137] and Entropy-based Category Coverage Difference (ECCD) [52] have been used in other studies for Android malware detection. Next, we compare the performance of the ranking obtained using various *MCDM* techniques with the Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD). Tables 3.17 and 3.18 highlight the top 10 permissions, intents, and hardware components ranked using Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD), respectively.

Table 3.17: Top 10 features ranked using PCA

PERMISSIONS	**INTENTS**	**HARDWARE COMPONENTS**
READ_SYNC_STATS	NOTIFICATION_ RECEIVED_ PROXY	sensor.heartrate.ecg
WRITE_OWNER_DATA	PUSH_TIME	type.automotive
CHANGE_WIFI_STATE	REPORT	sensor.ambient _temperature
READ_OWNER_DATA	PushService	sensor.relative _humidity
WRITE_CALL_LOG	REGISTER	faketouch. multitouch. jazzhand
READ_USER _DICTIONARY	NOTIFICATION _OPENED	sensor.hifi _sensors
WRITE_SETTINGS	MESSAGE _RECEIVED	camera. capability. manual_post _processing
READ_SYNC_SETTINGS	NOTIFICATION _RECEIVED	camera. capability. manual_sensor
GET_TASKS	CONNECTION	camera.external
RECEIVE_WAP_PUSH	DaemonService	opengles.aep

Table 3.18: Top 10 features ranked using ECCD

PERMISSIONS	**INTENTS**	**HARDWARE COMPONENTS**
MOUNT_UNMOUNT _FILESYSTEMS	USER _PRESENT	autofocus
READ_PHONE_STATE	PACKAGE _REMOVED	service. GwBroadcast MonitorService
CHANGE_WIFI_STATE	NOTIFICATION _RECEIVED _PROXY	camera
GET_TASKS	PushService	screen.portrait
SYSTEM_ALERT _WINDOW	PUSH_TIME	fingerprint
READ_LOGS	REPORT	location
CHANGE_NETWORK _STATE	REGISTER	service. DevTransferService
WRITE_SETTINGS	MESSAGE _RECEIVED	touchscreen
ACCESS_WIFI_STATE	NOTIFICATION _OPENED	audio.pro
JPUSH_MESSAGE	NOTIFICATION _RECEIVED	location.network

For comparison, we ranked all three feature types, i.e., permissions, intents, and hardware components, using Principal Component Analysis (PCA) and Entropy-Based Category Coverage Difference (ECCD) and further applied the proposed Algorithm 1 to *DATASET-1* to obtain their corresponding detection accuracies. First, we apply the proposed detection algorithm to permissions after ranking them using Principal Component Analysis (PCA), and Entropy-based Category Coverage Difference (ECCD). The proposed algorithm, i.e., Algorithm 1,

will provide the best set of permissions with higher accuracy as an output. As we can see from Table 3.19, we obtain the highest accuracy of 87.56% with five permissions, namely {*READ_SYNC_STATS* , *WRITE_OWNER_DATA* , *CHANGE_WIFI_STATE* , *READ_OWNER_DATA*, and *WRITE_CALL_LOG* }, when we rank the permissions with PCA. Similarly, we obtained the highest accuracy of 89.68% with only one permission namely *MOUNT_UNMOUNT_FILESYSTEMS* ranked using ECCD. Simultaneously, with our proposed approach on the permission ranking given by *TOPSIS*, we obtained the highest accuracy of 98.01% with 15 permissions.

Table 3.19: Comparison of best detection results (in %) from *MCDM* techniques with Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on permissions

Approach used	**Number of PERMISSIONS used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**									
		DT	**RF**	**BC**	**NB**	**LR**	**SVC**	**ANN**	**MLP**	**DNN**	**CNN**
TOPSIS (Our approach)	15	**98.01**	88.73	88.73	68.28	69.66	73.12	63.70	41.54	60.86	74.25
PCA [137]	05	**87.56**	87.54	87.53	78.76	78.76	78.76	74.61	75.43	78.50	78.54
ECCD [52]	01	**89.68**	86.66	89.61	80.12	80.12	80.2	79.87	72.08	79.95	76.65

Next, we apply the proposed detection algorithm to intents, after ranking them using Principal Component Analysis (PCA), and Entropy-based Category Coverage Difference (ECCD). The proposed algorithm, i.e., Algorithm 1, will provide the best set of intents with higher accuracy as an output. The results are summarized in Table 3.20, as it can be observed that we obtain the highest accuracy of 95.43% with nine intents, namely {*NOTIFICATION_RECEIVED_PROXY* , *PUSH_TIME* , *REPORT* , *PushService* , *REGISTER* , *NOTIFICATION_OPENED* , *MESSAGE_RECEIVED* , *NOTIFICATION_RECEIVED*, and *CONNECTION* }, when we rank the intents using PCA. Similarly, we obtain the highest accuracy of 96% with 19 intents, namely {*USER_PRESENT PACKAGE_REMOVED* , *NOTIFICATION_RECEIVED_PROXY* , *PushService* , *PUSH_TIME*....., *UNREGISTRATION* , *SERVICE* , *START_FROM_AGOO* , *ELECTION* and *PING_V4* } , when we rank the intents with ECCD. At the same time, with our proposed approach on the intents ranking given by *TOPSIS*, we obtain the highest accuracy of 99.10% with 46 intents. Hence, our model using the *MCDM* techniques outperforms the Principal Component Analysis (PCA), and Entropy-based Category Coverage Difference (ECCD) on intents.

Now, we apply the proposed detection algorithm to hardware components, after ranking them using Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD). The proposed algorithm, i.e., Algorithm 1, will provide the best set of hardware components with higher accuracy as an output. The results are summarized in Table 3.21, as it can be observed that we obtain the highest accuracy of 88.88% with 11 hard-

Table 3.20: Comparison of best detection results (in %) from *MCDM* techniques with Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on intents

Approach used	**Number of INTENTS used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**									
		DT	**RF**	**BC**	**NB**	**LR**	**SVC**	**ANN**	**MLP**	**DNN**	**CNN**
TOPSIS (our approach)	46	**99.10**	95.43	94.71	95.45	88.20	88.59	88.26	74.19	88.68	88.45
PCA [137]	09	**95.43**	95.41	95.41	88.93	88.54	88.50	88.61	88.40	88.40	89.33
ECCD [52]	19	**96.00**	**96.00**	95.98	90.42	90.42	88.65	90.8	89.5	90.76	85.54

ware components, namely {*sensor.heartrate.ecg* , *type.automotive* , *sensor.ambient_temperature* , *sensor.relative_humidity* , *faketouch.multitouch.jazzhand* , *sensor.hifi_sensors* , *camera. capability.manual _post_processing* , *camera.capability.manual_sensor* , *camera.external* , *opengles.aep* and *camera.capability.raw* }, when we rank the hardware components with PCA. Similarly, we obtain the highest accuracy of 90.50% with 15 hardware components, namely {*autofocus* , *service.GwBroadcastMonitorService* , *camera* , *screen.portrait*,.....*vulkan* , *telephony* , *vibrate* , *touchscreen.multitouch.distinct* }, when we rank the hardware components with ECCD. At the same time, with our proposed approach on the hardware components ranking given by *WASPAS*, we obtain the highest accuracy of 93.78% with 22 hardware components. Hence, our model using the *MCDM* techniques outperforms the Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on hardware components too.

Table 3.21: Comparison of best detection results (in %) from *MCDM* techniques with Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) on hardware components

Approach used	**Number of HARDWARE COMPONENTS used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**									
		DT	**RF**	**BC**	**NB**	**LR**	**SVC**	**ANN**	**MLP**	**DNN**	**CNN**
WASPAS (Our approach)	22	**93.75**	**93.75**	90.00	**93.75**	**93.75**	78.57	78.05	72.54	78.85	78.78
PCA [137]	11	66.66	66.66	**88.88**	44.44	44.44	58	55.49	56.5	58.5	77.77
ECCD [52]	15	**90.50**	**90.50**	**90.50**	86.89	90.42	82.64	85.82	80.53	84.44	84.44

3.4.5 Comparison with other statistical tests

We applied various *MCDM* techniques to rank the features. However, statistical tests such as Mutual Information, Pearson Correlation Coefficient, and T-Test have been used in other studies such as [131] for Android malware detection. Hence, next, we compare the performance of the ranking obtained using various *MCDM* techniques with the mutual information, Pearson correlation coefficient, and T-test. Tables 3.22 , 3.23 and 3.24 highlight the top ten permis-

sions, intents, and hardware components ranked with mutual information, Pearson Correlation Coefficient, and T-Test, respectively.

Table 3.22: Top 10 features ranked using Mutual Information

PERMISSIONS	INTENTS	HARDWARE COMPONENTS
MOUNT_ UNMOUNT_ FILESYSTEMS	USER_PRESENT	touchscreen
READ_PHONE_ STATE	PACKAGE_ REMOVED	camera
CHANGE_WIFI_ STATE	DEFAULT	Camera.autofocus
GET_TASKS	REGISTER	touchscreen. multitouch. distinct
SYSTEM_ALERT_ WINDOW	NOTIFICATION_ RECEIVED_ PROXY	touchscreen. multitouch
READ_LOGS	PACKAGE_ ADDED	location
WRITE_SETTINGS	PushService	location.network
CHANGE_ NETWORK_ STATE	PUSH_TIME	location.GPS
ACCESS_WIFI_ STATE	REPORT	screen.portrait
ACCESS_COARSE_ LOCATION	MESSAGE_ RECEIVED	telephony

Table 3.23: Top 10 features ranked using Pearson Correlation Coefficient

PERMISSIONS	INTENTS	HARDWARE COMPONENTS
ACCESS	action	action.NEW_ PICTURE
BADGE_COUNT_ READ	ACTION_ RICHPUSH_ CALLBACK	type.watch
ACCESS_ BACKGROUND_ LOCATION	LAUNCHER	audio.low_latency
ACCESS_COARSE_ LOCATION	ACTION_ SHUTDOWN	biometrics
REQUEST_IGNORE_ BATTERY_ OPTIMIZATIONS	DAYDREAM	moxx.mobility. android.hardware platform. firebaseinitprovider
ACCESS_COARSE_ UPDATES	BATTERY_ CHANGED	camera.ar
BROADCAST_ PACKAGE_ ADDED	CREATE_ SHORTCUT	autofocus
ACCESS_GPS	COCKROACH	BLUETOOTH_ ADMIN
ACCESS_NETWORK_ STATE	CLICK	portrait
GOOGLE_PHOTOS	SEND	vibrate

Table 3.24: Top 10 features ranked using T-Test

PERMISSIONS	INTENTS	HARDWARE COMPONENTS
SEND_DOWNLOAD_ COMPLETED_INTENTS	webview	action.NEW_ PICTURE
UPDATE_APP_ OPS_STATS	UNREGISTER	type.watch
READ_OWNER_ DATA	ELECTION_ RESULT_V4	BLUETOOTH_ ADMIN
READ_USER_ DICTIONARY	action	sensor. ACCELEROMETER
BAIDU_LOCATION_ SERVICE	WALLPAPER_ CHANGED	faketouch.multitouch. jazzhand
RECEIVE_WAP_ PUSH	DATE_CHANGED	portrait
MESSAGE	MEDIA_ CHECKING	sensor.ambient_ temperature
INSTALL_ PACKAGES	COCKROACH	sensor.heartrate.ecg
WRITE_MEDIA_ STORAGE	PING_V4	sensor.relative_ humidity
WRITE_CALL_LOG	NEW_OUTGOING_ CALL	type.automotive

For comparison, we ranked all three feature types, i.e., permissions, intents, and hardware components, using mutual information, Pearson Correlation Coefficient, and T-Test and further applied the proposed Algorithm 1 on *DATASET-1* to obtain their corresponding detection accuracies. First, we apply the proposed detection algorithm to permissions after ranking them using mutual information, Pearson's correlation coefficient, and T-test. The proposed algorithm, i.e., Algorithm 1, will provide the best set of permissions with higher accuracy as

an output. As we can see from Table 3.25, we obtain the highest accuracy of 89.68% with only one permission, namely *MOUNT_UNMOUNT_FILESYSTEMS*, when we rank the permissions with Mutual Information. With Pearson's correlation coefficient, we obtain the highest accuracy of 85.48% again with only one permission, namely *ACCESS*. Similarly, we obtained the highest accuracy of 88.93% with 65 permissions ranked using the T-test. Simultaneously, with our proposed approach on the permission ranking given by *TOPSIS*, we obtain the highest accuracy of 98.01% with 15 permissions.

Table 3.25: Comparison of best detection results (in %) from *MCDM* techniques with Mutual Information, Pearson Coefficient, and T-Test on permissions

Approach used	**Number of PERMISSIONS used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**									
		DT	**RF**	**BC**	**NB**	**LR**	**SVC**	**ANN**	**MLP**	**DNN**	**CNN**
TOPSIS (Our approach)	15	**98.01**	88.73	88.73	68.28	69.66	73.12	63.70	41.54	60.86	74.25
Mutual Information [131]	01	**89.68**	89.67	89.62	80.12	80.12	80.20	79.87	72.08	79.95	79.65
Correlation Coefficient [131]	01	**85.48**	85.42	85.43	72.66	72.66	72.66	73.26	65.49	73.1	72.76
T-Test [131]	65	**88.93**	88.92	88.92	83.45	83.45	83.60	83.15	82.86	83.26	83.08

Next, we apply the proposed detection algorithm to intents, after ranking them using mutual information, Pearson's correlation coefficient, and T-Test. The proposed algorithm, i.e., Algorithm 1, will provide the best set of intents with higher accuracy as an output. The results are summarized in Table 3.26, as it can be observed that we obtain the highest accuracy of 92.18% with only two intents, namely *USER_PRESENT* and *PACKAGE_REMOVED* when we rank the intents with Mutual Information. However, with Pearson's correlation coefficient, we obtain the highest accuracy of 89.27% with two intents, namely *action* and *ACTION_RICHPUSH_CALLBACK*. Similarly, we obtain the highest accuracy of 94.57% with 51 intents, namely {*webview* , *UNREGISTRATION* , *ELECTION_RESULT_V4* , *action* , *WALLPAPER_CHANGED*,...*NOTIFICATION_OPENED* , *PUSH_TIME* , *REPORT* , *PushService*, and *NOTIFICATION_RECEIVED_PROXY* }, when we rank the intents with T-test. At the same time, with our proposed approach on the intents ranking given by *TOPSIS*, we obtain the highest accuracy of 99.10% with 46 intents. Hence, our model using the *MCDM* techniques outperforms the Mutual Information, Pearson Correlation Coefficient, and T-Test on intents.

Now, we apply the proposed detection algorithm to hardware components, after ranking them using mutual information, Pearson's correlation coefficient, and T-Test. The proposed algorithm, i.e., Algorithm 1, will provide the best set of hardware components with higher

Table 3.26: Comparison of best detection results (in %) from *MCDM* techniques with Mutual Information, Pearson Coefficient, and T-Test on intents

Approach used	**Number of INTENTS used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**									
		DT	**RF**	**BC**	**NB**	**LR**	**SVC**	**ANN**	**MLP**	**DNN**	**CNN**
TOPSIS (Our approach)	46	**99.10**	95.43	94.71	95.45	88.20	88.59	88.26	74.19	88.68	88.45
Mutual Information [131]	02	92.17	**92.18**	92.12	80.24	79.58	80.90	79.36	80.46	80.51	80.75
Correlation Coefficient [131]	02	**89.27**	89.24	89.11	63.67	63.67	63.66	63.02	60.81	60.87	63.38
T-Test [131]	51	94.56	**94.57**	94.54	87.44	87.44	85.69	85.99	80.80	60.87	87.30

accuracy as an output. The results are summarized in Table 3.27, as it can be observed that we obtain the highest accuracy of 76.49% with only one hardware component, namely *touchscreen* when we rank the hardware components with mutual information. With both Pearson's correlation coefficient and the T-test, we obtained the highest accuracy of 91.67% with seven and two hardware components respectively. At the same time, with our proposed approach on the hardware components ranking given by *WASPAS*, we obtain the highest accuracy of 93.75% with 22 hardware components. Hence, our model using the *MCDM* techniques outperforms the Mutual Information, Pearson Correlation Coefficient, and T-Test on hardware components.

Table 3.27: Comparison of best detection results (in %) from *MCDM* techniques with Mutual Information, Pearson Coefficient, and T-Test on hardware components

Approach used	**Number of HARDWARE COMPONENTS used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**									
		DT	**RF**	**BC**	**NB**	**LR**	**SVC**	**ANN**	**MLP**	**DNN**	**CNN**
WASPAS (Our approach)	22	**93.75**	**93.75**	90.00	**93.75**	**93.75**	78.57	78.05	72.54	78.85	78.78
Mutual Information [131]	01	74.05	74.05	74.05	73.56	74.05	74.04	74.64	74.48	74.49	**76.49**
Correlation Coefficient [131]	07	**91.67**	**91.67**	**91.67**	48.92	86.02	**91.67**	54.84	54.91	63.04	53.69
T-Test [131]	02	**91.66**	**91.66**	**91.66**	83.33	83.33	83.33	57.58	67.99	57.00	66.66

3.5 Detection Results on *DATASET-2*

The applications in *DATASET-1* are dated from 2016 to 2022. In the following subsections, we discuss the results obtained by testing our proposed Algorithm 1 over a new and more recent dataset, i.e., 2000 malicious applications downloaded from *Androzoo* that were detected between 2021 and 2022. To check the efficiency of the three most commonly used features in the *AndroidManifest* file, we again perform three experiments, considering 1) permissions,

2) intents, and 3) hardware components, by applying the three *MCDM* techniques individually, but this time on *DATASET-2*. We discuss these results in upcoming subsections.

3.5.1 Detection Results with *TOPSIS*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over *DATASET-2* while using the rankings given by *TOPSIS*. Figures 3.11 , 3.12 and 3.13 summarize the detection results when we consider permissions, intents, and hardware components for detection. We note that in the figures mentioned above, we do not mention the names of all the ranked features because the accuracy upon eliminating them lies within similar ranges to the mentioned ones.

Detection Results with *TOPSIS* using permissions

Figure 3.11 can be understood as follows. While simultaneously considering all permissions without using the *TOPSIS* ranking, we achieve 70.79% accuracy with the DT and RF classifiers. In the first iteration, on eliminating the least ranked permission named *INTERNET* from the *DATASET-2*, we observe that we get 71.75% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked permissions, i.e., *INTERNET* and *ACCESS_NETWORK_STATE* from *DATASET-2*. In this iteration, we obtain an accuracy of 72.33% with the RF classifier. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked permissions and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.11, we achieved the highest detection accuracy of 87.89% with BC classifier upon eliminating 123 permissions out of the total lot of 129, i.e., by considering only the top six permissions namely {*UPDATE_APP_OPS_STATS* , *USE_BIOMETRIC* , *MAPS_RECEIVE* , *READ_OWNER_DATA* , *READ_USER_DICTIONARY* , *SEND_DOWNLOAD_COMPLETED_INTENTS* and *QUERY_ALL_PACKAGES* }, the highest detection accuracy was achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 87.89% when we apply the proposed Algorithm 1 to permissions.

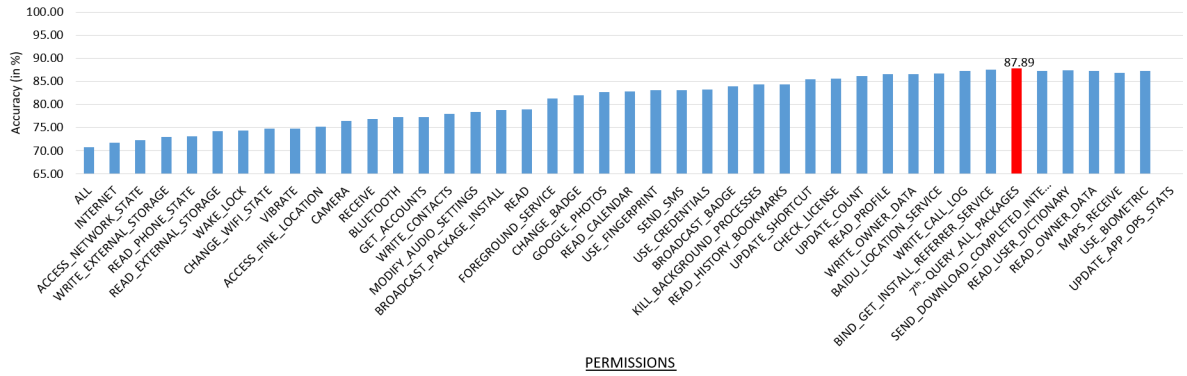


Figure 3.11: Detection results with *TOPSIS* using permissions

Detection Results with *TOPSIS* using intents

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked intents using *TOPSIS*. The algorithm provides the best intents with higher accuracy as an output. Figure 3.12 can be understood as follows. While considering all intents simultaneously without using the *TOPSIS* ranking, we achieve 65.35% accuracy with RF. In the first iteration, after eliminating the least ranked intent named *MAIN* from the *DATASET-2*, we observe that we get 65.97% accuracy with several machine learning classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked intents, i.e., *MAIN* and *LAUNCHER* from the *DATASET-2*. In this iteration, we obtain an accuracy of 67.05% with DT, RF, and NB classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked intents and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.12, we achieved the highest detection accuracy of 95.85% with RF classifier upon eliminating 33 intents out of the total lot of 79, i.e., upon considering only the top 46 intents, the highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 95.85% when we apply the proposed Algorithm 1 to the intents.

Detection Results with *TOPSIS* using hardware components

Next, we apply the proposed detection algorithm (Algorithm 1) with the hardware components ranked by *TOPSIS*. The algorithm provides the best hardware components with higher accuracy as an output. Figure 3.13 can be understood as follows. While considering all the hardware components simultaneously without using the *TOPSIS* ranking, we achieve 65.39%

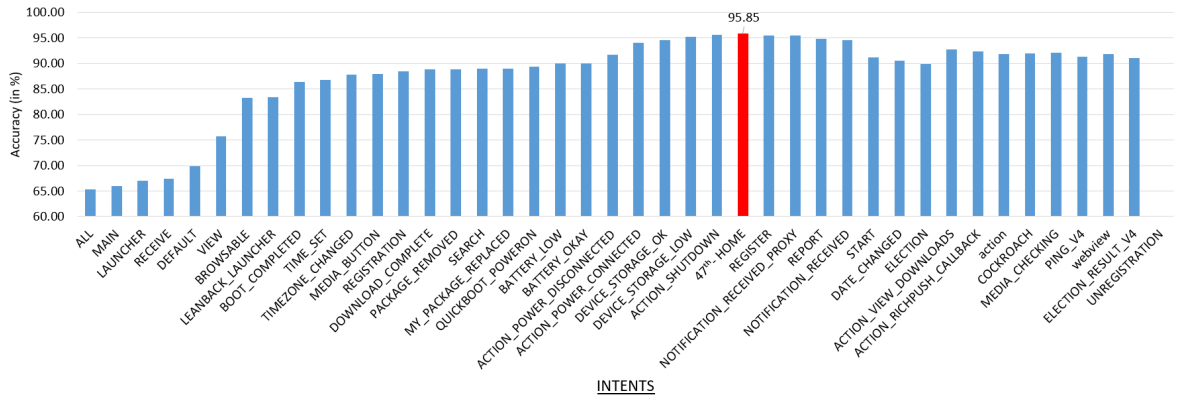


Figure 3.12: Detection results with *TOPSIS* using intents

accuracy with the BC machine learning classifier. In the first iteration, after eliminating the least ranked hardware component named *camera* from the *DATASET-2*, we observe that we get 66.45% accuracy with the BC classifier. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked hardware components, i.e., *camera* and *Camera.autofocus* from the *DATASET-2*. In this iteration, we obtain an accuracy of 67.90% with the RF classifier. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked hardware components and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.13, we achieved the highest detection accuracy of 94.44% with DT, BC and RF classifiers upon eliminating 72 hardware components out of the total lot of 88, i.e., upon considering only the top 16 hardware components namely {*faketouch.multitouch.jazzhand* , *sensor.ambient_temperature* , *sensor.heartrate.ecg* , *sensor.relative_humidity* , *type.automotive* , *portrait* , *BLUETOOTH_ADMIN* , *sensor.heartrate* , *sensor.ACCELEROMETER* , *type.watch*..... *faketouch.multitouch.distinct* , *touchscreen.multitouch.jazzhand* , *camera.capability.manual_post_processing* , *camera.capability.manual_sensor* and *READ_EXTERNAL_STORAGE* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 94.44% when we apply the proposed Algorithm 1 to hardware components.

3.5.2 Detection Results with *EDAS*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over *DATASET-2* while using the rankings given by *EDAS*. Figures 3.14 , 3.15 and

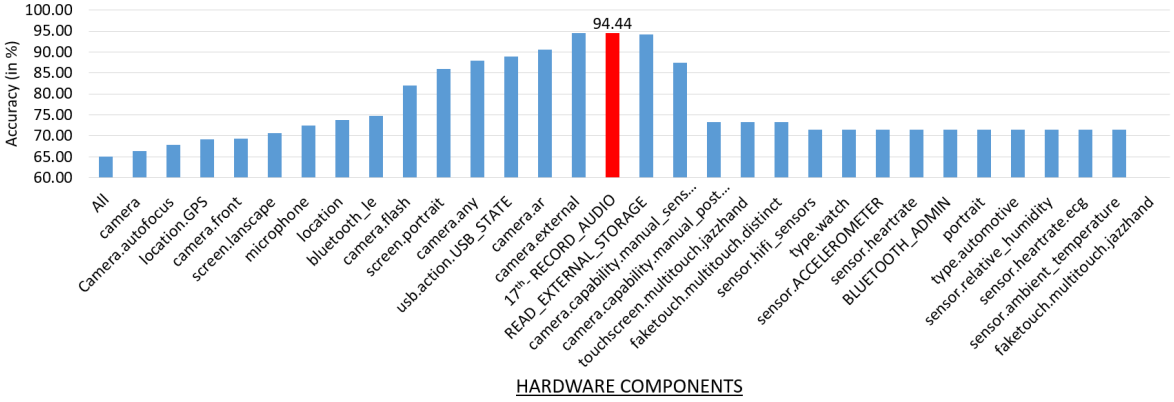


Figure 3.13: Detection results with *TOPSIS* using hardware components

3.16 summarize the detection results when we consider permissions, intents, and hardware components for detection. We note that in the figures mentioned above, we do not mention the names of all the ranked features because the accuracy upon eliminating them lies within similar ranges to the mentioned ones.

Detection Results with *EDAS* using permissions

Figure 3.14 can be understood as follows. While simultaneously considering all permissions without using the *EDAS* ranking, we achieve 70.79% accuracy with the DT and RF classifiers. In the first iteration, on eliminating the least ranked permission named *INTERNET* from the *DATASET-2*, we observe that we get 71.75% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked permissions, i.e., *INTERNET* and *ACCESS_NETWORK_STATE* from the *DATASET-2*. In this iteration, we obtain an accuracy of 72.33% with the RF classifier. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked permissions and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.14, we achieved the highest detection accuracy of 88.67% with DT classifier upon eliminating 120 permissions out of the total lot of 129, i.e., upon considering only the top nine permissions namely {*READ_OWNER_DATA* , *SEND_DOWNLOAD_COMPLETED_INTENTS* , *WRITE_OWNER_DATA* , *UPDATE_APP_OPS_STATS* , *READ_USER_DICTIONARY* , *DEVICE_POWER* , *READ_SYNC_STATS* , *RECEIVE_WAP_PUSH* and *RECEIVE_MCS_MESSAGE* }, the highest detection accuracy was achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we

obtain the highest accuracy of 88.67% when we apply the proposed Algorithm 1 to permissions.

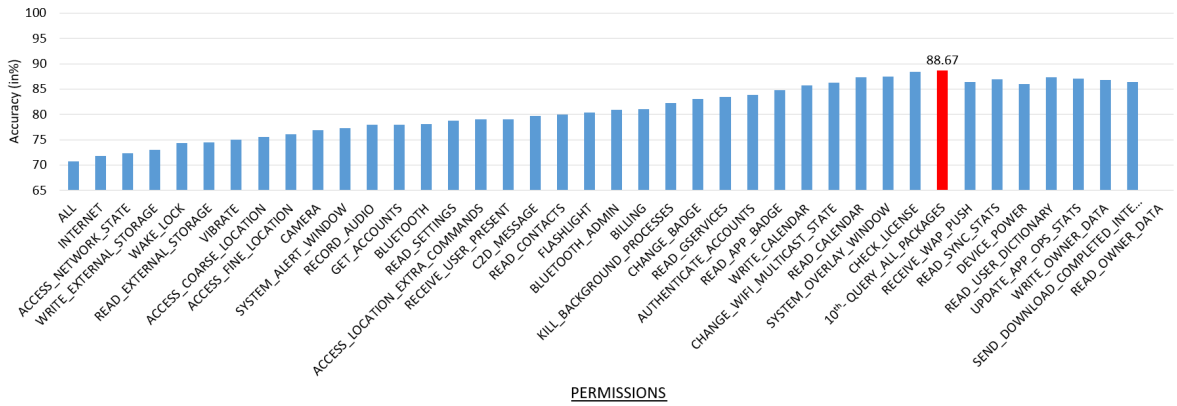


Figure 3.14: Detection results with *EDAS* using permissions

Detection Results with *EDAS* using intents

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked intents using *EDAS*. The algorithm provides the best intents with higher accuracy as an output. Figure 3.15 can be understood as follows. While considering all intents simultaneously without using the *EDAS* ranking, we achieve 65.35% accuracy with RF. In the first iteration, after eliminating the least ranked intent named *MAIN* from the *DATASET-2*, we observe that we get 65.97% accuracy with several machine learning classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked intents, i.e., *MAIN* and *LAUNCHER* from the *DATASET-2*. In this iteration, we obtain an accuracy of 67.05% with DT, RF, and NB classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked intents and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.15, we achieved the highest detection accuracy of 93.72% with DT classifier upon eliminating 60 intents out of the total lot of 79, i.e., upon considering only the top 19 intents namely {*SEND_MULTIPLE* , *webview* , *MESSAGE_CLICKED* , *MESSAGE_ARRIVED* , *ELECTION_RESULT_V4* , *DATE_CHANGED* , *action* , *BATTERY_CHANGED* , *MEDIA_CHECKING* , *COCKROACH* , *PING_V4* , *WALLPAPER_CHANGED* , *ACTION_VIEW_DOWNLOADS* , *NEW_OUTGOING_CALL* , *SCREEN_ON* , *HEART_BEAT* , *HEADSET_PLUG* , *FEEDBACK* and *MESSAGE* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest accuracy of 93.72% when we apply the proposed Algorithm

1 on intents.

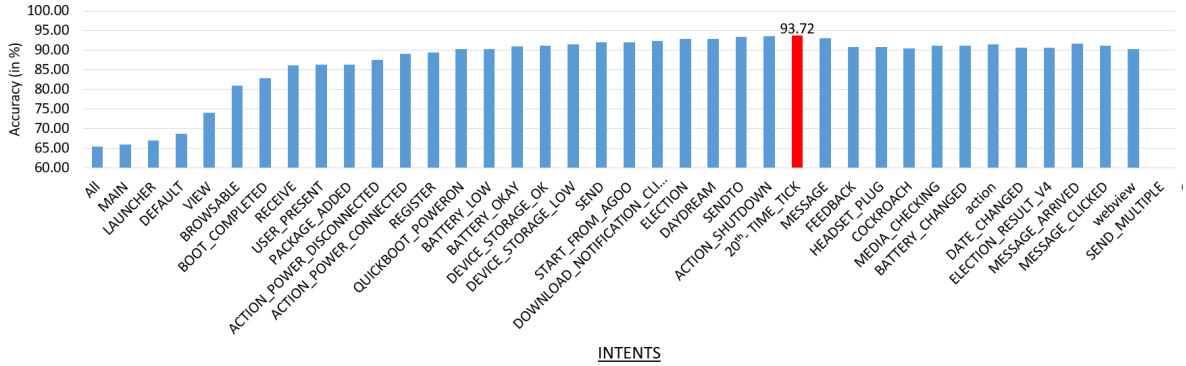


Figure 3.15: Detection results with *EDAS* using intents

Detection Results with *EDAS* using hardware components

Next, we apply the proposed detection algorithm (Algorithm 1) with the hardware components ranked by *EDAS*. The algorithm provides the best hardware components with higher accuracy as an output. Figure 3.16 can be understood as follows. While considering all the hardware components simultaneously without using the *EDAS* ranking, we achieve 65.39% accuracy with BC machine learning classifiers. In the first iteration, after eliminating the least ranked hardware component named *camera* from the *DATASET-2*, we observe that we get 66.45% accuracy with the BC classifier. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked hardware components, i.e., *camera* and *Camera.autofocus* from the *DATASET-2*. In this iteration, we obtain an accuracy of 67.90% with the RF classifier. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked hardware components and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As shown in Figure 3.16, we achieved the highest detection accuracy of 94.44% with DT, RF and BC classifiers upon eliminating 71 hardware components out of the total lot of 88, i.e., upon considering only the top 17 hardware components namely {*faketouch.multitouch.jazzhand* , *sensor.ambient_temperature* , *sensor.heartrate.ecg* , *sensor.relative_humidity* , *type.automotive* , *type.watch* , *sensor.ACCELEROMETER* , *BLUETOOTH_ADMIN* , *portrait* , *sensor.heartrate* , *faketouch.multitouch.distinct* , *touchscreen.multitouch.jazzhand* , *moxx.mobility.android.hardwareplatform. firebaseinitprovider* , *sensor.hifi_sensors* , *camera.capability.manual_post_processing* , *camera.capability.manual_sensor* and *READ_EXTERNAL_STORAGE* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we obtain the highest

accuracy of 94.44% when we apply the proposed Algorithm 1 to hardware components.

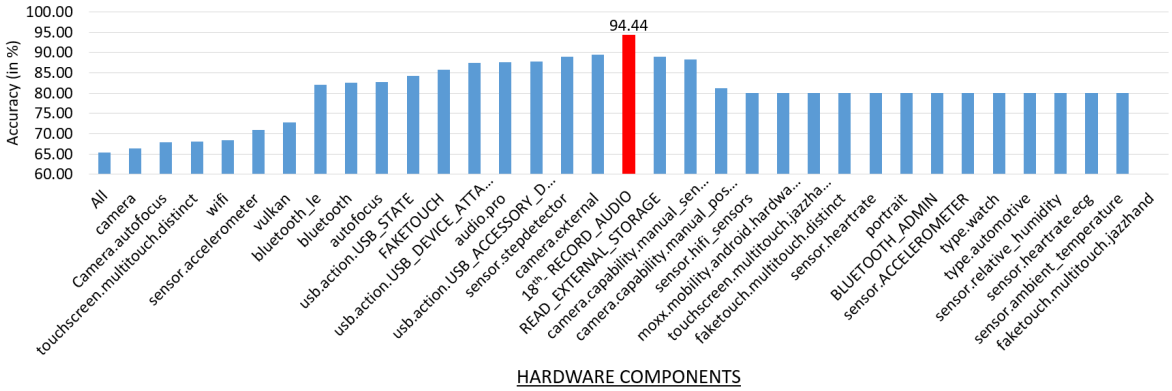


Figure 3.16: Detection results with *EDAS* using hardware components

3.5.3 Detection Results with *WASPAS*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over the *DATASET-2* while using the rankings given by *WASPAS*. Figure 3.17 , 3.18 and 3.19 summarize the detection results when we consider permissions, intents, and hardware components for detection respectively. We note that in the figures mentioned above, we don't mention the names of all the ranked features as the accuracy upon eliminating them lie within similar ranges to the mentioned ones.

Detection Results with *WASPAS* using permissions

Figure 3.17 can be understood as follows. While considering all the permissions simultaneously without utilising the *WASPAS* ranking, we achieve 70.79% accuracy with DT and RF classifiers. At the first iteration, on eliminating the least ranked permission named *INTERNET* from the *DATASET-2*, we observe that we get 71.75% accuracy. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked permissions, i.e., *INTERNET* and *WRITE_EXTERNAL_STORAGE* from the *DATASET-2*. In this iteration, we get an accuracy of 72.61% with DT and RF classifiers. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked permissions and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Figure 3.17, we achieved the highest detection accuracy of 89.28% with DT classifier upon eliminating 99 permissions out of the total lot of 129, i.e., upon considering only the top 30 permissions namely {*READ_OWNER_DATA* , *READ_USER_DICTIONARY* ,

SEND_DOWNLOAD_COMPLETED_INTENTS , *UPDATE_APP_OPS_STATS* , *RECEIVE_WAP_PUSH* , *QUERY_ALL_PACKAGES* , *WRITE_OWNER_DATA* , *READ_SYNC_STATS*,.....*ACCESS_BACKGROUND_LOCATION* , *RECEIVE_MMS* , *RUN_INSTRUMENTATION* , *RECORD_VIDEO* , *USE_BIOMETRIC* , *WRITE_APN_SETTINGS* , *SET_WALLPAPER_HINTS* and *SET_ALARM* } , highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we get the highest accuracy of 89.28% when we apply the proposed Algorithm 1 on permissions.

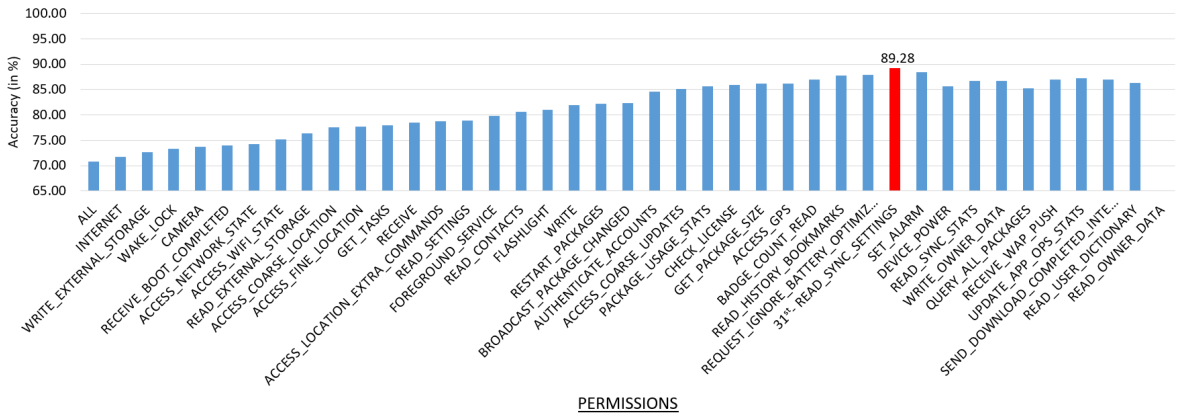


Figure 3.17: Detection results with *WASPAS* using permissions

Detection Results with *WASPAS* using intents

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked intents by *WASPAS*. The algorithm will give the best intents with higher accuracy as an output. Figure 3.18 can be understood as follows. While considering all the intents simultaneously without utilizing the *WASPAS* ranking, we achieve 65.35% accuracy with RF. At the first iteration, on eliminating the least ranked intent named *BATTERY_CHANGED* from the *DATASET-2*, we observe that we get the same 65.54% accuracy with RF and BC classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked intents, i.e., *BATTERY_CHANGED* and *MEDIA_REMOVED* from the *DATASET-2*. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked intents and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Figure 3.18, we achieved the highest detection accuracy of 95.58% with RF classifier upon eliminating 53 intents out of the total lot of 79, i.e., upon considering only the top 26 intents namely {*MESSAGE_ARRIVED* , *ELECTION_RESULT_V4* , *SCREEN_ON*

, *webview* , *HOME* , *PUSH_TIME* , *NOTIFICATION_OPENED* , *PushService.....PHONE_STATE* , *REGISTER* , *PACKAGE_REMOVED* , *SERVICE* , *MEDIA_EJECT* and *HEADSET_PLUG* } , highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we get the highest accuracy of 95.58% when we apply the proposed Algorithm 1 on intents.

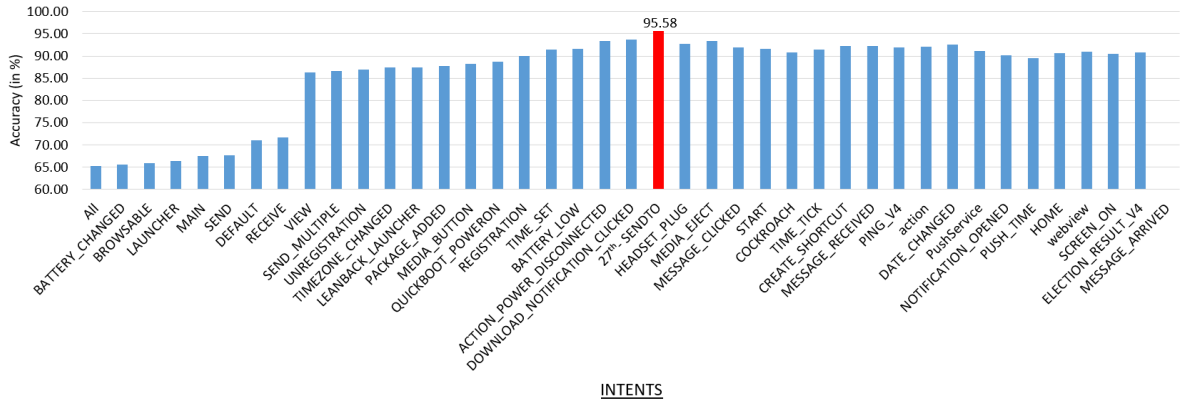


Figure 3.18: Detection results with *WASPAS* using intents

Detection Results with *WASPAS* using hardware components

Next, we apply the proposed detection algorithm (Algorithm 1) with ranked hardware components by *WASPAS*. The algorithm will give the best hardware components with higher accuracy as an output. Figure 3.19 can be understood as follows. While considering all the hardware components simultaneously without utilizing the *WASPAS* ranking, we achieve 65.39% accuracy with BC machine learning classifiers. At the first iteration, on eliminating the least ranked hardware component named *sensor.compass* from the *DATASET-2*, we observe that we get 64.93% accuracy with DT and RF classifiers. We call this the first iteration and move on to the next iteration when we eliminate the bottom two ranked hardware components, i.e., *sensor.compass* and *camera.ar* from the *DATASET-2*. As discussed in Algorithm 1, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked hardware components and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Figure 3.19, we achieved the highest detection accuracy of 94.44% with DT, BC, and RF classifiers upon eliminating 71 hardware components out of the total lot of 88, i.e., upon considering only the top 17 hardware components namely {*faketouch.multitouch.jazzhand sensor.ambient_temperature* , *sensor.heartrate.ecg* , *sensor.relative_humidity* , *type.automotive* , *BLUETOOTH_ADMIN* , *portrait*,.....*sensor.ACCELEROMETER* , *camera.capability.manual_post_processing*

, *camera.capability.manual_sensor* , *moxx.mobility.android.hardwareplatform.* *firebaseinitprovider* , *READ_EXTERNAL_STORAGE* }, highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we conclude that we get the highest accuracy of 94.44% when we apply the proposed Algorithm 1 on hardware components.

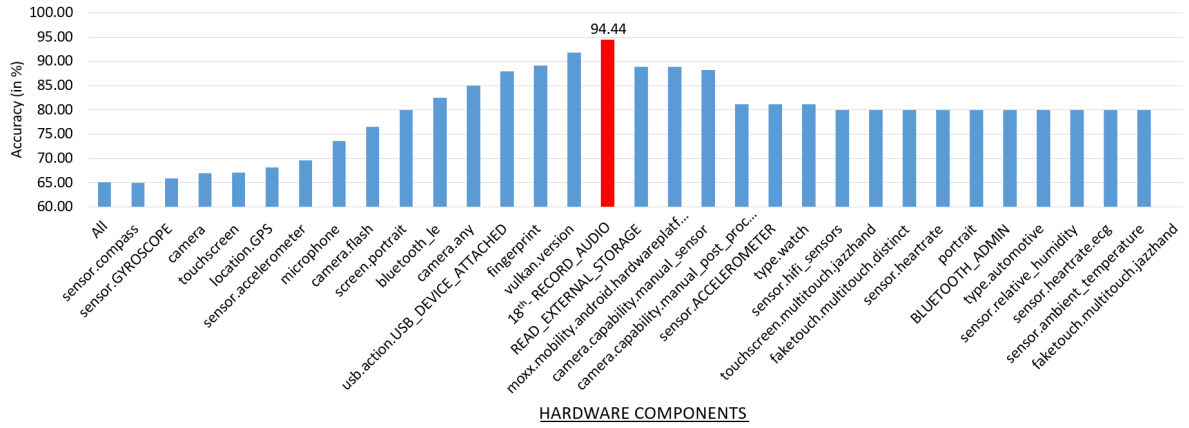


Figure 3.19: Detection results with *WASPAS* using hardware components

The compiled detection results when we apply the proposed algorithm to *DATASET-2* are summarized in Table 3.28. From the table, we observe that we obtain the highest accuracy of 89.28% using 30 permissions when we apply the proposed Algorithm 1 to the ranking formulated by *WASPAS*. Similarly, the highest accuracy of 95.85% can be achieved using 46 intents when we apply the proposed Algorithm 1 to the ranking formulated by *TOPSIS*, whereas the ranking given by *TOPSIS* results in the highest detection accuracy of 94.44% using 16 hardware components. At the same time, when no feature ranking of any type is used and all features are fed to the classifiers at once, i.e., on simultaneously considering all permissions, intents, or hardware components, we observe that the highest detection accuracy obtained is merely 70.79%, 65.35%, and 65.39% respectively.

Table 3.28: Compiled Detection results (in %) on applying the proposed algorithm on *DATASET -2*

Feature Ranking Method used	**PERMISSIONS**		**INTENTS**		**HARDWARE COMPONENTS**	
	Number used	**Accuracy (in %)**	**Number used**	**Accuracy (in %)**	**Number used**	**Accuracy (in %)**
TOPSIS	06	87.89	**46**	**95.85**	**16**	**94.44**
EDAS	09	88.67	19	93.72	17	94.44
WASPAS	**30**	**89.28**	26	95.58	17	94.44
No Ranking (All features used)	129	70.79	79	65.35	88	65.39

Hence, in response to research question four, we conclude that *TOPSIS*' top-ranked 46 intents, i.e., intents, give the best detection accuracy of 95.85% in the case of the unknown dataset.

3.6 Discussion

In the upcoming subsections, we compare the performance of our proposed model with some existing literature works in the field of Android malware detection, followed by discussing a few limitations of our proposed approach.

3.6.1 Comparison with other related works

Table 3.29 annotates the performance of our proposed model with some existing literature works in the field of Android malware detection that have used permissions, intents, or hardware components as features. As shown in the table, our work outperforms all these studies in terms of detection accuracy. If we take a closer look at some of the studies, we observe that researchers have ranked the features based on frequency or with tests such as Mutual Information and Pearson Correlation Coefficient in the past. Other studies have used ML-based feature selection techniques, whereas some authors have formed permission pairs for Android malware detection. Only three studies, Li et al. [16], Arp et al. [69] and Wang et al. [131], have used a larger number of normal applications in their analysis than ours. However, the dataset size for malware apps is still relatively small. Moreover, our work outperforms them in terms of detection accuracy. Hence, our proposed model is better than many state-of-the-art techniques presented in the literature for Android malware detection.

3.6.2 Limitations

In this section, we elucidate a few limitations of the proposed approach. In simple words, our model aims and successfully ranks features such as permissions, intents, and hardware components to detect Android malware without actually executing the code of an application; hence, the model falls under the category of static detection. Consequently, our model has the same shortcomings as any static detection model. Although static techniques prove to be quite efficient in terms of ease during the extraction of features as well as in terms of expenses, they still fall short when dealing with advanced malware behaviors such as code obfuscation and dynamic code loading. Application collusion is an emerging threat to Android-based

Table 3.29: Comparison of proposed work with the existing literature based on malware detection using permissions, intents or hardware components.

Related Work	**Feature selection/Feature ranking technique used**	**Dataset Size**		**Detection accuracy (in %)**
		Normal	**Malware**	
Wang et al. [131]	Permissions ranking with Mutual Information, Correlation Coefficient and T-test	310,926	4,868	94.62
Li et al. [16]	Permissions ranking based on frequency	310,926	62,838	93.62
Mahindru and Sangal [56]	Feature selection using Gain Ratio, Filtered Subset selection, Information feature, LR analysis, PCA	5,00,000	-	98.2
Arp et al. [69]	Pattern analyzing via joint vector space	123,453	5,560	94
Feldman et al. [126]	ML-based classification model	307	307	90
Arora et al. [138]	Normal and malicious graphs of permission pairs	7,533	7,533	95.44
Talha et al. [18]	Risk score calculated for each app	1,853	6,909	88.28
Shang et al. [21]	Naive Bayes and Pearson Correlation Coefficient	945	1,725	86.54
Tchakounté et al. [22]	Sequence alignment based similarity score	534	534	79.58
Khariwal et al. [139]	Raked features using Information gain	1,414	1,714	94.73
***PHIGrader* (Proposed Model)**	**Feature Ranking with Frequency-based *TOPSIS* method**	**77,000**	**77,000**	**99.10**

devices that seem almost immune to static feature-based detection systems. In app collusion, two or more Android apps collude to perform a malicious action that they cannot accomplish independently. In this way, they perform malicious tasks without displaying malware behavior. As a result, some stealthier malware might evade the kind of detection proposed by our model. Hence, we will work on combining the merits of dynamic analysis and some safe systems for colluding apps with the shortcomings of static analysis to form a much more efficient malware detection system in our future work. Additionally, some of the techniques in the literature have focused on Android malware family/category classification such as [140] , [141] , [142] , [143] , [144] , [145] , [146] and [147]. In this work, however, we did not aim for malware family classification. Hence, we will aim to enhance the capabilities of our model in our future work by including malware family classification in addition to malware detection.

3.7 Conclusion and Future work

In this chapter, we aimed to evaluate the efficiency of the top three most commonly used static features from the *AndroidManifest* file when used for Android malware detection. We first

assigned weights to features based on their frequency difference in the malware and normal training datasets. Subsequently, we ranked the three weighted feature sets, i.e., permissions, intents, and hardware components, by applying *TOPSIS*, *EDAS* and *WASPAS*, *Multi-Criteria Decision-Making* techniques in order of preference. Finally, we proposed a novel algorithm to identify the best set of features and the best type of feature among them. Our experimental results indicate that intents rank first in terms of performance as a feature for Android malware detection. Furthermore, the results showed that *TOPSIS*, among the three proposed frequency-based *MCDM* techniques, gives an adequate detection accuracy of 99.10% with 46 intents. Moreover, our experiments indicate that the proposed frequency-based *MCDM* approach gives us better accuracy than the popularly used feature ranking methods such as Principal Component Analysis (PCA) and Entropy-based Category Coverage Difference (ECCD) and also better than other statistical tests such as Mutual Information, Pearson Correlation Coefficient, and T-test. In addition, we proved that our proposed method is better than many state-of-the-art techniques for Android malware detection in terms of detection accuracy. In our future work, we will address the limitations of static analysis by incorporating some dynamic analysis techniques. Additionally, we will aim to assess the effectiveness of the *MCDM* techniques across other tasks such as malware family detection.

Chapter 4

PHIAnalyzer: A novel Android malware detection system using ranked Manifest file components

---

In this chapter, we present *PHIAnalyzer*, a novel Android malware detection system that ranks permissions, intents, and hardware components using a frequency-based *Chi-Square* test. The detection algorithm then evaluates seven possible feature combinations—permissions alone, intents alone, hardware components alone, as well as all combinations to identify the best set of features achieving higher detection accuracy. Our experiments demonstrate that the proposed frequency-based *Chi-Square* ranking is better than other various statistical tests when applied to the same datasets. In Section 4.1, we explain the motivation behind the work done and a brief overview of the proposed methodology. The rest of the chapter is structured as follows. We explain in detail the proposed methodology in Section 4.2. Subsequently, we discuss the feature ranking, detection and comparison results of our proposed model in Section 4.3. Finally, we conclude the paper with future work directions in Section 4.4.

4.1 Introduction

Among all the components present within the *AndroidManifest* file of an application, the most important, influential, and widely used are permissions, intents, and hardware components. As clearly depicted by the detection results shown in Chapter 3, these features individually have demonstrated significant potential in achieving high detection accuracy. But simultaneously historical evidence underscores that combining different types of indicators has proven useful, as seen in multi-factor security approaches, where layering security measures helps catch threats that single-factor methods may overlook. An example of this synergy can be seen in location-tracking malware: by combining GPS permissions, messaging intents, and hardware access to sensors, this kind of malware can covertly track and transmit a user's location, an operation unlikely to be flagged when only one feature type is considered.

Theoretically, permissions focus on application-level access controls, which may overlook certain interactions that intents, responsible for inter-component communication, can capture. Similarly, hardware components provide insights into the physical capabilities accessed by the application, which may complement or reveal gaps left by permissions and intents. Therefore, combining these features leverages their unique strengths and mitigates their individual limitations, creating a more comprehensive detection framework. In this work, we aim to explore seven distinct combinations of these three feature types to identify the most optimal feature subset achieving higher detection accuracy.

4.1.1 Drawbacks of existing approaches

Several related works, such as [15] , [148], and [149], have used permissions as the main feature in the process of detecting Android malware. To talk about them in a bit of detail, Şahın et al. [15] used multiple linear regression methods while feeding permissions as inputs for their calculations and concluded their paper by comparing the results of their proposed permission-based classifiers with the machine learning ones. Alsoghyer and Almomani [148] worked on developing a detection model based upon the frequently used permissions in both normal and malware datasets, followed by applying the machine learning algorithms. In contrast, Shrivastava and Kumar [149] started with the same approach of using permission frequency but instead used it to calculate a particular risk score to classify applications as normal or malware. The authors in [150] and [151] worked on combining the two features, permissions,

and intents. More specifically, the authors in [150] developed a malware classification system that classified applications as normal or malware by observing the feature frequency, whereas in [151], the authors started with a similar approach of monitoring the frequency of most requested features but later used it to make a detection matrix as a part of the malware detection system.

None of the above works used the key concept of ranking the features and hence missed the feature reduction step, which could have enhanced the quality of their results. In several other related works, such as [16] and [139], the authors built a detection system using the ranking of features, be it permissions or permissions and intents combined. More specifically, Li et al. [16] worked on ranking the permissions that are being used in one type of dataset only, either normal or malware applications set by using the frequency method. Khariwal et al. [139] also worked on ranking the features, but they took it a step further by including the ranking of intents and the combined ranking of intents and permissions obtained from the Information Gain score in their research work. However, both works were implemented on a smaller set of malware applications as compared to the huge malware dataset in our proposed work. More importantly, our work outperforms both of them in terms of detection accuracy while using a lesser number of ranked features.

4.1.2 Objectives and Need of Proposed Approach

We aim to build a robust and efficient static analysis-based Android malware detection system capable of identifying malicious behavior of applications on Android smartphones. At the same time, we are driven to fulfill this objective using the least as well as the best combination of features only amongst the top three most commonly used static feature types, i.e., permissions, intents, and hardware components. Instead of using just one feature type, we have opted for a hybrid approach to choose the best feature combination, the reasons of which are twofold. Firstly, we believe generalizing a theory needs more than one tested scenario before it can to be called a fact. Similarly, to prove the robustness of our proposed algorithm, we checked it on seven possible feature combinations—permissions alone, intents alone, hardware components alone, as well as on all possible combinations. Secondly, experimental results indicate that combining different feature types can lead to elevated detection accuracy instead of using any of them individually, and as mentioned earlier, a malware detection model is as good as its detection accuracy.

Combining features can be a simple yet effective approach to detecting malicious applications. Therefore, this work aims to analyze permissions, intents, and hardware components while taking their frequency as input, ranking them using a statistical *Chi-Square* test, and further combining them to find the best subset of features achieving higher detection accuracy. The following research questions emerge in the light of proposing a detection model based on the ranking of manifest file features:

- **RQ1** Why do we need to rank the permissions, intents, and hardware components, and subsequently, why is feature reduction needed instead of feeding all the features as inputs?
- **RQ2** How to incorporate feature ranking, i.e., how to rank the permissions, intents, and hardware components?
- **RQ3** How to frame a detection approach based on the ranking of the three feature types used in this study?

We are motivated to answer these questions with a vision to develop an Android malware detector, named *PHIAnalyzer*, based on the combinations of ranked permissions, intents, and hardware components. We have used a frequency-based *Chi-Square* test to rank the three feature types. We have used the *Chi-Square* test because of its numerous advantages, such as its robust nature to the data distribution and comparatively more straightforward computation. Moreover, the *Chi-Square* test can handle data whose parametric assumptions cannot be met, irrespective of two-group or multiple-group studies. Further, we have proposed a novel detection algorithm that uses ranked permissions, intents, and hardware components and applies various machine learning and deep learning techniques to detect Android malware effectively. The work proposed in this chapter employs a mix of old and recent datasets for evaluation. Our detection results are better than many state-of-the-art techniques proposed in the existing literature. Moreover, our experiments demonstrate that the proposed *Chi-Square*-based feature ranking gives us better accuracy than the Mutual Information and Pearson Correlation Coefficient, which have been used in [131], which we evaluate against the same dataset of normal and malicious apps.

Contributions: The main contributions of this research are highlighted below:

- Firstly, we ranked the permissions, intents, and hardware components in order of their absolute frequency difference between the malware and normal dataset and used the

values as pre-requisite in the *Chi-Square* test.

- Next, we applied the frequency-based *Chi-Square* test on the permissions, intents, and hardware components and ranked them based on the F-score given as an output by the test.

- We proposed a novel algorithm to merge the individual rankings of permissions, intents, and hardware components to develop an efficient Android malware detection system.

- We observed that the detection results of the proposed approach are relatively better than various state-of-the-art techniques existing in the literature for Android malware detection.

4.2 System Design

In this section, we explain our proposed methodology in detail. Our proposed model *PHIAnalyzer* is divided mainly into two modules. In the first module, named as *Ranking Module*, we extract the permissions, intents, and hardware components from the training dataset and aim to rank them using a frequency-based *Chi-Square* test. Such a ranking will help us eliminate irrelevant features that negatively affect the detection accuracy. In the *Detection Module*, we propose a novel algorithm that applies machine learning and deep learning techniques to get the best features that can provide higher detection accuracy. The following subsections discuss in detail both modules of the proposed model.

RANKING MODULE

4.2.1 Data Acquisition and Representation

The datasets used for the validation of the proposed model *PHIAnalyzer* in this chapter, namely *DATASET-1* and *DATASET-2*, are the same as those detailed earlier in Subsection 3.2.1 of the Chapter 3. Following the extraction of the three features—permissions, intents, and hardware components, as detailed in 3.2.2—feature vector tables were constructed for representation. Each feature vector, formulated for individual applications, adopts a binary format, where *1* indicates the presence of a requested feature (permissions, intents, or hardware components), and *0* signifies its absence. In this way, we create three separate vector tables, one

each for permissions, intents, and hardware components, represented by P_{VT}, I_{VT} and H_{VT}, respectively. For instance, if there are a total of five permissions, say $<P_1, \ldots..P_5>$ and five intents say $<I_1, \ldots..I_5>$ in the system, and any application A_j has permissions P_1, P_2, P_5 and intents I_3, I_4, I_5, then the app A_j is represented as *11001* and *00111* in P_{VT} *and* I_{VT} respectively.

We observe that some features have a high frequency in normal or malware datasets. The frequency difference between the malware and normal dataset for any feature can give us valuable insights for feature ranking. Therefore, before applying the *Chi-Square* method to rank the features, we initially assign weights to all the permissions, intents, and hardware components, based on their absolute frequency difference in normal and malware datasets. Then, we take the absolute frequency difference for each feature in normal and malware datasets. For instance, if there are an x number of features, the feature with the highest absolute frequency difference will be assigned a weight of one, and the feature with the lowest frequency difference will be given a weight of x. The process is repeated separately for permissions, intents , and hardware components.

After assigning the weights to all the features, for every occurrence of *1* for each permission, intent, and hardware component, we replace *1* by its corresponding weight in P_{VT}, I_{VT} and H_{VT}. For instance, again, consider the same app A_j, which was initially represented as *11001* and *00111* in P_{VT} *and* I_{VT} respectively. Suppose the weights for $P_1, P_2,$ *and* P_5 are $\alpha_1, \alpha_2, \alpha_5$ respectively, and weights for I_3, I_4 *and* I_5 are $\beta_3, \beta_4, \beta_5$, then A_j is now represented as $\alpha_1\alpha_200\alpha_5$ and $00\beta_3\beta_4\beta_5$ in P_{VT} *and* I_{VT} respectively.

4.2.2 Features Ranking

We have used a statistical *Chi-Square* test to rank the features. Such a ranking helps eliminate irrelevant features, and their removal will help improve detection accuracy. The *Chi-Square* statistic is used to determine whether the variables of different categories defined are independent of each other. It can also be used to measure the significant difference between variables and their expected values. The *Chi-Square* test is specifically designed to assess the independence between two categorical variables. This makes it particularly suitable for feature ranking/selection when dealing with categorical or discrete data, similar to the dataset used in our work. Moreover, *Chi-Square* does not require equality of variances among the study groups or homoscedasticity in the data, nor does it form any prior assumptions about the distribution, making it a suitable method for feature ranking even in the presence of data with missing

values or measurement errors. The *Chi-Square* formula [152] is defined in the below equation.

$$\chi_c{}^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{4.2.1}$$

where:

c=Degrees of freedom,

O=Observed value(s), and

E=Expected value(s)

This test's null hypothesis says there is no link between the original and expected data. The alternate hypothesis states that the actual and expected data depend on each other. For a basic *Chi-Square* test of independence, where n denotes the number of observations and k is the number of categories, the computational complexity is generally considered to be O(nk). We apply the *Chi-Square* test on the three feature vector tables we have formulated for our training dataset, i.e., P_{VT}, I_{VT} and H_{VT}. The F score [1] that comes after applying the *Chi-Square* test on a categorical type data can be very efficiently used to select the best set of features, amongst all features, by ranking them from highest to lowest F score value. The feature that can better distinguish normal and malware datasets will have a higher F-score value. We apply this ranking technique separately on permissions, intents, and hardware components, and we get, as an output of this module, three ranked lists, P_{List}, I_{List} and H_{List}, one each for permissions, intents, and hardware components, respectively.

4.2.3 Machine Learning and Deep Learning Classifiers

We have used several machine learning and deep learning classifiers [135] in our detection approach. We applied nine widely used techniques, namely Decision Trees (DT), Random Forest (RF), Bagging classifier (BC), Gaussian Naive Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM) as machine learning classifiers and Multilayer Perceptron (MLP), Artificial Neural Networks (ANN), Dense Neural Network (DNN) as deep learning classifiers.

All experiments with these classifiers were performed using ten-fold cross-validation [136]. The code concludes by printing the cross-validation results, including the accuracy scores for

[1] https://scikit-learn.org/stable/modules/generated/ sklearn.feature_selection.chi2.html

Algorithm 2 Proposed Malware Detection Algorithm

Input: $P_{List} \leftarrow$, Ranked Permissions List $I_{List} \leftarrow$ Ranked Intent List, $H_{List} \leftarrow$ Ranked Hardware component List
Output: Best set of features with higher detection rate
$BestFeatures \leftarrow$ Blank List
$Comb_{List} \leftarrow$ Blank List
$NP_{List} \leftarrow$ Number of Permissions in P_{List}
$NI_{List} \leftarrow$ Number of Intents in I_{List}
$NH_{List} \leftarrow$ Number of Hardware components in H_{List}
$P_{all} \leftarrow$ List of all permissions from testing dataset (non unique)
$I_{all} \leftarrow$ List of all intents from testing dataset (non unique)
$H_{all} \leftarrow$ List of all Hardware components from testing dataset (non unique)
$D_{Max} \leftarrow$ Maximum accuracy obtained, initialized to zero.
$D_{Acc} \leftarrow$ Accuracy obtained after each iteration.
for i $\leftarrow$ 1 to NP_{List} **do**
    Insert P_i in BestFeatures
    FindAll P_i in P_{all}, **CopyAll**
    Insert P_i in $Comb_{List}$
    for j $\leftarrow$ 1 to NI_{List} **do**
        when j=i do
        Insert I_j in BestFeatures
        FindAll I_j in I_{all}, **CopyAll**
        Insert I_j in $Comb_{List}$
        for k $\leftarrow$ 1 to NH_{List} **do**
            when k=j do
            Insert H_k in BestFeatures
            FindAll H_k in H_{all}, **CopyAll**
            Insert H_k in $Comb_{List}$
            Find D_{Acc} using ML algorithms for features present in $CombList$
            if $D_{Acc} > D_{Max}$ **then**
                $D_{Acc} = D_{Max}$
                else exit
            end if
        end for
    end for
end for
return $BestFeatures$
return D_{Max}

each fold and the mean accuracy across all folds. This provides insights into the model's consistency and overall performance across diverse subsets of the dataset.

DETECTION MODULE

4.2.4 Proposed Malware Detection Algorithm

This section describes our detection algorithm, summarized in Algorithm 2. As discussed in the previous subsection, we compute the F score and determine the features' relevance. The higher the F score value, the higher the relevancy. Hence, we rank the features in decreasing order of their F score values. We aim to find the best subset of features to give better detection accuracy. P_{List}, I_{List}, and H_{List} represent the ranked permissions, intents, and hardware components in decreasing order of their F score values.

In the first iteration of the algorithm, we select the top-ranked permission, intent, and hardware component from P_{List}, I_{List}, and H_{List} respectively. We then execute machine learning

algorithms on the testing data by considering only these three features, i.e., top-ranked permission, intent, and hardware component and observe the detection accuracy, say D_{Acc}. The maximum accuracy, say D_{Max}, is initialized to zero. At every iteration, we compare D_{Acc} and D_{Max}. If the accuracy at the current iteration, i.e., D_{Acc}, is higher than D_{Max}, we proceed towards the next iteration, and we set D_{Max} as D_{Acc}.

In the next iteration, we select the top two ranked permissions, intents, and hardware components and find the detection accuracy on the testing data by considering these six features, i.e., D_{Acc} for the current iteration. Again, we compare the D_{Max} and D_{Acc}, and if D_{Acc} is higher than D_{Max}, we proceed toward the next iteration to select top three ranked permissions, intents, and hardware components. The algorithm continues the same way and terminates when the detection accuracy does not improve further. At a stage when D_{Acc} is not higher than D_{max}, we return the D_{Max} and the best set of permissions, intents, and hardware components. From the proposed approach, the best set of features will always contain the equal number of feature types. Overall, the computational complexity of the proposed malware detection algorithm can be expressed as O((NP_{List} + NI_{List} + NH_{List}) * M * f(N)), where NP_{List} is the number of permissions in P_{List}, NI_{List} is the number of intents in I_{List}, NH_{List} is the number of hardware components in H_{List}, M is the maximum number of permissions or intents or hardware components in the testing dataset, and f(N) is the time complexity of the machine learning algorithm used. **This answers our research question three, i.e, how to frame a detection approach based on the ranking of permissions, intents, and hardware components. We describe the results obtained from the proposed approach in the next section.**

4.3 Results and Discussion

In this section, we showcase and discuss the experimental results obtained from the proposed *PHIAnalyzer* model. We point out that we have separate datasets for training and testing. As described in Section 4.2.1, we have 77,000 applications, each in the normal and malware category. Out of them, we use 56,000 normal apps and 56,000 malware apps in the ranking module. The remaining 21,000 normal and 21,000 malware apps are used in the detection module. We name this dataset *DATASET-1*. Additionally, considered a second dataset called *DATASET-2*, which contains recent and stealthier malware samples detected in 2021 and 2022. In the upcoming subsections, first, we discuss the ranking obtained from the frequency-based *Chi-Square* test, and after that, we describe the detection results on *DATASET-1* and *DATASET-2*.

Further, we also compare our proposed work with similar works in Android malware detection.

4.3.1 Allotting Weights To The Features

Firstly, we assign weights to the three feature types based on their absolute frequency difference in the normal and malware training dataset. We note that we have three separate rankings, one each for permissions, intents, and hardware components. Tables 4.1 , 4.2 and 4.3 summarize the top ten permissions, intents, and hardware components respectively, along with their frequency difference and weights. As seen from Table 4.1, permission named *MOUNT_UNMOUNT_FILESYSTEMS* is assigned the weight of one as it has the highest frequency difference in both datasets. Similarly, we can acknowledge the weights of other top permissions from the table. The permission named *SET_WALLPAPER* had the lowest frequency difference of 10 and hence, had the highest weight of 129, amongst all 129 permissions.

Table 4.1: Top 10 permissions along with their corresponding weights.

PERMISSIONS	**Weights allotted according to ranking**	**Normal frequency**	**Malware frequency**	**Absolute difference**
MOUNT_UNMOUNT_ FILESYSTEMS	1	1264	41324	40060
READ_PHONE_STATE	2	14176	53586	39410
GET_TASKS	3	4527	43399	38872
CHANGE_WIFI_STATE	4	4337	43165	38828
SYSTEM_ALERT_WINDOW	5	4561	38594	34033
WRITE_SETTINGS	6	7258	39497	32239
CHANGE_NETWORK_ STATE	7	2745	30874	28129
READ_LOGS	8	1071	29112	28041
ACCESS_COARSE_ LOCATION	9	16650	42425	25775
ACCESS_WIFI_STATE	10	28554	53886	25332

As can be seen from Table 4.2, the intent named *USER_PRESENT* is assigned the weight of one as it has the highest frequency difference in both datasets. Similarly, we can acknowledge the weights of other top intents from the table. The intent named *MAIN* had the lowest frequency difference of 87 and hence, had the highest weight of 79, amongst all 79 intents.

Table 4.2: Top 10 intents along with their corresponding weights.

INTENTS	**Weights allotted according to ranking**	**Malware frequency**	**Normal frequency**	**Absolute difference**
USER_PRESENT	1	33108	1894	31214
PACKAGE_REMOVED	2	26806	963	25843
DEFAULT	3	45689	21291	24398
PACKAGE_ADDED	4	21111	2689	18422
VIEW	5	35548	18922	16626
BROWSABLE	6	33915	17545	16370
REGISTER	7	14419	134	14285
NOTIFICATION_RECEIVED _PROXY	8	14139	91	14048
PushService	9	14004	84	13920
PUSH_TIME	10	13998	84	13914

As can be seen from Table 4.3, the hardware component named *touchscreen* is assigned the weight of one as it has the highest frequency difference in both datasets. Similarly, we can acknowledge the weights of other top features from the table. The hardware component named *type.watch* had the lowest frequency difference of 1 and hence, had the highest weight of 88, amongst all 88 intents.

Table 4.3: Top 10 hardware components along with their corresponding weights.

HARDWARE COMPONENTS	**Weights allotted according to ranking**	**Normal frequency**	**Malware frequency**	**Absolute difference**
touchscreen	1	12147	2101	10046
camera	2	12337	21063	8726
Camera.autofocus	3	10446	19080	8634
touchscreen.multitouch	4	8999	1334	7665
touchscreen.multitouch.distinct	5	8765	1256	7509
location.network	6	7103	825	6278
location.GPS	7	7468	1262	6206
location	8	6223	413	5810
telephony	9	4725	674	4051
screen.portrait	10	4136	160	3976

4.3.2 Features Ranking

To identify the distinguishing features, we separately applied the statistical *Chi-Square* test on P_{VT}, I_{VT} and H_{VT} . The *Chi-Square* test, as its output, calculates the corresponding F score values for all the features. Further, we used these F score values to rank the features such that the feature with the highest F score value is the top-ranked feature and hence, the most distinguishing one. Tables 4.4 , 4.5 and 4.6 summarize the top ten permissions, intents, and hardware components respectively according to their F-scores obtained from the *Chi-Square* test. **Using the frequency-based *Chi-Square* method, we answer our research question two, i.e., how to rank the features to identify the distinguishing ones among them.**

Table 4.4 highlights that the permission named *BIND_GET_INSTALL_ REFERRER_SERVICE* is the most distinguishing permission with the highest F-score. Similarly, we can infer rankings of other permissions based on their F-scores from the table. The permission named *SET_WALLPAPER* had the lowest F score value of 13.803 amongst all permissions and hence, is the least distinguishing permission.

Similarly, Table 4.5 highlights that the intent named *CONNECTION* is the most distinguishing intent with the highest F-score. Similarly, we can infer rankings of other intents based on their F-scores from the table. The intent named *MAIN* had the lowest F-score value of 0.9848 and hence, is the least distinguishing intent.

Table 4.4: Top 10 permissions with their corresponding F-scores

PERMISSIONS	F Score
BIND_GET_INSTALL_REFERRER_SERVICE	232552.1
JPUSH_MESSAGE	232152
RESTART_PACKAGES	229418.5
SEND_SMS	228428.9
RECEIVE_SMS	217999.7
READ_SMS	216163.5
CHANGE_CONFIGURATION	213456.9
RECEIVE_USER_PRESENT	213214.9
BROADCAST_PACKAGE_INSTALL	211856.9
BROADCAST_PACKAGE_REPLACED	211465.4

Table 4.5: Top 10 intents with their corresponding F-scores

INTENTS	F Score
CONNECTION	136262.7
DaemonService	129391.5
NOTIFICATION_RECEIVED	125791.8
NOTIFICATION_OPENED	117317.9
MESSAGE_RECEIVED	108364.1
START_FROM_AGOO	107091.5
REPORT	105758.9
COMMAND	101132.6
SERVICE	98767.32
ELECTION	98598.16

Table 4.6 highlights that the hardware component named *location* is the most distinguishing one with the highest F-score. Similarly, we can infer rankings of other features based on their F-scores from the table. The hardware component named *type.watch* had the lowest F-score value and hence, is the least distinguishing one.

Table 4.6: Top 10 hardware components with their corresponding F-scores

HARDWARE COMPONENTS	F Score
location	28691.51
screen.portrait	25523.57
vulkan	22279.86
location.GPS	21811.74
location.network	20900.12
touchscreen.multitouch.distinct	20024.67
telephony	19259.59
bluetooth	17616.42
screen.lanscape	17395.16
nfc.hce	16804.3

In the following subsection, we present the detection results obtained with the proposed model.

4.3.3 Detection Results on *DATASET-1*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over the *DATASET-1*. For comparison, we perform seven experiments, considering 1)

permissions alone, 2) intents alone, 3) hardware components alone, 4) permissions and intents combined, 5) intents and hardware components, 6) permissions and hardware components, and 7) permissions, intents and hardware components, all three of them combined. We discuss these results in upcoming subsections.

Detection with Permissions Alone

First, we apply the proposed detection algorithm (Algorithm 2) with permissions alone. The algorithm will give the best permissions with higher accuracy as an output. Table 4.7 summarizes the detection results when we use permissions alone for detection. The table can be understood as follows. With the top-ranked permission, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE*, we get 95.55% accuracy with several machine learning classifiers. We call this the first iteration, then we move to the next iteration when we consider the top two ranked permissions, i.e., combining *BIND_GET_INSTALL_REFERRER_SERVICE* with *JPUSH_MESSAGE* for detection and repeat the process mentioned above. In this iteration, we get an accuracy of 96.96% from several machine learning classifiers. As discussed in Algorithm 2, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we consider the top three permissions and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Table 4.7, we achieved the highest detection accuracy on the tenth iteration, i.e., upon adding the top ten permissions, namely { *BIND_GET_INSTALL_REFERRER_ SERVICE* , *JPUSH_ MESSAGE* , *RESTART_PACKAGES* , *SEND_SMS* , *RECEIVE_SMS* , *READ_SMS* , *CHANGE_CONFIGURATION* , *RECEIVE_ USER_PRESENT* , *BROADCAST_PACKAGE_INSTALL* , *BROADCAST_PACKAGE_REPLACED* }, we get the highest accuracy of 97.70%. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we observe that we get the highest accuracy of 97.70% when we apply the proposed Algorithm 2 only on permissions.

Detection with Intents Alone

Next, we apply the proposed detection algorithm (Algorithm 2) with intents alone. The algorithm will give the best intents with higher accuracy as an output. Table 4.8 summarizes the detection results when we use intents alone for detection. With the top-ranked intent, i.e., *CONNECTION*, we get 95.27% accuracy with all the classifiers. We call this the first iteration, then we move to the next iteration when we consider the top two ranked intents, i.e., combining *CONNECTION* with *DaemonService* for detection and repeating the above-mentioned process.

Table 4.7: Detection results with proposed approach considering only permissions

PERMISSIONS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_REFERRER_SERVICE	95.55	95.55	94.29	95.55	95.55	95.55	94.60	94.60	94.60
JPUSH_MESSAGE	96.69	96.69	51.71	96.69	96.69	96.69	96.96	96.96	96.96
RESTART_PACKAGES	96.86	96.86	60.68	96.86	96.86	96.86	96.79	96.79	96.79
SEND_SMS	97.06	97.06	66.59	97.06	97.06	97.06	97.05	97.05	97.25
RECEIVE_SMS	97.42	97.42	70.83	97.42	97.42	97.42	97.25	97.25	97.25
READ_SMS	97.42	97.42	74.46	97.42	97.42	97.42	97.46	97.46	97.46
CHANGE_CONFIGURATION	97.29	97.29	77.15	97.29	97.29	97.29	97.27	97.27	97.27
RECEIVE_USER_PRESENT	97.39	97.39	70.16	97.39	97.39	97.39	97.41	97.41	97.41
BROADCAST_PACKAGE_INSTALL	97.60	97.60	71.63	97.60	91.65	97.60	97.50	97.50	97.50
BROADCAST_PACKAGE_REPLACED	**97.70**	**97.70**	74.54	**97.70**	92.16	**97.70**	97.58	97.58	97.58
BROADCAST_STICKY	97.20	97.20	91.70	97.20	91.91	97.20	97.14	97.14	97.14
PROCESS_OUTGOING_CALLS	97.32	97.32	78.10	97.32	92.55	97.32	97.22	97.22	97.22

In this iteration, we note that we get an accuracy of 95.35% from all the machine learning classifiers. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Table 4.8, we achieved the highest detection accuracy on the second iteration, i.e., upon adding the top two intents, namely *CONNECTION* and *DaemonService*, we get the highest accuracy of 95.35%. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we observe that we get the highest accuracy of 95.35% when we apply the proposed Algorithm 2 only on intents.

Table 4.8: Detection results with proposed approach considering only intents

INTENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
CONNECTION	95.27	95.27	94.33	95.27	95.27	95.27	95.21	95.21	95.21
DaemonService	**95.35**	**95.35**	65.32	**95.35**	**95.35**	**95.35**	95.30	95.30	95.30
NOTIFICATION_RECEIVED	95.27	95.27	72.99	95.27	95.27	95.27	95.32	95.32	95.32
NOTIFICATION_OPENED	95.20	95.20	75.78	95.20	95.20	95.20	95.05	95.05	95.05

Detection with Hardware components Alone

Next, we apply the proposed detection algorithm (Algorithm 2) with hardware components alone. Table 4.9 summarizes the detection results when we use hardware components alone for detection. With the top-ranked hardware component, i.e., *location*, we get 92.78% accuracy with all the classifiers. We call this the first iteration, then we move to the next iteration when we consider the top two ranked hardware components, i.e., combining *location* with *screen.portrait* for detection and repeating the above mentioned process. In this iteration, we note that we get an accuracy of 89.87% from all the machine learning classifiers. As shown in Table 4.9, we achieved the highest detection accuracy on the first iteration itself, i.e., upon

adding the top hardware component, namely *location*. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we observe that we get the highest accuracy of 92.78% when we apply the proposed Algorithm 2 only on hardware components.

Table 4.9: Detection results with proposed approach considering only hardware components

HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
location	**92.78**	**92.78**	**92.78**	**92.78**	**92.78**	**92.78**	**92.78**	**92.78**	**92.78**
screen.portrait	89.87	89.87	89.87	89.87	89.87	89.87	89.87	89.87	89.87
vulkan	89.01	89.01	89.01	89.01	89.01	89.01	89.01	89.01	89.01

Detection with Combination of Permissions and Intents

Further, we apply the proposed detection algorithm (Algorithm 2) on the combination of permissions and intents. Table 4.10 summarizes the detection results when we use permissions and intents for detection. With the top-ranked pair, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE* and *CONNECTION*, we get 96.56% accuracy with several classifiers. In the second iteration, we get an accuracy of 97.92% from several classifiers. Hence, next, we consider the top three pairs of permissions and intents and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy, and as shown in Table 4.10, we achieved the highest detection accuracy on the sixth iteration, i.e., upon adding six permissions, namely { *BIND_GET_INSTALL_REFERRER_SERVICE* , *JPUSH_MESSAGE*, *RESTART_PACKAGES* , *SEND_SMS* , *RECEIVE_SMS* , *READ_SMS* and six intents namely *CONNECTION* , *DaemonService* , *NOTIFICATION_RECEIVED* , *NOTIFICATION_OPENED* , *MESSAGE_RECEIVED* and *START_FROM_AGOO* } ,we get the highest accuracy of 98.49%. From the next iteration, we observe that the detection accuracy starts decreasing. We observe that we get the highest accuracy of 98.49% when we apply the proposed Algorithm 2 on the set of 12 features that contains six permissions and six intents.

Detection with combination of Intents and Hardware components

Next, we apply the proposed detection algorithm (Algorithm 2) on the combination of intents and hardware components. Table 4.11 summarizes the detection results when we use intents and hardware components for detection. With the top-ranked pair, i.e., *CONNECTION* and *location*, we get 97.21% accuracy with several classifiers. In the second iteration, we get an accuracy of 97.09% from several classifiers. Hence, next, we consider the top three pairs of intents and hardware components and repeat the entire procedure. Subsequently, as shown in Table 4.11, we achieved the highest detection accuracy on the third iteration, i.e., upon adding

Table 4.10: Detection results with proposed approach considering the combination of permission and intents

PERMISSIONS and INTENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_ REFERRER_SERVICE and CONNECTION	96.56	96.56	56.65	96.56	96.56	96.56	96.55	96.55	96.55
JPUSH_MESSAGE and DaemonService	97.90	97.90	70.34	97.90	97.90	97.90	97.92	97.92	97.92
RESTART_PACKAGES And NOTIFICATION_RECEIVED	98.02	98.02	97.90	98.02	98.02	98.02	97.92	97.92	97.92
SEND_SMS and NOTIFICATION_OPENED	98.32	98.32	82.65	98.32	98.32	98.32	98.14	90.84	98.14
RECEIVE_SMS and MESSAGE_RECEIVED	98.41	98.41	85.38	98.41	98.41	98.41	98.29	98.29	98.29
READ_SMS and START_FROM_AGOO	**98.49**	**98.49**	86.84	**98.49**	**98.49**	**98.49**	98.35	98.35	98.35
CHANGE_CONFIGURATION and REPORT	98.27	98.27	88.37	98.27	88.92	98.27	88.26	88.26	98.26
RECEIVE_USER_ PRESENT and COMMAND	98.26	98.26	90.18	98.26	90.32	98.26	90.21	90.21	98.29

three intents, namely { *CONNECTION* , *DaemonService* and *NOTIFICATION_RECEIVED* and three hardware components namely *location* , *screen.portrait* and *vulkan* } ,we get the highest accuracy of 97.43%. From the next iteration, we observe that the detection accuracy starts decreasing. We observe that we get the highest accuracy of 97.43% when we apply the proposed Algorithm 2 on the set of 6 features that contains three intents and three hardware components.

Table 4.11: Detection results with proposed approach considering the combination of intents and hardware components

INTENTS and HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
CONNECTION and location	97.21	97.21	97.21	97.21	97.21	97.21	97.21	97. 21	97.21
DaemonService and screen.portrait	97.09	97.09	97.09	97.09	97.09	97.09	97.09	97.09	97.09
NOTIFICATION_RECEIVED and vulkan	**97.43**	**97.43**	**97.43**	**97.43**	**97.43**	**97.43**	**97.43**	**97.43**	**97.43**
NOTIFICATION_OPENED and location.GPS	96.34	96.34	96.34	96.34	96.34	96.34	96.34	96.34	96.34
MESSAGE_RECEIVED and location.network	96.5	96.5	96.5	96.5	96.5	90.75	96.5	90.75	96.5

Detection with Combination of Permissions and Hardware components

Further, we apply the proposed detection algorithm (Algorithm 2) on the combination of permissions and hardware components. Table 4.12 summarizes the detection results when we use permissions and hardware components for detection. With the top-ranked pair, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE* and *location*, we get 94.15% accuracy with several classifiers. In the second iteration, we get an accuracy of 95.41% from several classifiers. As shown in Table 4.10, we achieved the highest detection accuracy on the second itera-

tion, i.e., upon adding two permissions, namely *BIND_GET_INSTALL_REFERRER_SERVICE* and *JPUSH_MESSAGE* and two hardware components namely *location* and *screen.portrait*. From the next iteration, we observe that the detection accuracy starts decreasing. We observe that we get the highest accuracy of 95.41% when we apply the proposed Algorithm 2 on the set of four features that contains two permissions and two hardware components.

Table 4.12: Detection results with proposed approach considering the combination of permissions and hardware components

PERMISSIONS and HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_REFERRER _SERVICE and location	94.15	94.15	94.15	94.15	94.15	94.15	94.15	94.15	94.15
JPUSH_MESSAGE and screen.portrait	**95.41**	**95.41**	**95.41**	**95.41**	**95.41**	94.05	**95.41**	94.05	**95.41**
RESTART_PACKAGES and vulkan	95.37	95.37	95.36	95.37	95.36	83.23	95.36	80.32	95.36
SEND_SMS and location.GPS	94.56	94.56	94.55	94.56	71.22	71.22	94.55	90.54	94.55

Detection with Combination of Permissions, Intents and Hardware components

Further, we apply the proposed detection algorithm (Algorithm 2) on the combination of permissions, intents, and hardware components. Table 4.13 summarizes the detection results when we use permissions, intents, and hardware components for detection. With the top-ranked trio, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE* , *CONNECTION* and *location*, we get 95.93% accuracy with several classifiers. We call this the first iteration and then move to the next iteration when we consider the top two ranked trio of permissions, intents, and hardware components, i.e., combining *BIND_GET_INSTALL_REFERRER_ SERVICE* , *CONNECTION* and *location* with *JPUSH_MESSAGE* , *DaemonService* and *screen.portrait* for detection and repeat the process mentioned above. In this iteration, we get an accuracy of 96.89% from several classifiers. Hence, next, we consider the top three pairs of permissions and intents and repeat the entire procedure. As shown in Table 4.13, we achieved the highest detection accuracy on the third iteration, i.e., upon adding three permissions, namely { *BIND_GET_INSTALL_REFERRER_SERVICE* , *JPUSH_MESSAGE* and *RESTART_PACKAGES* }, three intents namely { *CONNECTION* , *DaemonService* and *NOTIFICATION_RECEIVED* } and three hardware components namely, { *location* , *screen.portrait* and *vulkan* }, we get the highest accuracy of 96.99%. From the next iteration, we observe that the detection accuracy starts decreasing.

On comparing the highest accuracies obtained with permissions alone (97.70%), intents alone (95.35%), hardware components alone (92.78%), combination of intents and hardware

Table 4.13: Detection results with proposed approach considering the combination of permissions, intents and hardware components

PERMISSIONS, INTENTS and HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_REFERRER _SERVICE, CONNECTION and location	95.93	95.93	95.93	95.93	95.93	88.09	95.93	89.54	95.93
JPUSH_MESSAGE, DaemonService and screen.portrait	96.86	96.86	96.86	96.86	89.06	88.03	96.86	86.52	96.86
RESTART_PACKAGES, NOTIFICATION_RECEIVED and vulkan	**96.99**	**96.99**	**96.99**	**96.99**	91.34	72.34	**96.99**	75.85	**96.99**
SEND_SMS, NOTIFICATION_OPENED and location.GPS	96.73	96.73	94.61	96.73	82.67	73.66	96.73	72.82	96.73
RECEIVE_SMS, MESSAGE_RECEIVED and location.network	96.33	96.33	94.69	96.33	83.92	75.97	96.33	70.68	96.04

components (97.43%), combination of permissions and hardware components (95.41%), combination of permissions, intents and hardware components (96.99%) we find that the combination of permissions and intents (98.49%) gives us better detection accuracy as compared to all three features when used alone or in other possible combinations.

Note that, according to our detection approach, we consider one pair or permission and intent in each iteration while performing the experiments to identify the best combination of features, which subsequently leads to the case of only an equal number of permissions and intents irrespective of the number of iterations. For instance, from Table 4.10, we get the highest accuracy of 98.49% with the combination of six permissions and six intents, i.e., 12 features. Hence, to cross-check our approach, we compare the detection accuracy of features in other combinations of 12, such as five permissions with seven intents, four permissions with eight intents, three permissions with nine intents, and two permissions with ten intents, and vice versa. Moreover, we consider other combinations when the total number of features used differs from 12, i.e., 10, 11, 13, 14, and 15. Finally, we summarize all these results in Table 4.14. From the table, we observe that the best set of six permissions and six intents obtained from our proposed approach proves to be better in terms of detection accuracy than other combinations of permissions and intents. Hence, our model outperforms different combinations of permissions and intents, and we find that we get the highest accuracy of 98.49% with the top six permissions and top six intents combined.

Comparison with other statistical tests

Table 4.15 summarizes the detection results when we use all permissions and intents for detection without applying any feature ranking technique. The table can be understood as follows.

On considering all the permissions simultaneously without utilizing frequency-based *Chi-Square*

Table 4.14: Comparison of proposed frequency-based *Chi-Square* test in terms of detection accuracy upon using different combinations of features.

Combination of features	**Total number of features used**	**Detection accuracy (in %)**
6 permissions - 4 intents	10	98.16
4 permissions - 6 intents	10	98.42
5 permissions - 6 intents	11	98.38
6 permissions - 5 intents	11	98.31
5 permissions - 7 intents	12	98.48
7 permissions - 5 intents	12	98.18
4 permissions – 8 intents	12	98.36
8 permissions- 4 intents	12	98.10
3 permissions - 9 intents	12	98.36
9 permissions - 3 intents	12	97.93
2 permissions - 10 intents	12	98.40
10 permissions -2 intents	12	97.89
Proposed approach 6 permissions - 6 intents	**12**	**98.49**
7 permissions - 6 intents	13	98.23
6 permissions - 7 intents	13	98.44
6 permissions - 8 intents	14	98.39
8 permissions - 6 intents	14	98.24
10 permissions - 5 intents	15	98.37
5 permissions - 10 intents	15	98.47

feature ranking, we observe that the highest detection accuracy obtained is 78.64%, whereas the highest detection accuracy obtained while considering all intents is 67.18%. The highest detection accuracy recorded while considering all hardware components is 71.84%. Such a low detection accuracy highlights the importance of ranking permissions and intents because such ranking helps us eliminate the irrelevant features that can hamper detection accuracy. This answers our research question one, i.e., why do we need to rank the features.

We have applied the frequency-based *Chi-Square* test to rank permissions, intents, and hardware components in this chapter. However, statistical tests such as Mutual Information and Pearson Correlation Coefficient have been used in other works such as [131] for Android malware detection. Hence, next, we compare the performance of the proposed frequency-based *Chi-Square* test with the Mutual Information and Pearson correlation Coefficient. Table 4.16 and 4.17 highlight the top ten permissions, intents and hardware components highlight ranked with Mutual Information and Pearson Correlation Coefficient.

Table 4.15: Detection results considering all features for *DATASET-1* without applying the proposed approach

FEATURES used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
All Permissions	74.64	74.64	69.55	78.64	69.60	69.60	44.95	69.80	69.71
All Intents	67.18	67.18	55.11	67.18	54.28	55.15	50.25	64.26	50.26
All Hardware components	71.84	71.84	65.88	71.84	66.28	60.25	65.88	65.88	65.88

For the comparison, we ranked permissions, intents, and hardware components using Mu-

Table 4.16: Top 10 permissions, intents and hardware components ranked using Mutual Information

PERMISSIONS	INTENTS	HARDWARE COMPONENTS
MOUNT_UNMOUNT_FILESYSTEMS	USER_PRESENT	touchscreen
READ_PHONE_STATE	PACKAGE_REMOVED	touchscreen.multitouch.distinct
CHANGE_WIFI_STATE	DEFAULT	touchscreen.multitouch
GET_TASKS	PUSH_TIME	location
SYSTEM_ALERT_WINDOW	NOTIFICATION_RECEIVED_PROXY	location.network
READ_LOGS	REGISTER	location.GPS
WRITE_SETTINGS	PushService	screen.portrait
CHANGE_NETWORK_STATE	PACKAGE_ADDED	telephony
ACCESS_WIFI_STATE	REPORT	Camera.autofocus
ACCESS_COARSE_LOCATION	NOTIFICATION_OPENED	camera

Table 4.17: Top 10 permissions, intents, and hardware components ranked using Correlation Coefficient

PERMISSIONS	INTENTS	HARDWARE COMPONENTS
RECEIVE_SMS	MAIN	touchscreen
QUERY_ALL_PACKAGES	DEFAULT	touchscreen.multitouch
INTERNET	ACTION_SHUTDOWN	touchscreen.multitouch.distinct
CHANGE_BADGE	SEND	location
MAPS_RECEIVE	PHONE_STATE	location.network
SYSTEM_OVERLAY_WINDOW	CREATE_SHORTCUT	location.GPS
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	DOWNLOAD_COMPLETE	screen.portrait
RUN_INSTRUMENTATION	SCREEN_ON	Camera.autofocus
READ_EXTERNAL_STORAGE	LEANBACK_LAUNCHER	telephony
ACCESS	MEDIA_BUTTON	camera

tual Information and Pearson's Correlation Coefficient and further applied Algorithm 2 on *DATASET-1* to obtain their corresponding detection accuracies. First, we apply the proposed detection algorithm (Algorithm 2), only on permissions, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The proposed algorithm, i.e., Algorithm 2, will give the best set of permissions with higher accuracy as an output. The results are summarized in Table 4.18. From the table, we observe that we get the highest accuracy of 97.61% with only one permission, namely *MOUNT_UNMOUNT_FILESYSTEMS*, when we rank the permissions with Mutual Information. With Pearson's Correlation Coefficient, we get the highest accuracy of 96.02% again with only one permission, namely *RECEIVE_SMS*. With our proposed frequency-based *Chi-Square* test on permissions, we get the highest accuracy of 97.70% with ten permissions. Therefore, on *DATASET-1*, the frequency-based *Chi-Square* test is better than both Mutual Information and Pearson Correlation Coefficient when we rank permissions with these techniques. Moreover, as seen in Table 4.10, we get the highest accuracy of 98.49% from the proposed model with the frequency-based *Chi-Square* test on the combination of permissions and intents, which is higher than the accuracy obtained from Pearson Coefficient and Mutual Information. Hence, our model outperforms Mutual Information and Pearson Correlation Coefficient on permissions.

Table 4.18: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on permissions

Approach used	**Number of Permissions used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square* test (our approach)	10	97.70	97.70	74.54	97.70	92.16	97.70	97.58	97.58	97.58
Mutual Information [131]	01	97.43	97.43	95.67	97.43	25.6	97.43	97.61	97.61	97.61
Correlation Coefficient [131]	01	96.02	96.02	94.52	96.02	19.75	96.02	95.46	95.46	95.46

Next, we apply the proposed detection algorithm (Algorithm 2), only on intents, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The proposed algorithm, i.e., Algorithm 2, will give the best set of intents with higher accuracy as an output. The results are summarized in Table 4.19. From the table, we observe that we get the highest accuracy of 95.35% with only two intents, namely *USER_PRESENT* and *PACKAGE_REMOVED* when we rank the intents with Mutual Information. Whereas, with Pearson's Correlation Coefficient, we get the highest accuracy of 63.45% with only one intent, namely *Main*. With our proposed frequency-based *Chi-Square* test on intents, we get an accuracy of 95.35% with two intents, the same as that obtained from Mutual Information and better than that from the Pearson Coefficient.

Table 4.19: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on intents

Approach used	**Number of Intents used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square* (our approach)	02	95.35	95.35	65.32	95.35	95.35	95.35	95.30	95.30	95.30
Mutual Information [131]	02	95.35	95.35	95.13	95.35	95.35	95.35	96.12	96.12	96.12
Correlation Coefficient [131]	01	61.96	61.96	63.45	61.96	38.04	61.96	62.12	62.12	62.12

Next, we apply the proposed detection algorithm (Algorithm 2), only on hardware components, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The proposed algorithm, i.e., Algorithm 2, will give the best set of hardware components with higher accuracy as an output. The results are summarized in Table 4.20. From the table, we observe that we get the highest accuracy of 83.42% with six hardware components, namely { *touchscreen* , *touchscreen.multitouch.distinct* , *touchscreen.multitouch* , *location* , *location.network* and *location.GP* } when we rank the hardware components with Mutual Information. Whereas,

with Pearson's Correlation Coefficient also, we get the same highest accuracy of 83.42% with the same six hardware components. With our proposed frequency-based *Chi-Square* test on hardware components, we get an accuracy of 92.78% with 01 hardware components, better than that obtained from Mutual Information and Pearson Coefficient.

Table 4.20: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on hardware components

Approach used	**Number of Hardware Components used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square* (our approach)	01	92.78	92.78	92.78	92.78	92.78	92.78	92.78	90.75	92.78
Mutual Information [131]	06	83.42	83.42	83.21	83.42	83.42	83.42	83.42	83.14	83.21
Correlation Coefficient [131]	06	83.42	83.42	83.21	83.42	83.42	83.42	83.42	83.14	83.21

Moreover, as seen in Table 4.10, we get the highest accuracy of 98.49% from the proposed model with the frequency-based *Chi-Square* test on the combination of permissions and intents, which is higher than the accuracy obtained from Pearson Coefficient and Mutual Information applied on intents.

Now, we apply the proposed detection algorithm (Algorithm 2), on the combination of permissions and intents, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The proposed algorithm, i.e., Algorithm 2, will give the combined best set of permissions and intents with higher accuracy as an output. The results are summarized in Table 4.21. From the table, we observe that we get the highest accuracy of 96.82% with only one pair of permission and intent, namely *MOUNT_UNMOUNT_FILESYSTEMS and USER_PRESENT* when we rank permissions and intents with Mutual Information. Whereas, with Pearson's Correlation Coefficient, we get the highest accuracy of 63.86% again with only one pair, namely *RECEIVE_SMS and MAIN*. With our proposed frequency-based *Chi-Square* test on the combination of permissions and intents, we get the highest accuracy of 98.49% with six permissions and six intents. Hence, our model outperforms Mutual Information and the Pearson Correlation Coefficient on the combination of permissions and intents.

Now, we apply the proposed detection algorithm (Algorithm 2), on the combination of intents and hardware components, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The results are summarized in Table 4.22. From the table, we observe that we get the highest accuracy of 92.82% with two pairs of intents and hard-

Table 4.21: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of permission and intents

Approach used	**Number of Permission-Intent pairs used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square*	06	98.49	98.49	86.84	98.49	98.49	98.49	98.35	98.35	98.35
Mutual Information [131]	01	96.82	96.82	96.59	96.82	96.82	96.82	96.64	96.64	96.64
Correlation Coefficient [131]	01	63.86	63.86	39.88	63.86	39.88	63.86	63.55	63.55	63.55

ware components, namely *USER_PRESENT and touchscreen* , *PACKAGE_REMOVED and touchscreen.multitouch.distinct* when we rank intents and hardware components with Mutual Information. Whereas, with Pearson's Correlation Coefficient, we get the highest accuracy of 62.64% again with only one pair, namely *MAIN* and *touchscreen*. With our proposed frequency-based *Chi-Square* test on the combination of intents and hardware components, we get the highest accuracy of 97.43% with three intents and three hardware components. Hence, our model outperforms Mutual Information and the Pearson Correlation Coefficient on the combination of intents and hardware components.

Table 4.22: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of intents and hardware components

Approach used	**Number of Intent - Hardware component pairs used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square*	03	97.43	97.43	97.43	97.43	97.43	97.43	97.43	97.43	97.43
Mutual Information [131]	02	92.82	92.82	92.81	92.82	92.82	92.27	92.82	92.25	92.81
Correlation Coefficient [131]	01	62.64	62.64	62.62	62.64	62.62	62.62	62.62	62.21	62.61

Now, we apply the proposed detection algorithm (Algorithm 2), on the combination of permissions and hardware components, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The results are summarized in Table 4.23. From the table, we observe that we get the highest accuracy of 92.96% with only one pair of permission and hardware component, namely *MOUNT_UNMOUNT_FILESYSTEMS and touchscreen* when we rank permissions and intents with Mutual Information. Whereas, with Pearson's Correlation Coefficient, we get the highest accuracy of 86.19% again with only one pair, namely *RECEIVE_SMS and touchscreen*. With our proposed frequency-based *Chi-Square* test on the combination of permissions and hardware components, we get the highest accuracy of 95.41% with three permissions and three hardware components. Hence, our model outperforms Mutual Information

and the Pearson Correlation Coefficient on the combination of permissions and intents.

Table 4.23: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of permissions and hardware components

Approach used	**Number of Permission- Hardware Component pairs used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square*	03	95.41	95.41	95.41	95.41	95.41	90.05	95.41	90.05	95.41
Mutual Information [131]	01	92.96	92.96	92.96	92.96	92.96	92.96	92.96	92.96	92.96
Correlation Coefficient [131]	01	86.19	86.19	83.91	86.19	86.19	86.19	86.19	84.25	83.91

Now, we apply the proposed detection algorithm (Algorithm 2), on the combination of permissions and hardware components, after ranking them using Mutual Information and Pearson's Correlation Coefficient. The results are summarized in Table 4.24. From the table, we observe that we get the highest accuracy of 94.25% with only one trio of permission, intents and hardware component, namely { *MOUNT_UNMOUNT_FILESYSTEMS, USER_PRESENT and touchscreen* } when we rank permissions, intents and hardware components with Mutual Information. Whereas, with Pearson's Correlation Coefficient, we get the highest accuracy of 63.99% again with only one trio, namely { *RECEIVE_SMS, MAIN and touchscreen* }. With our proposed frequency-based *Chi-Square* test on the combination of permissions, intents and hardware components, we get the highest accuracy of 96.99% with three permissions, three intents, and three hardware components. Hence, our model outperforms Mutual Information and the Pearson Correlation Coefficient on the combination of permissions, intents and hardware components.

Table 4.24: Comparison of frequency-based *Chi-Square* test with Mutual Information and Pearson Coefficient on the combination of permissions, intents and hardware components

Approach used	**Number of Permission- Intents - Hardware Component trios used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
Frequency-based *Chi-Square*	03	96.99	96.99	96.99	96.99	91.34	72.34	96.99	70.34	96.99
Mutual Information [131]	01	94.25	94.25	94.24	94.25	88.91	94.24	94.24	88.91	94.24
Correlation Coefficient [131]	01	63.99	63.99	63.99	63.99	63.97	63.97	63.99	63.97	63.99

4.3.4 Detection Results on *DATASET-2*

The applications in the *DATASET-1* were collected over the period from 2016 to 2022. In this subsection, we discuss the results obtained from testing our proposed approach over a new and more recent dataset, i.e., on malicious applications downloaded from Androzoo that were detected in 2021 and 2022. Again, we perform seven experiments, considering 1) permissions alone, 2) intents alone, 3) hardware components alone, 4) permissions and intents combined, 5) intents and hardware components, 6) permissions and hardware components, and 7) permissions, intents and hardware components, all three of them combined.

Detection with Permissions alone

First, we apply the proposed detection algorithm (Algorithm 2) with permissions alone. Table 4.25 summarizes the detection results when we use permissions alone for detection on the recent dataset. With the top-ranked permission, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE*, we get the highest accuracy of 96.84% accuracy with one of the classifier. Then we move to the next iteration when considering the top two ranked permissions, i.e., combining *BIND_GET_INSTALL_REFERRER_SERVICE* with *JPUSH_MESSAGE* for detection. In this iteration, we get an increased accuracy of 97.02 %. Next, we consider the top three permissions and repeat the entire procedure. The procedure terminates until we observe a potential decrease in the detection accuracy. As shown in Table 4.25, we achieved the highest detection accuracy on the third iteration, i.e., upon adding the top three permissions, namely { *BIND_GET_INSTALL_REFERRER_SERVICE* , *JPUSH_MESSAGE* and *RESTART_ PACKAGES* }, we get the highest accuracy of 97.13%. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we observe that we get the highest accuracy of 97.13% when we apply the proposed Algorithm 2 on the recent dataset with permissions alone.

Table 4.25: Detection results with proposed approach considering only permissions

PERMISSIONS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_ REFERRER_ SERVICE	96.06	96.06	96.84	96.06	96.06	96.06	96.66	96.66	96.65
JPUSH_MESSAGE	97.02	97.02	60.93	97.02	97.02	97.02	96.65	96.65	96.65
RESTART_PACKAGES	**97.13**	**97.13**	53.28	**97.13**	**97.13**	**97.13**	96.51	96.51	96.51
SEND_SMS	94.50	94.50	52.10	94.50	94.50	94.50	95.76	95.76	95.77
RECEIVE_SMS	95.50	95.50	51.30	95.50	95.50	95.50	95.02	95.02	95.03
READ_SMS	92.85	92.85	50.90	92.85	92.85	92.85	94.60	94.60	94.62

Detection with Intents alone

Next, we apply the proposed approach to the recent dataset, with intents alone. Table 4.26 summarizes the detection results when we use intents alone for detection. With the top-ranked intent, i.e., *CONNECTION,* we get 96.03% accuracy. Then we move to the next iteration; when we consider the top two ranked intents, i.e., combining *CONNECTION* with *DaemonService* for detection and repeating the process, we get an accuracy of 96.26%. As shown in Table 4.26, we achieved the highest detection accuracy on the sixth iteration, i.e., upon adding the top six intents, namely { *CONNECTION* , *DaemonService* , *NOTIFICATION_RECEIVED* , *NOTIFICATION_ OPENED* , *MESSAGE RECEIVED,* and *START FROM AGOO* }, we get the highest accuracy of 97.89%. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we observe that we get the highest accuracy of 97.89% when we apply the proposed Algorithm 2 on the recent dataset with intents.

Table 4.26: Detection results with proposed approach considering only intents

Intents used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
CONNECTION	96.03	96.03	85.32	96.03	96.03	96.03	84.32	92.50	84.32
DaemonService	96.26	96.26	91.23	96.26	96.26	96.26	90.05	95.80	90.05
NOTIFICATION_RECEIVED	97.19	97.19	94.66	97.19	97.19	97.19	94.76	96.50	94.76
NOTIFICATION_OPENED	97.42	97.42	95.30	97.42	97.42	97.42	95.22	96.46	95.22
MESSAGE_RECEIVED	97.55	97.55	95.44	97.55	97.55	97.55	95.40	95.40	95.40
START_FROM_AGOO	**97.89**	**97.89**	88.97	**97.89**	**97.89**	**97.89**	89.21	90.20	89.21
REPORT	97.40	97.40	84.14	97.51	97.40	97.40	84.04	84.04	84.04
COMMAND	97.27	97.27	81.22	97.29	97.27	97.26	80.45	80.45	80.45

Detection with Hardware components Alone

Next, we apply the proposed detection algorithm (Algorithm 2) with hardware components alone. Table 4.27 summarizes the detection results when we use hardware components alone for detection. With the top-ranked hardware component, i.e., *location*, we get 86.93% accuracy with all the classifiers. We call this the first iteration, then we move to the next iteration when we consider the top two ranked hardware components, i.e., combining *location* with *screen.portrait* for detection and repeating the above mentioned process. In this iteration, we note that we get an accuracy of 89.68% from all the machine learning classifiers. As shown in Table 4.27, we achieved the highest detection accuracy on the second iteration itself, i.e., upon adding the top two hardware components, namely *location* and *screen.portrait*. From the next iteration, we observe that the detection accuracy starts decreasing. Finally, we observe that we get the highest accuracy of 89.68% when we apply the proposed Algorithm 2 only on

hardware components.

Table 4.27: Detection results with proposed approach considering only hardware components

HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
location	86.93	86.93	86.93	86.93	80.89	86.93	86.93	81.69	86.93
screen.portrait	**89.68**	**89.68**	**89.68**	**89.68**	85.1	**89.68**	**89.68**	86.57	**89.68**
vulkan	89.68	89.68	89.68	89.68	89.68	89.68	89.68	89.68	89.68
location.GPS	89.4	89.4	89.4	89.4	89.4	89.4	89.4	89.4	89.4

Detection with Combination of Permissions and Intents

Further, we apply the proposed approach to the recent dataset with the combination of permissions and intents. Table 4.28 summarizes the detection results. With the top-ranked pair, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE* and *CONNECTION*, we get 96.19% accuracy. Then we move to the next iteration when considering the top two ranked pairs of permissions and intents, i.e., combining., *BIND_GET_INSTALL_REFERRER_SERVICE* and *CONNECTION* with *JPUSH_MESSAGE* and *DaemonService* and we get an increased accuracy of 98.42%. Next, we consider the top three pairs of permissions and intents and repeat the entire procedure. We achieved the highest detection accuracy on the third iteration, i.e., upon adding the top three permissions, namely { *BIND_GET_INSTALL_REFERRER_SERVICE* , *JPUSH_MESSAGE*and *RESTART_PACKAGES* }, and top three intents namely { *CONNECTION* , *DaemonService* and *NOTIFICATION_RECEIVED* }, we get the highest accuracy of 98.74%. Hence, we can conclude that the proposed approach in this work can detect recent malware samples with an efficient accuracy of 98.74%.

Table 4.28: Detection results with proposed approach considering the combination of permissions and intents

PERMISSIONS and INTENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_ REFERRER_SERVICE and CONNECTION	96.19	96.19	73.09	96.19	96.19	96.19	97.42	97.42	97.42
JPUSH_MESSAGE and DaemonService	97.35	97.35	72.78	97.35	97.35	97.35	98.42	98.42	98.42
RESTART_PACKAGES And NOTIFICATION_RECEIVED	**98.74**	**98.74**	67.42	**98.74**	89.37	**98.74**	97.60	97.60	97.60
SEND_SMS and NOTIFICATION_OPENED	98.17	98.17	70.14	98.17	87.24	98.17	97.26	97.26	97.26
RECEIVE_SMS and MESSAGE_RECEIVED	96.75	96.75	73.60	96.75	96.75	96.75	97.05	97.05	97.05

Detection with combination of Intents and Hardware components

Next, we apply the proposed detection algorithm (Algorithm 2) on the combination of intents and hardware components. Table 4.29 summarizes the detection results when we use intents and hardware components for detection. With the top-ranked pair, i.e., *CONNECTION* and *location*, we get 92.94% accuracy with several classifiers. In the second iteration, we get an accuracy of 95.96% from DNN classifier. Hence, next, we consider the top three pairs of intents and hardware components and repeat the entire procedure. Subsequently, as shown in Table 4.29, we achieved the highest detection accuracy on the fourth iteration, i.e., upon adding four intents, namely { *CONNECTION* , *DaemonService* , *NOTIFICATION_RECEIVED* and *NOTIFICATION_OPENED* } and four hardware components namely { *location* , *screen.portrait* , *vulkan* and *location.GPS* } ,we get the highest accuracy of 96.55%. From the next iteration, we observe that the detection accuracy starts decreasing. We observe that we get the highest accuracy of 96.55% when we apply the proposed Algorithm 2 on the set of 8 features that contains four intents and four hardware components.

Table 4.29: Detection results with proposed approach considering the combination of intents and hardware components

INTENTS and HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
CONNECTION and location	92.94	92.94	92.94	92.94	90.29	90.29	92.94	90.29	92.94
DaemonService and screen.portrait	92.94	92.94	62.01	92.94	90.29	90.29	92.94	90.29	95.96
NOTIFICATION_RECEIVED and vulkan	95.96	95.96	83.96	95.96	94.56	94.56	94.87	94.56	92.11
NOTIFICATION_OPENED and location.GPS	**96.55**	**96.55**	71.86	**96.55**	95.45	95.45	95.45	94.95	82.18
MESSAGE_RECEIVED and location.network	95.36	95.36	78.75	95.36	94.48	94.48	94.48	92.25	91.45
START_FROM_AGOO and touchscreen.multitouch.distinct	94.93	94.93	78.96	94.93	94.26	86.57	93.28	91.58	90.76

Detection with Combination of Permissions and Hardware components

Further, we apply the proposed detection algorithm (Algorithm 2) on the combination of permissions and hardware components. Table 4.30 summarizes the detection results when we use permissions and hardware components for detection. With the top-ranked pair, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE* and *location*, we get 94.06% accuracy with several classifiers. In the second iteration, we get an accuracy of 95.9% from several classifiers and as shown in Table 4.30, we achieved the highest detection accuracy on the second iteration itself, i.e., upon adding two permissions, namely *BIND_GET_INSTALL_REFERRER_SERVICE* and *JPUSH_MESSAGE* and two hardware components namely *location* and *screen.portrait*. From the

next iteration, we observe that the detection accuracy starts decreasing. We observe that we get the highest accuracy of 95.9% when we apply the proposed Algorithm 2 on the set of four features that contains two permissions and two hardware components.

Table 4.30: Detection results with proposed approach considering the combination of permissions and hardware components

PERMISSIONS and HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_REFERRER _SERVICE and location	94.06	94.06	94.06	94.06	94.06	94.06	94.06	94.06	94.06
JPUSH_MESSAGE and screen.portrait	**95.9**	**95.9**	67.56	**95.9**	**95.9**	67.56	95.04	67.56	92.94
RESTART_PACKAGES and vulkan	95.17	95.17	83.46	95.17	83.13	55.28	94.43	80.32	83.21
SEND_SMS and location.GPS	93.73	93.73	75.62	93.73	63.46	63.79	93.01	70.54	80.65

Detection with Combination of Permissions, Intents and Hardware components

Further, we apply the proposed detection algorithm (Algorithm 2) on the combination of permissions, intents, and hardware components. Table 4.31 summarizes the detection results when we use permissions, intents, and hardware components for detection. With the top-ranked trio, i.e., *BIND_GET_INSTALL_REFERRER_SERVICE* , *CONNECTION* and *location*, we get 95.16% accuracy with several classifiers. We call this the first iteration and then move to the next iteration when we consider the top two ranked trio of permissions, intents, and hardware components, i.e., combining *BIND_GET_INSTALL_REFERRER_ SERVICE* , *CONNECTION* and *location* with *JPUSH_MESSAGE* , *DaemonService* and *screen.portrait* for detection and repeat the process mentioned above. In this iteration, we get an accuracy of 96.96% from several classifiers. As shown in Table 4.31, we achieved the highest detection accuracy on the second iteration, i.e., upon adding two permissions, namely *BIND_GET_INSTALL_REFERRER_SERVICE* and *JPUSH_MESSAGE*, two intents namely *CONNECTION* and *DaemonService* and two hardware components namely, *location* and *screen.portrait*, we get the highest accuracy of 96.96%. From the next iteration, we observe that the detection accuracy starts decreasing.

On comparing the highest accuracies obtained with permissions alone (97.40%), intents alone (95.78%), hardware components alone (89.68%), combination of intents and hardware components (96.55%), combination of permissions and hardware components (95.9%), combination of permissions, intents and hardware components (96.96%) we find that the combination of permissions and intents (98.18%) gives us better detection accuracy as compared to all three features when used alone or in other possible combinations.

Table 4.31: Detection results with proposed approach considering the combination of permissions, intents and hardware components

PERMISSIONS, INTENTS and HARDWARE COMPONENTS used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**ANN**	**BC**	**NB**	**LR**	**MLP**	**SVM**	**DNN**
BIND_GET_INSTALL_REFERRER _SERVICE, CONNECTION and location	95.16	95.16	76.62	95.16	76.62	76.62	94.1	76.62	80.15
JPUSH_MESSAGE, DaemonService and screen.portrait	**96.96**	**96.96**	**96.96**	86.27	**96.96**	72.88	96.34	85.91	91.17
RESTART_PACKAGES, NOTIFICATION_RECEIVED and vulkan	96.53	96.53	85.58	96.53	96.03	68.89	96.03	80.24	86.35
SEND_SMS, NOTIFICATION_OPENED and location.GPS	96.55	96.55	87.96	95.55	73.19	65.56	95.06	75.86	92.11

4.3.5 Comparison with other related works

In this section, we compare the performance of our proposed model with other similar works of Android malware detection that have used permissions or intents as features. Table 4.32 summarizes this comparison. As seen from the table, our work outperforms all these works in terms of detection accuracy. Some works have ranked the permissions based on frequency or with tests like Mutual Information and Pearson Correlation Coefficient. Some other works have applied feature selection techniques with Linear Regressions or Naive Bayes, whereas some authors have used permissions in pairs for Android malware detection. Only two works, i.e., Li et al. [16] and Wang et al. [131], have used a larger number of normal applications in their analysis than ours. However, their dataset size for malware apps is smaller than ours. Moreover, our work outperforms them in terms of detection accuracy. Hence, our proposed model is better than many state-of-the-art techniques presented in the literature for Android malware detection.

4.3.6 Limitations

Now, we describe a few limitations of the proposed approach. The proposed model ranks permissions and intents for malware detection, and hence, the model is a static detection. Static techniques are generally inexpensive in terms of complexity compared to dynamic approaches, as static features can be more easily extracted than dynamic ones. However, static methods have a few disadvantages, such as their inability to recognize stealthier behavior of code obfuscation and dynamic code loading. As a result, some malicious apps may incorporate such stealthy behavior and evade detection by the proposed static model. Hence, we will integrate

Table 4.32: Comparison of proposed work with the existing literature based on malware detection using permissions and intents.

Related Work	**Feature selection/Feature ranking technique used**	**Dataset Size**		**Detection accuracy (in %)**	**Number of best features**
		Normal	**Malware**		
Li et al. [16]	Permissions ranking based on frequency	310,926	62,838	93.62	22 permissions
Khariwal et al. [139]	Raked features using Information gain	1,414	1,714	94.73	37 features
Wang et al. [131]	Permissions ranking with Mutual Information, Correlation Coefficient and T-test	310,926	4,868	94.62	40 permissions
Yerima et al. [40]	Mutual Information gain based permissions and code based features	1,000	1,000	97.7	15 permissions
Chaudhary and Masood [43]	*Chi-Square* as a feature reduction technique	5065	426	96.4	-
Mahindru and Sangal [56]	Feature selection using *Chi-Square*, Gain Ratio, Filtered Subset selection, Information feature, LR analysis, PCA	5,00,000		98.2	
Şahin et al. [153]	Feature selection with Linear regression	1,000	1,000	96.1	27 permissions
Talha et al. [18]	Risk score calculated for each app	1,853	6,909	88.28	-
Doğru and Önder [20]	Permission groups score calculated, to sum up, app's risk Score	5,554	5,554	96.19	-
Shang et al. [21]	Naive Bayes and Pearson Correlation Coefficient	945	1,725	86.54	-
Tchakounté et al. [22]	Sequence alignment based similarity score	534	534	79.58	-
Kato et al. [39]	Similarity score between malware and normal permission pairs	11,500	19,000	97.3	-
Arora et al. [138]	Normal and malicious graphs of permission pairs	7,533	7,533	95.44	-
***PHIAnalyzer* (Proposed Model)**	**Permissions, Intents, and Hardware components ranking with Frequency-based *Chi-Square***	**77,000**	**77,000**	**98.49**	**12 features**

dynamic analysis with static features to detect stealthy malware samples in our future work. Moreover, some mobile attacks can be because of colluding apps, i.e., malicious behavior is distributed across several apps rather than one. However, the proposed model, in its current form, does not target colluding apps. Therefore, to further enhance the detection capability of the proposed model, we aim to target colluding apps in our future work.

4.4 Conclusion and Future Work

In this work, we proposed a novel static technique to detect Android malware using seven possible combinations of ranked permissions, intents, and hardware components. Initially, we ranked the permissions, intents, and hardware components separately based on their frequency difference in normal and malware datasets. Subsequently, we ranked the features using

a frequency-based statistical *Chi-Square* test. Finally, we proposed a novel algorithm with machine learning and deep learning techniques to merge the three ranked lists and find the best subset of features. Our experimental results demonstrate that the proposed model gives adequate detection accuracy of 98.49% with 12 features, i.e., the top six permissions combined with the top six intents. Furthermore, results showed that our proposed method is better than many state-of-the-art techniques for Android malware detection in terms of detection accuracy and the number of features used. In our future work, we will expand the analysis on other manifest file components such as broadcast receivers, activities, services, etc. We will also aim to integrate dynamic analysis to detect stealthier malware and colluding apps.

Chapter 5

CorrNetDroid: Android Malware Detector leveraging a Correlation-based Feature Selection for Network Traffic features

---

In this chapter, we present *CorrNetDroid*, a model designed to detect Android malware by analyzing network traffic from malicious and benign apps. Numerous traffic features can be extracted from captured data; however, using all features can reduce detection accuracy. Effective feature ranking is essential to address this, while avoiding redundancy among variables. This work manages both feature–class and feature–feature correlations to eliminate redundancy among ranked features. In Section 5.1, we outline the motivation and provide an overview of the proposed methodology. Section 5.2 details the methodology, while Section 5.3 presents the results and their analysis. Limitations are discussed in Section 5.4, and Section 5.5 concludes with future research directions.

5.1 Introduction

Over the years, static analysis has proved to be quite efficient in terms of extraction of features, cost, and detection accuracy. Investigating malware without executing the actual code but by collecting basic information about an app's functionality seemed to be profitable.

Hence, our previous works were primarily based on static analysis ([154], [155]). In static analysis, all the static features of an application can be scrutinized, such as the permissions, intents, hardware features, or other *AndroidManifest* file components. However, some stealthier malware functions without showing malicious behavior, i.e., they might even evade static detection techniques such as permission check systems. Although the results might come faster using static analysis, they are found to be incapable of detecting malware using advanced techniques such as code obfuscation, polymorphism, and encryption. Dynamic analysis, in which applications are monitored at runtime by actually executing their code, overcomes the shortcomings of static analysis. Therefore, in this chapter, we focus on creating a system-level lightweight malware identification framework using dynamic analysis.

Some malware are more dangerous than others because they connect to a remote server in the background to obtain commands or to leak/send private information of users or the device itself to the server. Hence, our work in this chapter aims to detect malware that connects to a server in the background without the user's knowledge. Because of this behavior, they produce network traffic. Keeping the seriousness of detection in mind, this work analyzed their network traffic behavior and compared it with normal mobile traffic to determine the deviations in the behavior of malware. The aim of the proposed work in this chapter is to detect Android malware remotely controlled by a server and obtain commands from that server or leak private user information. There are some other malware that initially do not contain any malicious code, but after installation on the device, they inject malicious code during the update. Both of these categories of malware have one thing in common: they connect to a network. Around 93% of Android malware samples have network connectivity. Because these types of malware are controlled by a remote server, they convert the mobile device into a mobile bot, which can pose a serious threat to the user community.

In the context of Android malware detection using network traffic , the most commonly used dynamic features are HTTP request headers, UDP flows, and TCP flows, which refer to the network communication patterns associated with different types of network protocols. Analyzing these flows can be crucial for identifying potential malicious behavior or communication patterns indicative of malware. Among these, we choose and handle TCP connections as our preferred network traffic feature for the proposed work. TCP (Transmission Control Protocol) is a connection-oriented protocol that ensures reliable and ordered delivery of data between devices. Malicious activities often involve the establishment of more persistent and reliable connections, which makes TCP a common choice for malware communication. Mon-

Table 5.1: Some traffic features and their range for malware and normal mobile traffic on smartphones

Feature Name	Range in normal traffic	Range in malware traffic
Flow_duration	0 - 49560.3403	0 - 52122.128886
Packets_sent_per_flow	1 - 110985	1 - 48265
Packets_received_per_flow	0 - 215014	0 - 104312
Bytes_sent	40 - 82771640	54 - 16288912
Bytes_received	0 - 1217408141	0 - 157283100

itoring TCP flows can help detect unusual connection patterns, such as connections to known malicious servers, multiple connections to different servers, or connections to non-standard ports.

Some dynamic Android malware detection techniques have been proposed in the literature using features such as cryptographic and network operation [81]. However, we chose network traffic flows as the preferred feature to perform dynamic analysis because they provide a comprehensive and real-time view of an application's external communication, making it particularly effective in the context of Android malware detection.

Motivation: Analyzing network traffic usage patterns is an effective way to detect the presence of malware. Therefore, network traffic flows have been widely used in the literature for Android malware detection. However, similar to permissions or intents, there are many similarities between the network traffic feature patterns of normal and malicious apps. Tables 5.1 summarize the network traffic ranges of several features extracted from a binary class dataset, i.e., benign and malware types. We extracted over 9 lakh network traffic flows of the normal class and an equal number for the malware class by combining the datasets from various repositories. More details about the dataset are provided in upcoming sections. Furthermore, we developed 16 network traffic features from the network traffic data. As seen in Table 5.1, commonly used features, namely *Flow_Duration*, *Packets_sent/received*, *Bytes_sent/received* from the source to destination or vise versa, are present in significantly overlapping ranges when observed in both normal and malware traffic data.

Such similarity in these features across both datasets motivates us to rank the features to propose an efficient detection model with distinguishing features. Moreover, experimental results indicate, as depicted in the upcoming sections , that if we use all of the available features as input for malware detection, irrelevant features will hamper detection accuracy. Hence, feature reduction is a key process in developing a detection algorithm. More importantly, the field of Android security revolves around accuracy; the better the accuracy of detecting malware, the better the detection system, and the best accuracy can only be obtained by using the best set of

features. Hence, feature ranking and selection are the key aspects of our research.

Drawbacks of existing approaches

Several related studies have used the dynamic features of TCP flows or HTTP packets as their base for the detection of Android malware. For instance, Wang et al. [104] analyzed multiple levels of network traffic features and emphasized that combining 2 levels, namely HTTP packet and TCP flow, can successfully lead to the creation of a lightweight server-based malware detection model. Lastly, on the belief that machine learning can be used to automatically discover the rules by analyzing the data, they applied machine learning algorithms on the training set and performed the testing experiments. However, they did not use the key concept of ranking the features and hence missed the feature reduction step, which could have enhanced the quality of their results.

In several other related works, such as [92] and [95], the authors built a detection system using the best subset of features by ranking or selection. More specifically, Arora and Peddoju [92] were able to obtain 22 network traffic features. Consequently, they aimed to reduce the feature set and used information gain and statistical techniques such as chi-square to rank the feature set. The results indicated that their approach was successful in reducing the training and testing time while keeping the detection accuracy at maximum. Shabtai et al. [95] tried to understand the reason behind the deviations in the application's network traffic behavior from the normal flow by observing the network traffic flows. Hence, they computed the probability scores depicting the deviation of features' behaviors from normal traffic patterns and later used the threshold approach to select the best subset of features. However, both studies were implemented on a smaller set of network traffic flows compared with the huge dataset in our proposed study. More importantly, our work outperforms both in terms of detection accuracy.

Objective of our Proposed Approach

We aim to build a robust and efficient dynamic analysis-based Android Malware detection system that is capable of identifying malicious behavior of applications on Android smartphones. At the same time, we are driven to fulfill this objective using only the least and the best features, aggregated using the TCP flows of the application's network data. Feature ranking methods are mainly criticized for their poor handling of redundant variables. Hence, in this work, we attempt to handle both feature– class and feature– feature correlations by ranking the features first using the statistical measure *crRelevance* and then further deploying another statistical technique called *Normalized Mean Residue Similarity (NMRS)* in our proposed approach to

remove the redundancy between the ranked features. The following research questions emerge in the light of proposing a dynamic detection model based on the ranking of network traffic features:

- **RQ1** Where does the need for ranking network traffic features arise and subsequently, what is the significance of feature reduction compared with feeding all the features as inputs at once?

- **RQ2** How to incorporate feature ranking while eliminating redundant features, i.e., how to rank network traffic features and select the least correlated subset?

- **RQ3** How to frame a detection approach while considering both feature– class and feature– feature correlations?

We are motivated to answer these questions with the vision to develop an Android malware detector, named *CorrNetDroid*, based on dynamic analysis using TCP flows via developed features. We ranked the features using a statistical technique called *crRelevance* to find the best feature– class correlated subset, i.e., features having the optimum ability to distinguish between normal and malware class labels. We used *crRelevance* because of its simple working and most suitable logic. The functioning of *crRelevance* prioritizes the feature showing steady network traffic patterns for prolonged durations, be it the normal pattern or malware. Therefore, the feature showing the least deviation from one class pattern is placed at the top of the table. Moreover, unlike *crRelevance*, several other statistical tests have specific assumptions that need to be met for the results to be valid. For instance, ANOVA assumes normality and homogeneity of variances, and chi-square and Mann–Whitney tests assume independence of observations with mutually exclusive categories. Most such tests are also found to be sensitive to outliers. These limitations make *crRelevance* the most appropriate choice for our study.

Next, it was observed that features that are closely related to each other often produce similar results or have a similar impact on the output data. Hence, we incorporated the criteria of feature– feature correlation to select the best and negatively correlated features from the *crRelevance* rankings. To do this, we have chosen another statistical measure called *Normalized Mean Residue Similarity (NMRS)* because of its simple yet effective working compared with other statistical measures such as Pearson's correlation coefficient or Spearman correlation coefficient, which often lead to the inclusion of some undesired features. After finding

the feature– feature correlation for all possible pairs, we proposed a novel *NMRS*-based detection algorithm that uses *crRelevance*-ranked features and applies various machine learning and deep learning techniques to effectively detect Android malware. Our detection results are better than many state-of-the-art techniques proposed in the existing literature. Moreover, our experiments demonstrate that the proposed *NMRS*-based detection algorithm with *crRelevance* rankings gives us better accuracy than other statistical tests such as chi-square, ANOVA, Mann– Whitney U test, Kruskal–Wallis test, and Pearson correlation coefficient. In addition, our proposed work outperforms other similar works on Android malware detection, which we evaluate against the same dataset of normal and malicious apps.

Contributions: The main contributions of this chapter are highlighted below:

- First, we ranked the TCP flow-based network traffic features according to their ability to distinguish between normal and malicious class labels using the statistical measure *crRelevance*.
- Next, we applied another statistical technique known as *NMRS* to analyze the feature–feature correlation.
- We proposed a novel correlation-based feature selection algorithm to deploy *NMRS* on the ranking given by *crRelevance* to filter out only the best as well as a redundancy-free subset of network traffic features optimal for building an Android malware detector.
- We observed that the detection results of the proposed approach are relatively better than various statistical and state-of-the-art techniques existing in the literature for Android malware detection.

5.2 System Design

In this section, we explain our proposed methodology in detail. Figure 5.1 summarizes a brief yet complete idea of our proposed model *CorrNetDroid*, which is divided mainly into two modules. In the first module, named as *Selection Module*, we compute the network traffic features from the training dataset's TCP flows and rank them using a statistical measure called *crRelevance*. Such a ranking will help us grade them according to their ability to distinguish between the two class labels, normal and malware. Simultaneously, we also rank the features on the basis of their inter-correlation score using another statistical technique called *NMRS*, which would help us eliminate the redundancy between the ranked features. In the *Detection Module*,

we propose a novel *NMRS*-based algorithm that applies machine learning and deep learning techniques to *crRelevance* rankings to obtain the best subset of features that can provide higher detection accuracy.

The following subsections discuss in detail both modules of the proposed model.

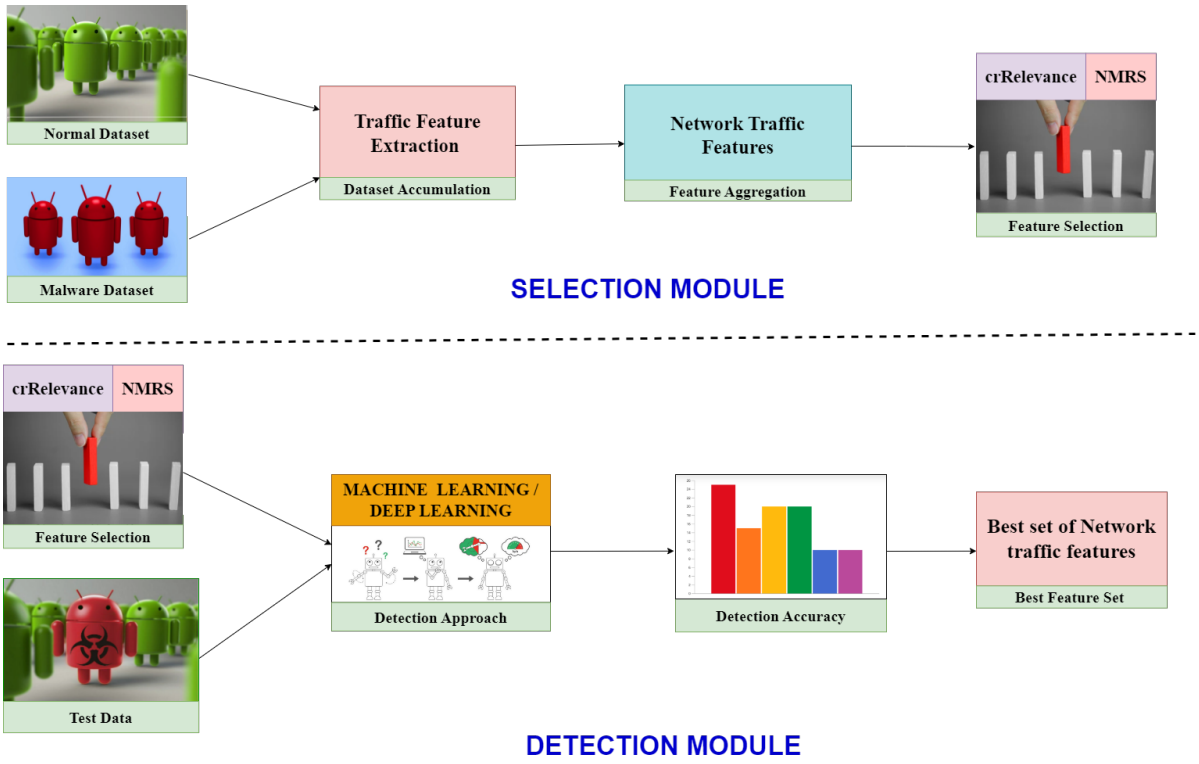


Figure 5.1: *CorrNetDroid* System Design

SELECTION MODULE

5.2.1 Dataset Collection

To begin with, we needed a vast dataset of mobile network traffic generated by both normal and malware applications to conduct our research. For this purpose, we rely on the well-defined *in-the-wild* type datasets provided by the Canadian Institute for Cybersecurity (CIC). The network traffic data used for training and testing were acquired from four datasets, namely CICAndMal2017 [156], CIC-InvesAndMal2019 [157], CIC-AAGM2017 [158] and USTC-TFC2016 [159]. Combining these datasets, we managed to gather 9,88,280 network traffic flows for normal and malware, which were further divided into training and testing sets. These datasets include a variety of both static and dynamic features extracted from normal and malware applications; however, for our work, we keep our focus limited to mobile network traffic already extracted and well-labeled in the form of pcap files. Moreover, among the com-

monly used features for malware traffic detection, such as the HTTP protocol and TCP and UDP flows, we choose and handle the TCP connection as the main interaction between the applications and the network.

5.2.2 Traffic Split

In mobile network traffic, Transmission Control Protocol (TCP) flows play a crucial role in ensuring reliable and efficient communication between devices. TCP is a connection-oriented protocol that facilitates the orderly and error-checked delivery of data between applications. TCP connections begin with a three-way handshake, where the sender and receiver exchange synchronization (SYN) and acknowledgment (ACK) packets to establish a connection. TCP connections are gracefully terminated using a four-way handshake, involving FIN (finish) and ACK flags.

Using TCP flows offers several benefits, particularly in the context of network communication. TCP includes mechanisms for error detection and correction and establishes a connection before data transmission begins, ensuring a reliable and orderly communication channel. Moreover, TCP is widely supported across different operating systems and network environments. Its ubiquity makes it suitable for diverse applications and scenarios. Since the network traffic data used in our study is already in the form of pcap files, Wireshark is used first to filter the pcap, and then the files are separated into the basic flow.

5.2.3 Features Aggregation

After data collection, we extracted several network traffic features from the normal and malware traffic. Table 5.2 summarizes the 16 traffic features used in our experiments. All such features are flow-based and can be easily extracted from the pcap flows of normal and malware traffic. For each feature, we have also written its short notation, for instance, *F1* for *Average_packet_size*, *F4* for *Flow_duration*, etc. In the Results section, we denote the features by their notations for easy understanding and interpretation of the results.

5.2.4 Feature Selection

Many existing techniques for feature selection often overlook the correlation among features. However, eliminating redundant features from a dataset can reduce the time required by the

Table 5.2: List of network traffic features

Average_packet_size **(F1)**	Packets_sent_per_second **(F9)**
Time_interval_between_packets_ sent **(F2)**	Packets_received_per_ second **(F10)**
Time_interval_between_packets_ received **(F3)**	Packets_received_per_flow **(F11)**
Flow_duration **(F4)**	Packet_size_received **(F12)**
Ratio_of_incoming_to_outgoing_ packets **(F5)**	Bytes_sent **(F13)**
Ratio_of_incoming_to_outgoing_ bytes **(F6)**	Bytes_sent_per_second **(F14)**
Packet_size_sent **(F7)**	Bytes_received **(F15)**
Packets_sent_per_flow **(F8)**	Bytes_received_per_ second **(F16)**

inducer module during classification. The removal of redundant or irrelevant features not only contributes to a significant improvement in the time efficiency of a machine learning technique but also aids in creating an optimal feature subset for a dataset.

When selecting an optimal feature subset, it is crucial to consider both feature– feature and feature– class correlations. An ideal feature subset should consist of features highly correlated with class labels while exhibiting minimal correlation with each other [160]. In contrast to many existing methods, the proposed feature selection technique addresses both feature– feature and feature–class correlations. It employs two statistical measures, namely *crRelevance* and *NMRS*, to achieve this goal. In the following subsections, we discuss the working of *crRelevance* and *NMRS* measures, which play a crucial role in the proposed feature selection method. These measures are used to quantify feature–class and feature– feature correlation.

crRelevance: Feature-Class Correlation

crRelevance [161] is a measure used to evaluate the ability of a feature to distinguish between various class labels. It produces a value in the range $[0,1]$. The following definitions provide the theoretical basis for *crRelevance*.

Definition 1 For a feature f_k with values $\{x_1, x_2 \ldots x_n\}$ corresponding to n objects or instances in the dataset, a class range can be defined as a range R $= [r_1, r_2]$ such that, $\forall x_i, x_j, r_1 \leq x_i \leq r_2, r_1 \leq x_j \leq r_2$ and $\text{class}_{f_k}(x_i) = \text{class}_{f_k}(x_j)$, where $\text{class}_f(x)$ is the class associated with value x over feature f. In other words, a class range R $= [r_1, r_2]$ over feature f is said to be associated with class A, if $\forall x, r_1 \leq x \leq r_2, \text{class}_f(x) = A$.

Definition 2 Cardinality of a class range R $= [r_1, r_2]$, denoted as rcard_f(R) is defined as the cardinality of the set $\{x \mid x \in V, r_1 \leq x \leq r_2\}$, where V is the set of values of all the objects for feature f.

Definition 3 Class-cardinality of a class A, ccard(A), can be defined as the cardinality of the set $\{x \mid \text{class}(x) = A\}$.

Definition 4 Core class range of class A denoted as ccrange(A), can be defined as the highest class range associated with the class A in terms of class range cardinality. A range R_i associated with class A is called the core class range of A if there is no R_j in F, such that $\text{rcard}(R_j) > \text{rcard}(R_i)$.

Now, core range-based relevance of a class A for feature f_i, denoted by $crRelevance_{\text{f}_\text{i}}^{\text{class}}(\text{A})$, is defined as follows.

$$crRelevance_{\text{f}_\text{i}}^{\text{class}}(\text{A}) = \frac{\text{rcard}(\text{ccrange}(\text{A}))}{\text{ccard}(\text{A})} \tag{5.2.1}$$

For a 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, denoted by *crRelevance* (f_i), for a dataset with n classes $A_1, A_2, \ldots, A_n$, can be defined as follows.

$$crRelevance(f_i) = \max_{1 \le j \le n} crRelevance_{f_i}^{\text{class}}(A_j) \tag{5.2.2}$$

In our case, we had two numerical-type continuous datasets comprising 16 features for two classes, namely normal and malware. Each of them consisted of 6,94, 261 flows or instances extracted from the TCP flows training dataset. Upon applying the above definitions to both the training datasets individually, we managed to obtain the *crRelevance*_normal and *crRelevance*_malware scores for each feature. To place the most distinguishing features at the top of the ranked list irrespective of their preference for either class, we compute the absolute difference between the *crRelevance*_normal and *crRelevance*_malware scores for each feature. We would like to point out that the absolute difference was taken instead of just the difference to avoid the accumulation of the best features on the endpoints of the ranked list rather than on the top. These scores were further used to rank the features such that the feature with the highest *crRelevance* score is the top-ranked feature and hence holds the best ability to distinguish between class labels.

NMRS- Feature-Feature Correlation

The above-mentioned technique ranks features according to their relevance to the output variable. However, it does not consider redundant features. It has been observed that features that are closely related to each other often produce similar results or have a similar impact on the output data. Hence, to determine whether two features have similar patterns, an appropriate similarity measure must be chosen. In other words, to find the correlation between features, we use an effective method called *Normalized Mean Residue Similarity (NMRS)* [162].

The level of concordance between feature $d_1 = (a_1, a_2, \ldots, a_n)$ with respect to another feature $d_2 = (b_1, b_2, \ldots, b_n)$ is defined by the following formula :

$$= 1 - \frac{\sum_{i=1}^{n} |a_i - a_{\text{mean}} - b_i + b_{\text{mean}}|}{2 \times \max\{\sum_{i=1}^{n} |(a_i - a_{\text{mean}})|, \sum_{i=1}^{n} |(b_i - b_{\text{mean}})|\}} \tag{5.2.3}$$

where a_{mean} is the mean of all the elements of feature d_1;

$$a_{\text{mean}} = \{a_1 + a_2 + \ldots + a_n\}/n,$$

and b_{mean} is the mean of all the elements of feature d_2 ;

$$b_{\text{mean}} = \{b_1 + b_2 + \ldots + b_n\}/n.$$

In our proposed work, because we wish to find the feature– feature correlation between all possible pairs without any relation to their classes, we concatenate the normal and malware traffic feature values, i.e., we combine the flows of both class labels for each feature and then compute the *NMRS* scores. In this way, we iteratively compute the *NMRS* score for each feature pair by taking them as d_1 and d_2 variables in the given *NMRS* formula. The most widely used similarity/correlation measures in the field of Android malware are the Pearson correlation coefficient, Spearman correlation coefficient, and mean squared residue. However, all of these methods are lacking in some way or another. For instance, along with shifting patterns, the Pearson correlation coefficient also detects scaling and other patterns that are normally not desired and may lead to the inclusion of features that have a considerable amount of difference between their expression levels. The Spearman Rank correlation coefficient uses ranks to calculate correlation, which can not detect shifting patterns or scaling patterns. The mean squared residue is sufficient to detect shifting patterns, but the aggregate measure can not operate in a mutual mode, i.e., it can not find the correlation between a pair of features. Due to the above-mentioned reasons, we have chosen *NMRS* as our preferred choice for the feature–feature correlation extractor.

Algorithm 3 Proposed Malware Detection Algorithm

Input: $FC_{List} \leftarrow$ *crRelevance* ranked list, $FF_{List} \leftarrow$ *NMRS* ranked list
Output: Best set of features with higher detection rate
$BestFeatures \leftarrow$ Initialized with all 16 traffic features
$\rho(F_i) \leftarrow$ Ranking of a feature in FC_{List}
$TestData \leftarrow$ Concatenated 16 traffic features' normal and malware class values kept for testing.
$D_{Max} \leftarrow$ Maximum accuracy obtained, initialized to zero.
$D_{Acc} \leftarrow$ Accuracy obtained after each iteration.
while $FF_{List} \neq \phi$ **do** // While feature pairs exist in FF_{List}
  Select $(F_i\text{-}F_j) \in FF_{List}$
  if $\rho(F_i) < \rho(F_j)$ **then**
    $TestData = TestData \setminus F_i$ // Delete F_i from $TestData$
    Find D_{Acc} using ML algorithms for features present in $TestData$
    if $D_{Acc} > D_{Max}$ **then**
      $D_{Max} = D_{Acc}$
      $BestFeatures = BestFeatures \setminus F_i$ // Delete F_i from $BestFeatures$
      $FF_{List} = FF_{List} \setminus \{F_x\text{-}F_y : F_x \text{ or } F_y = F_i\}$ // Delete all pairs having instance of F_i from FF_{List}
    end if
  end if
end while
return $BestFeatures$
return D_{Max}

DETECTION MODULE

5.2.5 Proposed Detection Algorithm

This section presents our proposed detection algorithm, termed Algorithm 3, which answers research question three, i.e., how to frame a detection approach while considering both feature–class and feature– feature correlations.

The goal is to find features that are not strongly correlated with each other but have high predictive and class-distinguishing abilities compared with their peers in the entire ranked set to provide better detection accuracy. We aim to find the best set of features to provide better detection accuracy. The proposed method starts with the computation of *crRelevance* scores for each feature and pairwise correlation between features using *NMRS*. Once these scores are computed and sorted in descending order, the processing starts from the feature pair with the highest *NMRS* score. FC_{List} and FF_{List} represent the feature– class correlation rankings computed by *crRelevance* and the feature– feature correlation rankings computed by *NMRS*, respectively, in Algorithm 3.

In the first iteration, we selected the top-ranked feature pair from FF_{List}. Next, we compared the individual rankings of both features from the FC_{List} and eliminated the feature with a lower ranking in FC_{List} from the testing dataset, which was initialized to all 16 traffic feature values. We then execute the machine learning and deep learning classifiers on the testing data by considering only these 15 features, i.e., after eliminating the lower-ranked feature, and observe the detection accuracy, say D_{Acc}. The maximum accuracy, say D_{Max}, is initialized to zero. At

each iteration, we compare D_{Acc} and D_{Max}. If the accuracy at the current iteration, i.e., D_{Acc}, is higher than D_{Max}, we proceed to the next iteration and set D_{Max} as D_{Acc}. In addition, we eliminated the same lower-ranked feature from the set of Best Features, which was initialized to all 16 features, and all its instances from FF_{List} list as well.

In the next iteration, we again select the top-ranked pair from FF_{List}, compare the individual rankings of both features from the FC_{List} and find the detection accuracy on the testing data after eliminating the lower ranked feature, i.e., D_{Acc} for the current iteration considering 14 features out of 16. Again, we compare the D_{Max} and D_{Acc}, and if D_{Acc} is higher than D_{Max}, the value of D_{Max} gets updated to D_{Acc} . Similarly, we eliminate the lower-ranked feature from the set of Best Features as well as from FF_{List}. The algorithm continues the same way and terminates till there exists a single pair in the FF_{List}. Finally, we return the D_{Max} and the best set of features. The primary factor influencing this complexity is the time taken by the embedded machine-learning algorithm and the number of feature pairs in FF_{List}. If the total number of features is denoted by n, then the total number of feature pairs in FF_{List} comes out to be $\binom{n}{2}$. As a result, the computational complexity of the proposed algorithm becomes $O(n^2 \cdot M)$, where M is the time taken by the ML algorithms.

Using the *NMRS*-based proposed feature selection method that further uses *crRelevance* developed rankings as mentioned above, we answer research question two, i.e., how to rank the features to recognize the most distinguishing as well as least correlated ones among them.

5.2.6 Machine Learning and Deep Learning Classifiers

We have used several machine learning and deep learning classifiers [135] in our detection approach. We applied nine widely used techniques, namely Decision Trees (DT), Random Forest (RF), Bagging classifier (BC), Gaussian Naive Bayes (NB), Logistic Regression (LR), as machine learning classifiers and Multilayer Perceptron (MLP), Artificial Neural Networks (ANN), Dense Neural Network (DNN) and Convolutional Neural Network (CNN) as deep learning classifiers.

All experiments with these classifiers were performed using ten-fold cross-validation [136]. The code concludes by printing the cross-validation results, including the accuracy scores for each fold and the mean accuracy across all folds. This provides insights into the model's consistency and overall performance across diverse subsets of the dataset.

5.3 Results and Discussion

In this section, we present and discuss the experimental results obtained using the proposed *CorrNetDroid* model. We point out that we have separate datasets for training and testing. As described in Section 5.2.1, we accumulated 9,88,280 TCP flows, each from the normal and malware categories, by compiling four datasets from various repositories. Of these, we use 6,94,261 flows of each category for training. The remaining 2,94,019 flows were used for testing. We named this as *Testing Dataset*. In the upcoming subsections, we first discuss the ranking obtained from the two statistical measures used in our study, *crRelevance* and *NMRS*. Next, we describe the detection results on the *Testing Dataset*. Furthermore, we compared our proposed model with similar models for Android malware detection.

5.3.1 Features Ranking

To identify the features having the best ability to distinguish between class labels, we applied the statistical measure *crRelevance* on the normal and malware TCP-based features individually kept for training. As an output, the technique produces a pair of feature-class correlation scores for each feature. Furthermore, the sorted absolute differences between the *crRelevance*_normal and *crRelevance*_malware of features were used to give the ranking. Table 5.3 summarizes the network features ranked according to the difference between the category scores. Table 5.3 highlights that the feature *Packet_size_received (F12)* has the best ability to distinguish between normal and malware class labels and thus sits at the top of the table with the highest absolute difference between category scores. Similarly, it can be seen that the feature *Bytes_sent_per_second (F14)* has the worst absolute difference, and it can be inferred that it has the worst distinguishing characteristics trait.

As described in subsection 5.2.5, our proposed feature selection and detection algorithm uses both feature– class and feature– feature correlation, i.e., both *crRelevance* and *NMRS* rankings, to choose the best subset of features. Hence, in the next step, we applied *NMRS* to all possible feature pairs (120 to be precise) and ranked them in such a way that the pair that is found to be the most correlated resides at the top of the table. Table 5.4 summarizes 20 traffic feature pairs, from the top and bottom , ranked on the basis of their *NMRS* correlation scores. The table can be understood as follows: The features *Average_packet_size (F1)* and *Packet_size_received (F12)* are found to have the best correlation score and thus have the best correlation between them.

Table 5.3: Traffic features ranked using *crRelevance* and their correspoding difference between category scores

Traffic Feature name	***crRelevance* Normal score**	***crRelevance* Malware score**	**Absolute Difference**
Packet_size_received (F12)	0.26596232	0.012590384	0.253371936
Ratio_of_incoming_to_ outgoing_packets (F5)	0.296365915	0.130850114	0.165515801
Average_packet_size (F1)	0.12240083	0.009480598	0.112920232
Bytes_received (F15)	0.126547691	0.03779996	0.088747731
Packets_received_per_flow (F11)	0.111917725	0.05154409	0.060373635
Time_interval_between_ packets_received (F3)	0.036142079	0.084497162	0.048355083
Packet_size_sent (F7)	0.021238441	0.055827788	0.034589347
Packets_sent_per_flow (F8)	0.080901968	0.063568404	0.017333564
Bytes_sent (F13)	0.02805145	0.016737245	0.011314205
Packets_received_per_second (F10)	0.000396105	0.009657765	0.00926166
Bytes_received_per_second (F16)	0.001100452	0.010035145	0.008934693
Ratio_of_incoming_to_ outgoing_bytes (F6)	0.024832195	0.020339642	0.004492553
Packets_sent_per_second (F9)	0.000162763	0.001974764	0.001812001
Flow_Duration (F4)	0.000331288	0.001971884	0.001640596
Time_interval_between_ packets_sent (F2)	0.000162763	0.001748624	0.001585861
Bytes_sent_per_second (F14)	0.000193011	0.001627632	0.001434621

Similarly, the features *Time_interval_between_ packets_sent (F2)* and *Packets_received_per_second (F10)* have the worst correlation between them and are listed at the end of the table.

Table 5.4: 20 traffic feature pairs, top and bottom each, ranked using *NMRS* and their corresponding correlation scores

Feature pairs (Top 20)	***NMRS* Correlation scores**	**Feature pairs (Bottom 20)**	***NMRS* Correlation scores**
F1 - F12	0.759744419	F3 - F4	0.492246708
F8 - F11	0.757228415	F3 - F12	0.491129541
F9 - F10	0.688153793	F6 - F7	0.490277257
F6 - F8	0.552430678	F4 - F12	0.490246288
F3 - F14	0.54525236	F7 - F10	0.488841839
F6 - F11	0.537300131	F2 - F11	0.488644157
F2 - F4	0.534887165	F8 - F10	0.488640907
F1 - F7	0.534729436	F4 - F10	0.486058046
F15 - F16	0.527539689	F9 - F11	0.480918693
F14 - F16	0.520365838	F7 - F8	0.475437864
F13 - F14	0.517503603	F4 - F9	0.474087994
F3 - F16	0.516788742	F1 - F4	0.473606551
F10 - F11	0.516232597	F2 - F8	0.472189992
F1 - F8	0.515903675	F7 - F9	0.469414718
F4 - F8	0.514681773	F2 - F9	0.447610523
F1 - F11	0.513675549	F7 - F11	0.434838592
F5 - F6	0.513187313	F8 - F9	0.405520241
F6 - F10	0.509712683	F2 - F6	0.40035683
F11 - F12	0.509375491	F4 - F7	0.36976355
F1 - F6	0.509328369	F2 - F10	0.328580529

In the following subsection, we present the detection results obtained using the proposed model.

5.3.2 Detection Results on *Testing Dataset*

In this section, we discuss the detection results, i.e., the accuracy obtained from our proposed approach over the *Testing Dataset*. We apply the proposed *NMRS*-based algorithm to the crRelevance rankings. The algorithm provides the best set of negatively correlated features with higher detection accuracy as an output. Table 5.5 summarizes the detection results at each iteration. The table can be understood as follows: In the first iteration of eliminating the lower *crRelevance* ranked feature named *F1* out of the highest correlated pair, we observe that we obtain the highest detection accuracy of 81.67% with RF classifier, i.e., on considering 15 features now out of the total lot of 16, the highest detection accuracy of 81.67% can be achieved. We call this the first iteration and move on to the next iteration, where we eliminate the lower *crRelevance* ranked feature from the second highest correlated pair, i.e., *F8* from the *Testing Dataset*. In this iteration, we obtain an accuracy of 80.97% on considering 14 features out of 16. As discussed in Algorithm-3, we proceed to the next iteration whenever the detection accuracy increases from the previous iteration. Hence, we eliminate the bottom three ranked network traffic features and repeat the entire procedure. The procedure is terminated until we observe a potential decrease in the detection accuracy. As highlighted in the table 5.5, we achieved the highest detection accuracy of 99.55% with DT classifier upon eliminating 14 of the total 16 traffic features, i.e., upon considering only the top two features, namely *Packet_size_received (F12)* and *Time_interval_between_ packets_received (F3)*, the highest detection accuracy can be achieved. From the next iteration, we observe that the detection accuracy starts decreasing.

Finally, we conclude that we obtain the highest accuracy of 99.55% when we apply the proposed Algorithm-3 to the *Testing Dataset*. It can be seen from Table 5.3 that the feature *F3* has been ranked lower than {*F5*, *F1*, *F15* and *F11*} according to *crRelevance*, yet our proposed feature selection algorithm filters out the top-ranked feature *F12* with *F3* as the best subset. This can be justified with the help of the *NMRS* rankings in Table 5.4. We would like to highlight that the correlation between the *crRelevance* top-ranked feature *F12* and *F3* is found to be much lesser than the correlation between *F12* and other features such as *F5*, *F1*, *F15* or *F11*. Hence, the elimination of such highly correlated features by our proposed feature selection algorithm to reduce the redundancy between the entire set is justified.

At the same time, when no feature ranking /statistical or correlation technique of any type is used and all the features are fed to the classifiers at once, i.e., on considering all the 16 net-

Table 5.5: Detection results when we apply our *NMRS*-based proposed algorithm on *crRelevance* feature ranking

Feature eliminated	**Features used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**BC**	**NB**	**GB**	**AB**	**MLP**	**DNN**	**CNN**
None (Using all features)	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16	81.12	81.12	80.09	55.34	79.18	80.10	58.75	72.32	59.5
F1	F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16	81.62	81.67	81.46	55.58	79.14	80.19	58.26	73.41	59.42
F8	F2, F3, F4, F5, F6, F7, F9, F10, F11, F12, F13, F14, F15, F16	80.97	80.97	80.88	55.93	83.33	82.55	63.44	71.44	61.44
F9	F2, F3, F4, F5, F6, F7, F10, F11, F12, F13, F14, F15, F16	83.39	83.39	83.38	56.89	82.40	79.45	56.98	70.89	56.55
F14	F2, F3, F4, F5, F6, F7, F10, F11, F12, F13, F15, F16	78.17	78.20	78.04	50.93	77.90	76.25	50.23	69.77	56.55
F6	F2, F3, F4, F5, F7, F10, F11, F12, F13, F15, F16	78.57	78.54	78.44	48.30	78.50	78.65	52.68	69.38	57.45
F2	F3, F4, F5, F7, F10, F11, F12, F13, F15, F16	77.75	77.93	77.81	46.99	78.72	79.65	55.8	73.23	58.15
F16	F3, F4, F5, F7, F10, F11, F12, F13, F15	78.57	78.59	78.57	46.75	78.75	77.57	51.05	62.45	53.51
F10	F3, F4, F5, F7, F11, F12, F13, F15	81.04	80.76	80.94	46.34	80.11	77.65	54.59	69.12	54.96
F4	F3, F5, F7, F11, F12, F13, F15	72.82	75.54	72.79	49.59	71.57	70.20	64.03	76.82	56.51
F11	F3, F5, F7, F12, F13, F15	77.64	77.32	77.62	48.70	73.71	67.79	61.80	60.41	51.17
F7	F3, F5, F12, F13, F15	54.36	5.99	54.40	46.67	51.10	49.68	62.56	63.36	51.49
F13	F3, F5, F12, F15	87.18	87.17	87.082	52.45	83.50	72.22	61.07	80.13	52.23
F15	F3, F5, F12	85.30	85.31	84.87	43.15	83.87	80.72	56.85	77.04	65.62
F5	**F3, F12**	**99.50**	95.95	95.85	53.84	97.44	94.39	52.32	75.13	73.34

work traffic features simultaneously, we observe that the highest detection accuracy obtained is merely 81.12%. **Based on the results and the low detection accuracy depicted by Table 5.5, we answer our research question one that feature ranking helps us eliminate irrelevant features that can hamper detection accuracy.**

5.3.3 Comparison with other statistical tests

In this subsection, we compare the performance of our proposed model with that of some commonly used statistical tests for Android malware detection. Our proposed approach involved computing the feature– class correlation, i.e., crRelevance ranking, and further deploying an *NMRS*-based detection algorithm to select the best set of features that produce higher detection accuracy. Hence, to compare our model's performance, we deploy the same *NMRS*-based detection algorithm on the following statistical measures:

1. **Chi-square** - The chi-square test measures the difference between the expected and observed values and determines whether the deviation between the observed and expected

values is acceptable or not. The chi-square formula [152] is defined by equation 5.3.1:

$$\chi_c{}^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{5.3.1}$$

where:

c=Degrees of freedom,

O=Observed value(s), and

E=Expected value(s)

This test is applied separately to malware traffic and the normal feature set. For any particular feature, say F_i, we have its range for malware samples. From these values, we calculate the average of the feature value to serve as the expected value for our experiments. We then measure how the values for that feature deviate from the expected value using the chi-square formula. The procedure is repeated for normal traffic and all features. The closer the values are to the average value, the lower the chi-square score and the higher the priority of the feature. Therefore, unlike the statistical measure used in our study, i.e., *crRelevance*, where a higher score leads to a higher ranking of the feature, a lower chi-square value leads to a higher ranking of the feature. Moreover, applying chi-square gives two separate rankings for features, i.e., one for the normal and another for the malware class.

2. **ANOVA** - ANOVA, or Analysis of Variance [163], is a statistical method used to analyze the differences among group means in a sample. It is often used to test the null hypothesis that the means of three or more groups are equal. In our case, normal and malware values of the same feature from the training dataset are taken as the two groups. The one-way ANOVA F-statistic is calculated using the following formula:

$$F = \frac{\text{Between-Group Variance (MSB)}}{\text{Within-Group Variance (MSW)}} \tag{5.3.2}$$

where:

MSB (Mean Square Between) is the variance among the group means and MSW (Mean Square Within) is the average of the variances within each group. Lastly, the features are ranked in order of their F-statistic value such that the feature with the highest score is the most distinguishing one.

3. **Mann-Whitney** - The Mann-Whitney U test [164], also known as the Wilcoxon rank-sum

test, is a non-parametric test used to determine whether there is a difference between two independent and randomly selected groups. The U statistic is calculated using the following formula:

$$U_1 = R_1 - \frac{n_1 \cdot (n_1 + 1)}{2} \tag{5.3.3}$$

$$U_2 = R_2 - \frac{n_2 \cdot (n_2 + 1)}{2} \tag{5.3.4}$$

where:

- U_1 and U_2 are the U statistics for Group 1 and Group 2, respectively.
- R_1 and R_2 are the sums of ranks for Group 1 and Group 2, respectively.
- n_1 and n_2 are the sample sizes for Group 1 and Group 2, respectively.

In our case, normal and malware features from the training dataset are considered Group 1 and Group 2, and the sample size for each group is taken as the number of flows. Lastly, the features are ranked in order of their U statistic value such that the feature with the highest score is the most distinguishing one.

4. **Kruskal-Wallis** - The Kruskal-Wallis test [165] is a non-parametric test used to determine whether there are statistically significant differences between three or more independent groups. The test assesses whether the samples originate from the same distribution or if at least one of the samples is different from the others. The formula for the test statistic is given by equation 5.3.5.

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1) \tag{5.3.5}$$

where:

- N is the total number of observations across all groups.
- k is the number of groups.
- R_i is the sum of ranks for group i.
- n_i is the number of observations in group i.

In our case, normal and malware features from the training dataset are considered as the two groups and the sample size for each group is taken as the number of flows. Lastly, the features are ranked in order of their H statistic value such that the feature with the highest score is the most distinguishing feature.

Table 5.6 summarizes the individual test rankings when we apply chi-square on the normal feature set and chi-square on the malware feature set along with other measures such

as ANOVA, Mann–Whitney U test, and Kruskal– Wallis test. As can be seen from the table, *Ratio_of_incoming_to_ outgoing_packets (F5)* has been ranked as the most distinguishing feature by Mann– Whitney as well as both rankings of the chi-square test, whereas, according to ANOVA and Kruskal– Wallis, the feature named *Time_interval_between_ packets_received (F3)* resides at the top of the table as the best feature.

Table 5.6: Traffic features ranked using various statistical tests

chi-square on normal	**chi-square on malware**	**ANOVA**	**Mann-Whitney**	**Kruskal-Wallis**
F5	F5	F3	F5	F3
F2	F6	F4	F7	F5
F3	F7	F7	F14	F12
F6	F1	F5	F10	F12
F9	F2	F2	F11	F6
F10	F12	F16	F9	F8
F7	F10	F1	F2	F15
F1	F4	F10	F13	F7
F4	F8	F8	F16	F4
F12	F11	F11	F4	F14
F8	F3	F13	F15	F10
F11	F9	F6	F8	F16
F14	F13	F15	F6	F11
F16	F16	F9	F1	F13
F13	F15	F12	F12	F9
F15	F14	F14	F13	F2

For comparison, we ranked network traffic features using the four statistical tests described above and further applied our proposed *NMRS*-based proposed algorithm on the *Testing Dataset* to obtain their corresponding accuracies. The results are summarized in Table 5.7 and it can be understood as follows. When we apply the *NMRS*-based proposed approach on chi-square rankings expressed on a normal training dataset as well as on Kruskal– Wallis rankings, we achieve the highest detection accuracy of 96.22% after eliminating 14 features out of the total lot of 16 traffic features, i.e., upon considering two features, namely *Ratio_of_incoming_to_ outgoing_packets (F5)* and *Time_interval_ between_ packets_received (F3)*, the highest detection accuracy can be achieved. At the same time, the highest detection accuracy of 96.37% can be achieved upon considering two features, namely *Ratio_of_incoming_to_ outgoing_packets (F5)* and *Time_interval_between_ packets_sent (F2)* when we apply the *NMRS*-based proposed algorithm on Mann– Whitney test rankings as well as chi-square test rankings for malware dataset. Similarly, when we apply the *NMRS*-based proposed algorithm to ANOVA test rankings, we obtain the highest detection accuracy of 98.10% while considering two features, namely *Time_interval_ between_ packets_received (F3)* and *Flow_Duration (F4)*. When we apply the *NMRS*-based proposed algorithm to the *crRelevance* rankings used in our work, we obtain an accuracy of 99.5% with two features, namely *Packet_size_received (F12)* and *Time_interval_between_ packets_received*

(F3). Hence, our model outperforms other similar statistical tests when we apply our proposed *NMRS* algorithm to their developed rankings.

Table 5.7: Comparsion of *NMRS*-based proposed algorithm on *crRelevance* rankings with various statistical tests when we apply the same algorithm applied on them

Statistical test used	**Features used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**BC**	**NB**	**GB**	**AB**	**MLP**	**DNN**	**CNN**
***crRelevance* (our approach)**	F12, F3	**99.5**	95.95	95.88	53.84	97.44	94.39	52.32	75.13	73.34
chi-square on normal	F5, F3	**96.22**	85.60	84.64	51.22	86.20	84.50	38.96	66.88	51.05
chi-square on malware	F5, F2	**96.37**	86.13	86.12	59.42	86.64	84.86	44.58	78.41	59.28
ANOVA	F3, F4	**98.10**	96.46	96.381	72.54	95.91	95.71	55.62	85.05	62.93
Mann-Whitney	F5, F2	**96.37**	86.13	86.12	59.42	86.64	84.86	44.58	78.41	59.28
Kruskal-Wallis	F3, F5	**96.22**	85.60	84.64	51.22	86.20	84.50	38.96	66.88	51.05

5.3.4 Comparison of *NMRS* with other Correlation Measures

In the previous subsection, we compared the performance of our proposed *NMRS*-based algorithm with that of other statistical tests when we applied the same *NMRS*-based algorithm to their rankings. In this section, we compare the performance of *NMRS* as an algorithm itself with a similar correlation measure called Pearson's correlation coefficient, i.e., we compute the feature– feature correlation score for all feature pairs using Pearson's correlation test and replace *NMRS* with Pearson's correlation coefficient as our chosen algorithm base to be applied upon the *crRelevance* feature rankings. Next, we briefly describe the working of Pearson's correlation coefficient.

The **Pearson correlation coefficient**, often denoted by r, is a measure of the linear relationship between two variables. It quantifies the degree to which a pair of variables change together. The formula for the Pearson correlation coefficient between two variables X and Y with n data points is as follows:

$$r = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}} \tag{5.3.6}$$

Where:

- X_i and Y_i are the individual data points for variables X and Y,

- $\bar{X}$ and $\bar{Y}$ are means of variables X and Y respectively.

In our case, because we wish to find the feature– feature correlation between all possible

pairs without any relation to their classes, we concatenate the normal and malware traffic feature values, i.e., we combine the flows of both class labels for each feature and then compute the Pearson correlation scores. In this way, we iteratively compute the Pearson's correlation coefficient score for each feature pair by considering them as X and Y variables in the above-mentioned formula. Hence, in the next step, we applied Pearson's correlation coefficient on all possible feature pairs (120 to be precise) and ranked them in such a way that the pair that is found to be the most correlated resides at the top of the table. Table 5.8 summarizes the top ten traffic feature pairs ranked on the basis of their Pearson correlation scores. The table can be understood as follows: The features *Packets_sent_per_second (F9)* and *Bytes_sent_per_second (F14)* are found to have the best correlation score and thus have the best correlation between them. Similarly, the features *Packets_sent_per_second (F9)* and *Packet_size_received (F12)* have the worst correlation between them and are listed at the end of the table.

Table 5.8: Top 10 Traffic feature pairs ranked using Pearson's correlation coefficient and their corresponding correlation scores

Feature Pair	**Pearson's correlation coefficient scores**
F9 - F14	0.989299
F8 - F11	0.938651
F1 - F12	0.918476
F11 - F15	0.817451
F10 - F16	0.764166
F1 - F6	0.732062
F6 -F12	0.729424
F8 - F15	0.728135
F2 - F4	0.466446
F6 - F16	0.436238

Next, we deploy Pearson correlation pairwise rankings instead of *NMRS* in our proposed algorithm and apply it to the ranked feature list given by *crRelevance*. The table 5.9 summarizes the detection results for the same. When we applied Pearson's correlation coefficient-based algorithm to our *crRelevance* rankings, we achieved the highest detection accuracy of 99.3% on considering two features, namely *Packets_sent_per_second (F9)* and *Packet_size_received (F12)*. However, with our proposed *NMRS*-based algorithm applied to *crRelevance* rankings, the highest detection accuracy of 99.5% can be achieved. Hence, we can conclude that *NMRS* is the best choice for calculating the feature– feature correlation and reducing the redundancy between them.

Table 5.9: Detection results when we apply Pearson's correlation coefficient-based algorithm on *crRelevance* feature ranking

Statistical test used	**Features used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**BC**	**NB**	**GB**	**AB**	**MLP**	**DNN**	**CNN**
***NMRS*-based proposed algorithm on *crRelevance* rankings**	F12, F3	**99.5**	95.95	95.88	53.84	97.44	94.39	52.32	75.13	73.34
Pearson's correlation coefficient based algorithm on *crRelevance* rankings	F12, F9	**99.3**	95.33	95.27	70.26	85.41	93.70	94.97	95.61	71.44

5.3.5 Comparsion with other related works

In this subsection, we compare the performance of our proposed *NMRS*-based algorithm on *crRelevance* rankings with another similar work incorporating TCP flows. We do so by implementing the approach followed by the authors in [95]. Similar to the working of their proposed model, we chose Decision Table and REPTree classifiers as the base learners for the *local learning model*. These classifiers were trained for each of the 16 features. The authors stated that when a feature vector representing a normal event is tested against the models generated during the learning phase, there is a higher probability that the predicted value will match or be very similar to the observed value. The more the predictions differ from the true values of the corresponding features, the more likely it is that the observed vector comes from a distribution different from that of the training set. Therefore, we ranked the features in order of their probability of deviation from the pattern of normal traffic features. Table 5.10 summarizes the traffic features ranked in order of their deviation from normal traffic behavior. As can be seen from the table, the feature *Packets_sent_per_flow (F8)* shows the highest probability of coming from an abnormal event. At the same time, the feature *Ratio_of_incoming_to_ outgoing_packets (F5)* scored the lowest relating to the least probability of coming from an abnormal event, and thus was situated at the end of the table. We would like to point out that we did not multiply all the individual probabilities following the approach used by the authors because of our vast dataset; instead, we added all the individual probabilities.

After ranking the network traffic features in order of their probability of deviation from normal traffic features' pattern, we move on to find the best feature set with higher detection accuracy. To do so, a threshold needs to be decided to filter out the best features capable of distinguishing between normal and anomalous vectors that are learned during the algorithm calibration phase. Table 5.11 summarizes the detection results using Decision Table and Rep-

Table 5.10: Traffic features ranked based on their deviation from normal traffic behavior

Traffic feature names	Probability scores
F8	248988.163
F15	248987.688
F11	248986.1281
F4	248970.6183
F13	248970.0707
F9	248955.3305
F14	248951.6604
F2	248867.7914
F16	248761.2274
F3	248747.6066
F10	248700.7806
F6	247767.6992
F12	244789.8246
F7	232428.8028
F1	230456.9791
F5	228989.9963

Tree classifiers as base learners. As it can be seen, upon eliminating four features out of the total set of 16 features, we managed to achieve the highest detection accuracy of 95.85% while using both Decision Table and RepTree classifiers as base learners, i.e., on considering the top 12 features, namely {*F8, F15, F11, F4, F13, F9, F14, F2, F16, F3, F10, and F6*}, both display the same highest detection accuracy of 95.85%. Whereas, with our *NMRS*-based proposed algorithm applied to *crRelevance* rankings highest detection accuracy of 99.5% can be achieved. Hence, we can conclude that our proposed approach outperforms the similar work of Android malware detection performing feature selection on TCP-based network traffic features.

Table 5.11: Detection results when we implement other related work

Statistical test used	**Features used**	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
		DT	**RF**	**BC**	**NB**	**GB**	**AB**	**MLP**	**DNN**	**CNN**
***NMRS*-based proposed algorithm on *crRelevance* rankings**	F12, F3	**99.5**	95.95	95.88	53.84	97.44	94.39	52.32	75.13	73.34
Shabtai et al. [95] approach (DT as the base learner)	F8, F15, F11, F4, F13, F9, F14, F2, F16, F3, F10, F6,	95.85	81.20	80.72	56.89	80.35	76.62	61.88	68.65	59.42
Shabtai et al. [95] approach (RepTree as the base learner)	F8, F15, F11, F4, F13, F9, F14, F2, F16, F3, F10, F6,	95.85	81.20	80.72	56.89	80.35	76.62	61.88	68.65	59.42

Furthermore, we compare some other similar works of Android malware detection that use TCP flows in terms of detection accuracy. Table 5.12 summarizes this comparison. Some studies have attempted to determine the probability of deviation from normal traffic features' patterns, whereas others have ranked the features using tests such as chi-square, information gain, and frequency ranges to select the best subset of features. As shown in the table, our

model outperforms all these studies in terms of detection accuracy. Hence, we can conclude that our proposed model is better than many state-of-the-art techniques presented in the literature for Android malware detection.

Table 5.12: Comparison of proposed work with the existing literature based on malware detection using TCP flows

Related Work	**Methodology**	**Detection accuracy (in %)**
Wang et al. [104]	ML- based malware detection using TCP and HTTP features	97.89
Arora and Peddoju [92]	Features ranked using IG and Chi square	97.3
Arora and Peddoju [166]	Calculated similarity scores between FP-Growth algorithm generated patterns and testing features	94.25
Upadhayay et al. [167]	Thresholds are set for testing on Frequency based rankings of permissions merged with network traffic rankings	95.96
Zulkifli et al. [101]	ML-based malware detection using TCP-based network traffic features	98.4%
Sihag et al. [168]	DL-based system using image form of features for binary classification	98.44
Alshehri [169]	Devised flow similarity using the Euclidian Algorithm	97.32
Liu et al. [107]	GNN model using node characteristics as well as the edge attributes of mobile traffic features	97
***NMRS*-based proposed algorithm on *crRelevance* rankings (our approach)**	**Correlation-based Feature selection using *crRelevance* and *NMRS***	**99.50**

5.4 Limitations

Now, we describe a few limitations of the proposed approach. The proposed model ranks TCP-based network traffic features for detection; hence, it falls under the category of dynamic analysis. The path of dynamic analysis overcomes several limitations of static analysis but also poses some barriers. Not all malware samples generate network traffic. It has been noticed that some malware might only send text messages in the background without generating any network traffic. Hence, network traffic-based detection mechanisms cannot detect such samples. Dynamic analysis tools can introduce performance overhead as they monitor and analyze the execution of the program. This overhead may affect the timing and behavior of the software, potentially masking certain performance-related issues. Moreover, some mobile attacks can be due to colluding apps, i.e., malicious behavior is distributed across several apps rather than one. However, the proposed model, in its current form, does not target colluding apps. Therefore, to further enhance the detection capability of the proposed model, we aim to target colluding apps in our future work.

5.5 Conclusion and Future Work

In this chapter, we ranked the network traffic features in order of their correlation with the class and amongst themselves using two statistical measures, namely *crRelevance* and *NMRS*. Subsequently, we proposed a novel *NMRS*-based detection algorithm to select the best and inversely correlated features by applying various machine learning and deep learning techniques. The experimental results highlight that our proposed *NMRS*-based detection algorithm on *crRelevance* rankings can effectively reduce the feature set while detecting Android malware with 99.50% accuracy on considering two network traffic features, namely *Packet_size_received* and *Time_interval_between_ packets_received*. Furthermore, our results showed that our proposed method is better than other statistical tests such as chi-square, ANOVA, Mann–Whitney U test, Kruskal–Wallis test, Pearson's correlation coefficient, and at the same time from other similar works of Android malware detection. Moreover, the proposed model can detect Android malware with better accuracy than various state-of-the-art techniques. In our future work, we aim to enhance the capabilities of our model by including malware category and family classification along with the binary classification performed in this study. We also aim to integrate static features to deal with the limitations of dynamic analysis and possibly build a robust hybrid detection model.

Chapter 6

Hybrid Android Malware Detection leveraging Static Permissions and Dynamic System Calls

---

In this chapter, we propose two hybrid detection models leveraging the merits of both static permissions and dynamic system calls. In Section 6.1, we explain the motivation behind proposing a hybrid Android malware detector model and a brief overview of the two models proposed. In Section 6.2, we explain in detail the methodology behind our first model named *AndroVRank*. Section 6.3 explains the results obtained from the proposed model - I. In Section 6.4, we explain the methodology behind our proposed hybrid model - II. Section 6.5 discusses the results obtained from the second hybrid model and finally, we conclude the chapter in Section 6.6.

6.1 Introduction

Static analysis, though fast and efficient for processing large-scale applications, has critical weaknesses. Malware developers can easily bypass it through techniques like code obfuscation, polymorphism, and encryption, and it struggles to detect runtime behaviors. Dynamic analysis addresses these gaps by monitoring application behavior in a controlled execution

environment, revealing malicious activities that static analysis may miss, such as network interactions or file system modifications. Yet, dynamic analysis also has limitations—it is resource-heavy, time-consuming, and can be fooled by malware that alters its behavior when sensing an analysis environment. Additionally, if a malicious app does not rely on network traffic, dynamic analysis using network-based features can fail. To mitigate these issues, hybrid analysis offers a more comprehensive solution by blending static and dynamic methods for stronger, more reliable detection. This study focuses on hybrid analysis integrating static code inspection with dynamic behavior tracking to create two hybrid malware detection models, aiming to boost detection accuracy and address the shortcomings of each technique.

With the first model *AndroV-Rank*, we aim to enhance Android malware detection by examining and ranking static permissions and dynamically extracted system calls. Our analysis revealed considerable overlap in commonly utilized features between the normal and malicious app classes. This result highlighted the importance of effective feature selection in enhancing detection accuracy. To achieve this, we utilized a *Multi-Criteria Decision-Making (MCDM)* approach, employing the *VIKOR* method to rank features according to their ability to distinguish between classes.

Classification involves assigning class labels to unlabeled test samples using a trained model. Various algorithms are widely employed in data mining and machine learning, including eager learners like Artificial Neural Networks (ANN), Decision Trees (DT), Naïve Bayes (NB), and statistical approaches like Linear and Logistic Regression (LR). Ensemble methods such as Random Forest (RF), AdaBoost, and Support Vector Machine (SVM) are also frequently used. Although these methods are common, they face challenges. Traditional approaches often suffer from low classification accuracy, while more complex techniques can lead to misclassification or be hindered by computational demands despite their higher accuracy.

Such limitations of static and dynamic analysis, along with the challenges faced while applying the traditional ML classifiers in the context of Android malware detection motivate us to build another simple yet effective hybrid detection model capable of producing optimal detection results without the use of any ML or DL algorithm. Hence, next, we propose an instance-based classifier *PattMatch*, that utilizes an *Average Weighted Pattern Score (AWPS)* technique in conjunction with *Attribute Score-based Ranking (ASR)* to accurately predict the class labels for unlabeled test samples.

Contributions - The major contributions of this chapter are as follows-

- We developed two hybrid Android malware detection systems that overcome the limitations of static and dynamic analysis by integrating both methods, thereby enhancing the detection performance.
- We implemented *VIKOR*, a *Multi-Criteria Decision-Making (MCDM) technique* in the first model, to rank features based on their *preference scores*, effectively measuring their ability to distinguish between malware and benign apps.
- We introduce a novel algorithm that utilizes the individual rankings of static, dynamic, and hybrid datasets derived from *VIKOR* to develop a robust malware detection system capable of achieving higher accuracy with fewer features.
- Additionally, we discuss another model introducing a simple yet efficient instance-based pattern-matching classifier capable of predicting the class labels for test samples without relying on traditional data mining algorithms.
- We advanced the field by employing an attribute rank-based feature selection method, which significantly improved the detection accuracy.

6.2 Proposed Hybrid Model - I

This section outlines the methodology of our first proposed model, *AndroV-Rank*, which is divided into two main parts: the *Ranking* and *Detection* segments. . First, we construct a comprehensive dataset incorporating static features (permissions) and dynamic features (system calls). This dataset is then divided into two parts: the *Training* set and the *Testing* set. In the *Ranking* segment, we process the *Training* set to rank features using the *VIKOR* method, where a preference score is computed for each feature. In the *Detection* segment, we introduce an innovative algorithm that utilizes both Machine Learning (ML) and Deep Learning (DL) techniques to select optimal features, enhancing detection accuracy.

Ranking Segment

6.2.1 Dataset Accumulation

To begin our research, we needed a comprehensive hybrid dataset that combined both static and dynamic features for detailed analysis. We chose the publicly available *Kronodroid* dataset

[170], which covers Android's evolution from 2008 to 2020, making it highly relevant for our study. The malware applications are gathered from repositories like Drebin, AMD, VirusTotal, and VirusShare whereas the benign applications come from F-droid, MARVIN, and APKMirror. The dataset includes numerous static permissions and dynamically extracted system calls, where permissions were represented in binary (1 for requested, 0 for not), and system calls recorded the frequency of their occurrence, with zeros indicating no call was made. The dataset comprised 78,137 applications, of which 41,382 were classified as malware and 36,755 as benign. For our analysis, we selected 20,000 applications (10,000 benign and 10,000 malware) as the *Testing* dataset, while the remaining 58,137 applications were used for *Training*. After preprocessing, 137 distinct permissions and 124 unique system calls were identified and retained as features in the dataset.

Our objective is to create a model that utilizes hybrid features for Android malware detection, aiming to overcome the limitations present in both static and dynamic analysis approaches. To enable a comprehensive comparison of the three analysis techniques, we divided the dataset into three distinct categories for both the *Training* and *Testing* sets: static, dynamic, and hybrid. The static category includes datasets with 137 permissions, the dynamic category comprises datasets with 124 system calls, and the hybrid category combines both, yielding a total of 261 features.

6.2.2 Features Ranking

Multi-Criteria Decision Making (MCDM) [171] is recognized as a highly effective approach for decision-making. This method incorporates a range of qualitative and quantitative criteria that must be clearly defined to determine the best alternative or feature. The fundamental steps in *MCDM* involve identifying criteria, assigning weights to them, ranking the available alternatives or features based on preferences, and ultimately selecting the best choice or a subset thereof.

The primary goal of *MCDM* is to categorize attributes as either preferrable or non preferrable. In our study, we applied *VIKOR*, a widely used *MCDM* technique to rank features from the static, dynamic, and hybrid datasets according to their preference. Further information on the technique used in our research is provided below:

VIKOR - *VIKOR* stands for "VIekriterijumsko KOmpromisno Rangiranje", a Serbian term for "multi-criteria optimization and compromise solution". This method reaches the final de-

cision of choosing the best alternative after forming the compromised ranking from the aggregating function called L_p metric. The series of steps involved in a *VIKOR* application is described below for a *MCDM* problem defined on m alternatives denoted as $A_1, A_2, \ldots, A_m$ and n decision criteria:

Step 1 - For the alternative A_1, the evaluation of the jth criterion is represented by f_{ij}. Identify the optimal f_j^* and the minimal f_j^- values for all criterion functions where $j = 1, 2, \ldots, n$ [171]. If the jth function indicates a benefit, then:

$$f_j^* = \max_i f_{ij}, \quad f_j^- = \min_i f_{ij}$$

if non-benefit, then vice versa.

Step 2 - Compute the values S_i and $R_i; i = 1, 2, \ldots, m$ [171], by these relations:

$$S_i = \sum_{j=1}^{n} w_j [(f_j^* - f_{ij}) / (f_j^* - f_j^-)]$$

$$R_i = \max_j w_j [(f_j^* - f_{ij}) / (f_j^* - f_j^-)],$$

where w_j are the weights of criteria, expressing their relative importance.

Step 3 - Compute the values $Q; i = 1, 2, \ldots, m$ [171], by the following relation:

$$Q_i = v[(S_i - S^*) / (S^- - S^*)] + (1 - v)[(R_i - R^*) / (R^- - R^*)]$$

where

$$S^* = \min_i S_i, \quad S^- = \max_i S_i,$$
$$R^* = \min_i R_i, \quad R^- = \max_i R_i,$$

v is introduced as the weight of the strategy of "the majority of criteria", here suppose that $v = 0.5$.

We applied *VIKOR* to the permissions vector tables of the static dataset, which contains information on both benign and malware applications, along with their requested permissions, aiming to compute *preference scores* (Q_i) for all the attributes. Permissions were treated as alternatives (i), and applications as criteria (j), with equal weights assigned to all applications since they contribute equally to the decision-making process. Benign applications were considered as benefit criteria, while malware applications were treated as cost criteria.

After completing **Steps 1- 3**, we successfully computed Q_i values for each permission. To evaluate the class-distinguishing ability of these permissions, we ranked them in decreasing order of *preference scores*, generating a *VIKOR*-ranked list of permissions.

The same steps were applied to the dynamic *Training* dataset for ranking system calls and to the hybrid dataset to compute combined rankings of permissions and system calls.

6.2.3 Machine Learning and Deep Learning Classifiers

In our detection strategy, we employed a variety of algorithms from both machine learning (ML) and deep learning (DL). The ML classifiers included, but were not limited to, Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest (RF), while the DL methods featured architectures such as Dense Neural Network (DNN) and Multilayer Perceptron (MLP). This diverse set of techniques, encompassing Bagging Classifier (BC), Gaussian Naive Bayes (NB), Decision Trees (DT), and Artificial Neural Networks (ANN) also, allowed us to effectively analyze the datasets [135].

Detection Segment

6.2.4 Proposed Malware Detection Algorithm

This section details our detection algorithm for identifying the most effective features for malware classification. We begin by ranking the features from the *Training* dataset using the *VIKOR* method, which assigns *preference scores* based on their ability to distinguish between malware and benign apps. Starting with the top-ranked feature, we evaluate its detection accuracy on the *Testing* dataset using both ML and DL techniques. The maximum accuracy is updated whenever a higher value is reached. In each iteration, the next highest-ranked feature is added, and classification accuracy is re-evaluated with the expanded feature set. This process continues until there is no further improvement in accuracy. The algorithm stops when no new features lead to higher accuracy, outputting the highest achieved accuracy and the set of features that contributed to it, optimizing malware detection performance.

The following section presents a discussion of the results obtained from the proposed approach.

6.3 Results and Discussion: Hybrid Model - I

In this section, we present the experimental results from applying our proposed malware detection model to the three dataset types: static, dynamic, and hybrid. As outlined earlier, the *VIKOR* method was employed on the *Training* dataset to calculate the *preference scores* of the features. Following this, in the *Detection* segment, we used several ML and DL classifiers on the *Testing* dataset to evaluate performance.

6.3.1 Feature Ranking using *VIKOR*

In this subsection, we discuss the rankings obtained using the proposed method, *VIKOR*, when applied to the three dataset categories. Tables 6.1 and 6.2 illustrate the top ranked features and their corresponding *preference scores*.

Table 6.1 presents the top 10 permissions ranked using our proposed method for the static category datasets. Notably, the permission *READ_PHONE_STATE* achieved the highest preference score of 0.5, placing it at the top of the table. Conversely, *FOREGROUND_SERVICE* received the lowest preference score, ranking last among all 137 permissions. Similarly, the system call *ioctl* ranked highest with a score of 0.5, while *sigaction* obtained the lowest score among the total of 124 system calls.

Table 6.1: Top 10 features ranked according to their *preference scores* using VIKOR for the static and dynamic category datasets

Static Category		**Dynamic Category**	
Permissions	**Preference scores**	**System Calls**	**Preference scores**
READ_PHONE_STATE	0.5	ioctl	0.5
ACCESS_WIFI_STATE	0.398538961	clock_gettime	0.477343173
WRITE_EXTERNAL_STORAGE	0.364966631	mprotect	0.310328142
RECEIVE_BOOT_COMPLETED	0.334573413	futex	0.209853072
SEND_SMS	0.326569264	mmap2	0.202570002
ACCESS_NETWORK_STATE	0.306479978	write	0.157811795
ACCESS_COARSE_LOCATION	0.291486291	getuid32	0.151138712
WAKE_LOCK	0.288825758	read	0.148676708
RECEIVE_SMS	0.270269661	SYS_310	0.139770505
ACCESS_FINE_LOCATION	0.268984488	munmap	0.121863244

Table 6.2 presents the top 30 permissions and system calls when the proposed ranking method using *VIKOR* was applied to the hybrid category *Training* dataset. The system call *ioctl* again ranks highest with a score of 0.5, while *sigaction* is at the bottom of the list comprising 261 attributes. The rankings and the scores of the other attributes can be understood from the table.

Table 6.2: Top 30 features ranked according to their *preference scores* using *VIKOR* for the hybrid category datasets

Features	Preference scores	Features	Preference scores
ioctl	0.5	SYS_317	0.088036
clock_gettime	0.476491	gettimeofday	0.086151
mprotect	0.310389	prctl	0.085812
futex	0.209874	READ_PHONE_STATE	0.079389
mmap2	0.20258	WRITE_EXTERNAL_STORAGE	0.074338
write	0.157857	ACCESS_NETWORK_STATE	0.069696
getuid32	0.150976	madvise	0.065325
read	0.148896	RECEIVE_BOOT_COMPLETED	0.064888
SYS_310	0.139958	getpid	0.06471
munmap	0.121839	ACCESS_WIFI_STATE	0.063018
pread	0.117367	SEND_SMS	0.058569
close	0.117344	WAKE_LOCK	0.056827
SYS_305	0.111424	writev	0.053507
rt_sigprocmask	0.107207	GET_TASKS	0.052355
fstat64	0.093932	READ_SMS	0.050868

6.3.2 Detection results on the Testing Dataset

This subsection presents the detection results obtained by applying ML and DL algorithms to the *Testing* datasets for the three categories considered in this study. Figure 6.1 displays the maximum detection accuracy achieved as the number of features in each category is adjusted. Although accuracies were recorded for the full set of features, the figure highlights the range from five to 120 features, as further iterations did not result in significant improvements in classification accuracy beyond this point.

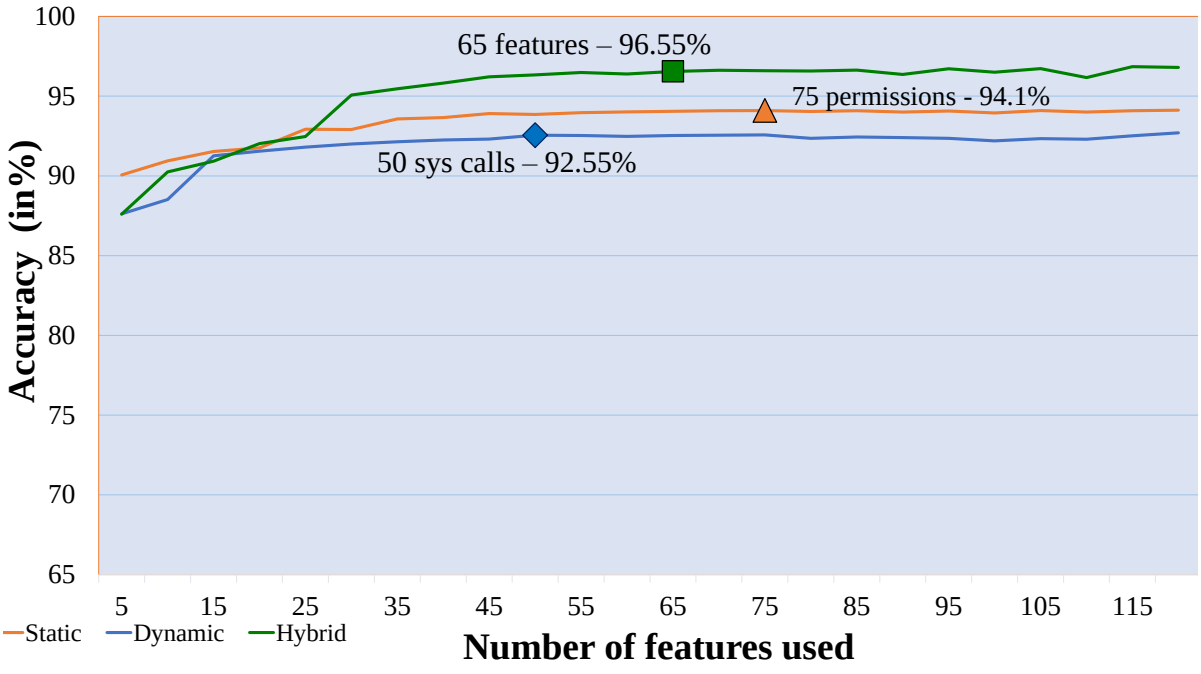


Figure 6.1: Detection results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets

Figure 6.1 presents the detection outcomes. We first applied ML and DL classifiers to the static *Testing* dataset, ranking the features using the *VIKOR* method. As additional features were incorporated, detection accuracy improved. Using only the top five permissions, most classifiers achieved an accuracy of 90.06%. As more permissions were added, accuracy continued to rise, reaching a maximum of 94.1% when 75 permissions were used with the Random Forest (RF) classifier. This marked the optimal point, with 75 out of 137 permissions (approximately 54.7%) yielding the best performance.

Next, we applied the classifiers to the dynamic *Testing* dataset, which contained system calls. With just five system calls, an accuracy of 87.62% was achieved. As the number of system calls increased, accuracy peaked at 92.55% using 50 system calls, reducing the set upto 40.3% of the total 124 system calls.

Finally, we evaluated the hybrid dataset, combining both permissions and system calls, ranked through the *VIKOR* method. Initially, the top five features produced an accuracy of 87.6%. As more features were incorporated, the highest accuracy reached 96.55% with 65 features, which constitutes approximately 24.9% of the total 261 features used in the study.

To summarize the classification results, Table 6.3 provides the detection performance of ML and DL classifiers across the three *Testing* datasets. The data indicate that static analysis, using 75 permissions, achieved a peak accuracy of 94.1%, while dynamic analysis, utilizing 50 system calls, reached a maximum accuracy of 92.55%.

Table 6.3: Compiled detection results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets

Number of features used	**Detection accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**SVM**	**BC**	**NB**	**LR**	**MLP**	**ANN**	**DNN**
75 permissions	93.16	**94.1**	93.18	93.36	72.21	90.89	93.52	90.66	90.66
50 system calls	88.36	**92.55**	70.94	91.38	67.53	72.63	83.8	85.82	86.95
65 permissions and system calls	93.69	**96.55**	70.91	95.58	67.04	77.85	90.55	81.16	82.74

These results underscore the fact that, while static and dynamic analyses each have their advantages, they are limited when employed in isolation. Static analysis may fail to capture runtime behaviors, whereas dynamic analysis can miss essential static features. In contrast, a hybrid approach that integrates both techniques harnesses their combined strengths. The proposed model demonstrated this by achieving a superior accuracy of 96.55% with just 65 features, highlighting its effectiveness in precisely classifying the datasets.

6.4 Proposed Hybrid Model - II

With our second proposed work, we aim to build a simple and robust hybrid analysis-based Android malware detection model capable of categorizing the test applications as benign or malware by matching them with the patterns of the training samples. Simultaneously, our goal is to achieve this objective by utilizing only the least and the best features amongst the total lot of the two feature types used for this study, i.e., permissions and system calls. Therefore, this work introduces an instance-based pattern-matching classifier, *PattMatch*, that utilizes an *Average Weighted Pattern Score (AWPS)* technique in conjunction with *Attribute Score-based Ranking (ASR)* to accurately predict the class labels for test samples.

The following research questions arise while considering the proposal of a hybrid detection model centering on a novel classifier.

- **RQ1** How can a classifier be constructed from the ground up to distinguish between benign and malicious applications without employing traditional data mining techniques?
- **RQ2** What necessitates the selection of relevant features, and why is feature reduction more advantageous than using all features as inputs simultaneously?
- **RQ3** How can feature ranking be utilized to eliminate irrelevant features, and what methods can be employed to rank them?

Our motivation is in the pursuit of tackling the limitations of both static and dynamic analysis by developing an Android malware classifier, *PattMatch*. Additionally, an *Attribute Score-based Ranking (ASR)* [172] algorithm has been presented to choose relevant features. This method calculates the attribute rank by considering the number of unique values in the set of training samples. The classification accuracy, or detection accuracy, is measured by comparing the predicted class labels generated by the classifier with the original class labels of the test samples in the end. By evaluating the degree of correspondence between these original and predicted labels, we can determine how accurately the model has performed in classifying the test samples. The major advantage of the proposed classifier is its simple working and ability to overcome most shortcomings of the existing classifiers. For instance, neural networks and AdaBoost are known to be prone to overfitting [173], the splitting process in the Decision trees may lead to loss of information [174], and the performance of algorithms like Naive Bayes, Linear regression, and Support Vector Machines is affected by the number of training samples,

output variables or the missing values [175]. These limitations make the combinations of *AWPS* and *ASR* the most appropriate choice for our study. This method is straightforward and doesn't require the use of any fancy data mining technique; nonetheless, it yields superior classification accuracy compared to other conventional classifiers such as the most commonly used ML and DL algorithms. Moreover, our experiments demonstrate that the proposed instance-based pattern-matching classifier outperforms other similar works on Android malware detection, which we evaluate against the same dataset of normal and malicious apps.

Figure 6.2 provides a concise and comprehensive overview of our proposed hybrid Android malware detection model revolving around our instance-based pattern-matching classifier. In the first step, we gather a vast dataset of the static permissions and the dynamic system calls for our study. In the next step, we divide the dataset into two parts namely, *Training* and *Testing* dataset. Both datasets are fed into our proposed classifier, *PattMatch*, which operates in two stages: the *Classification* stage and the *Feature Selection* stage. In the *Classification* stage, the classifier combines *Average Weighted Pattern Score (AWPS)* and *Attribute Score-based Ranking (ASR)* techniques to generate initial classification results and observe the detection accuracy. Following this, the process moves to the *Feature Selection* stage, where *ASR* ranking is utilized to iteratively select and retain only the most relevant features from both datasets. The reduced datasets, now with fewer features, are then reprocessed through the *Classification* stage to evaluate any improvements in classification accuracy. This iterative process of alternating between the *Classification* and *Feature Selection* stages continues until the optimal classification accuracy is achieved.

The subsequent subsections provide a comprehensive analysis of all the steps mentioned above.

6.4.1 Dataset Accumulation

To begin our research, we required a comprehensive hybrid dataset that included both static and dynamic features, meticulously organized for optimal use. For this purpose, we utilized the same publicly available *Kronodroid* [170] dataset utilized for the first proposed model *AndroV-Rank*. As discussed in Subsection 6.2.1, the *Kronodroid* dataset was extensive, encompassing numerous static permissions and dynamically extracted system calls. The permissions dataset was already in binary form, where a value of one indicated that the application requested certain permission, and zero denoted its absence. The system call dataset, on the other hand,

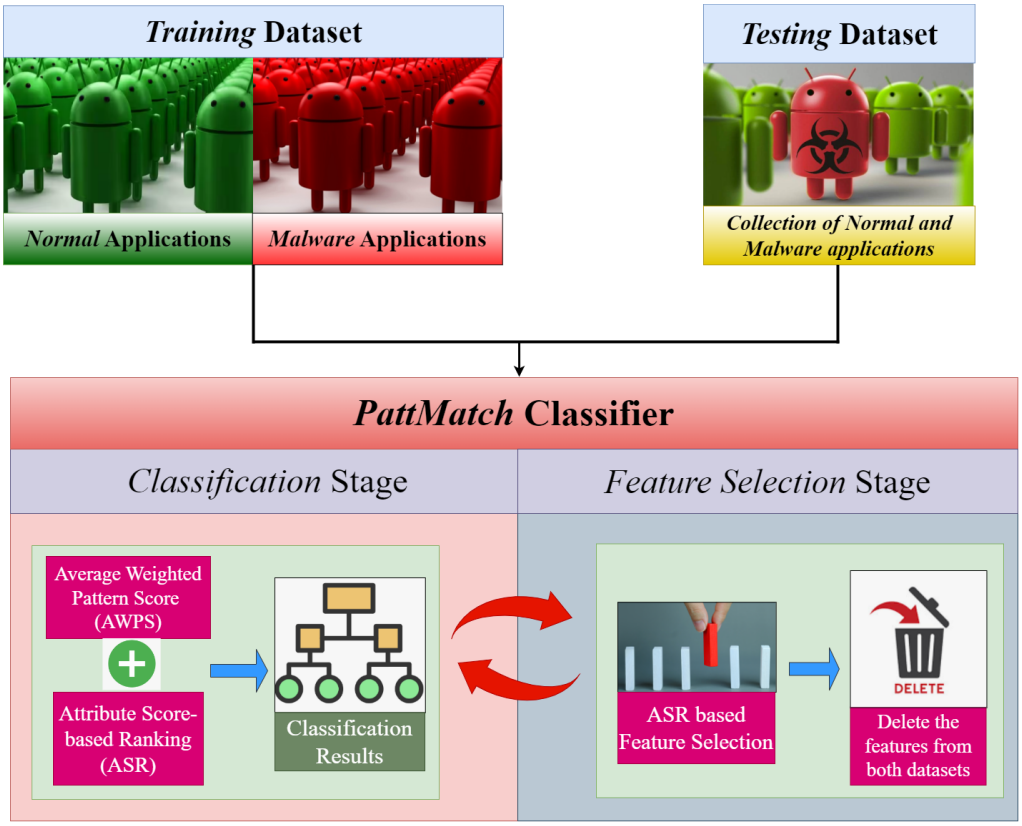


Figure 6.2: *PattMatch* System Design

was initially constructed with zeros representing the absence of a system call and non-zero values indicating the frequency of requests made by the application. To maintain a consistent approach for both permissions and system calls, we converted the system call dataset into binary form as well. This conversion ensured a uniform data structure, facilitating more straightforward analysis and comparison between the two types of data. The dataset consists of 78,137 rows representing applications, with 41,382 classified as malware and 36,755 as benign. From the total dataset, we selected 20,000 applications (10,000 benign and 10,000 malware) to serve as the *Testing* dataset. The remaining 58,137 applications were designated as the *Training* dataset. After some initial preprocessing, we identified 137 distinct permissions and 124 unique system calls, retained as columns in the dataset.

In order to carry out a comprehensive comparison between the three analysis techniques, we further processed the dataset to create three distinct categories for both the *Testing* and *Training* datasets, namely static, dynamic, and hybrid. The static category includes datasets comprised solely of 137 permissions, the dynamic category includes datasets comprised solely of 124 system calls, and the hybrid category includes datasets that combine both permissions and system calls, summing up to a total of 261 features.

Permissions provide a high-level overview, while system calls offer granular details. This multi-faceted approach helps in building more robust and precise detection systems by cross-validating suspicious behaviors from different perspectives. Permissions can be analyzed before an app is installed, providing an opportunity to flag potentially malicious apps early based on their permission requests. System calls, on the other hand, allow real-time monitoring of app behavior, enabling the detection of malware activities as they happen. This combination allows for both proactive and reactive malware detection. These benefits make permissions and system calls a perfect choice for our hybrid detection model.

6.4.2 Methodology

This section presents a detailed discussion of the *Pattmatch* classifier introduced in our study along with its working in Algorithm 4, which answers research question one, i.e., how to construct a classifier from scratch capable of distinguishing between benign and malicious applications without employing traditional data mining techniques.

The proposed classifier operates in two stages. In the *Classification* stage, we initially use the complete feature set and employ a method called *Average Weighted Pattern Score (AWPS)* combined with *Attribute Score-based Ranking (ASR)* [172] to generate match scores. These scores are then used to predict the category label of the test samples as either benign or malware. Given that the *Kronodroid* dataset used in our study is already labeled, we compare the predicted labels with the original ones to count the number of correct predictions. The classification accuracy at this stage is calculated as the ratio of correct predictions to the total number of test samples, multiplied by 100. To enhance the classification accuracy and reduce the number of false predictions, we proceed to the *Feature Selection* stage of the proposed classifier, which involves feature selection based on the rankings provided by the *ASR* method. In this stage, we reduce the number of features in both the *Testing* and *Training* datasets by omitting the lower-ranked irrelevant features of the *ASR* list. Next, we will discuss in detail the working of the methods used in the two stages.

Classification Stage

The proposed *PattMatch* classifier employs an *Average Weighted Pattern Scoring (AWPS)* mechanism in combination with *Attribute Score-based Ranking (ASR)*. This method improves upon the PMC algorithm proposed by Sreeja and Sankar [176] by including weights and rank weights

Algorithm 4 Hybrid Detection Model

Input: *Training* set D, unlabeled test sample x_{test} from *Testing* dataset
Output: Predicted class labels, Classification accuracy

Classification Stage

$m \leftarrow$ number of attributes
$n \leftarrow$ number of instances
$k \leftarrow$ number of distinct classes
Let $\mathbb{C} = \{C_1, C_2, \ldots, C_k\}$ be the set of all distinct classes.
$Score(D) \leftarrow$ Dataset's overall score
$A_{score} \leftarrow$ Attribute score
$w(a_j) \leftarrow$ Attribute weights
$\mathbb{G}_c \leftarrow$ The set of training samples belonging to class C_j
$n_{a'}(x_{ic})$ for each $x_{ic} \in \mathbb{G}_c \leftarrow$ Attribute match count.
$\hat{c}_{\text{test}} \leftarrow$ Predicted class label for x_{test}.
$Score(D)$=Avg$(\sum_{i=1}^{k} p_i^{p_i})$
for $i = 1$ **to** n in D **do**
    p_i=$\frac{\text{Number of instances belonging to } C_i}{\text{Total number of instances in } D}$
end for
for $j = 1$ **to** m in D **do**
    for $i = 1$ **to** n in D **do**
        $A_{score} = Score(D) - \text{Avg}(\sum_{j=1}^{n} p_{n_j} \times Score(G_j))$
    // Sort the attribute scores and rank the attributes
    end for
end for
for $j = 1$ **to** n **do**
    $w(a_j) = \frac{(n - r(a_j) + 1)}{\sum_{i=1}^{n}(n - r(a_i) + 1)}$ // Assigning weights to attributes
end for
for $j = 1$ **to** m **do**
    for $i = 1$ **to** n **do**
        if $a_j \in x_{\text{test}} = a_j \in x_D$ **then**
            $s(a_j) \leftarrow 1$
        else
            $s(a_j) \leftarrow 0$
        end if
    end for
    $n_{a'}(m) \leftarrow n_{a'}(m) + s(a_j)$ // Matching patterns to calculate match count
end for
$i_k \in \mathbb{G}_c$ **iff** $n_a^{'}(k) =$ maximum and $i_k \in C_j$
for each selected instance in the group **do**
    for $j = 1$ **to** n **do**
        $PS(x_i)$=$\sum_{i=1}^{n} S(a_i) \times w(a_i)$
    end for
end for
$x_{\text{test}} \in C_j$ **iff** $PS(x_c) = \max(PS(x_c))$
Predict the class labels for all instances in the *Testing* dataset and calculate classification accuracy

Feature Selection Stage

Initialize $n =$ initial number of features
while $n > 0$ **do**
    $n \leftarrow n - x$
    Remove x lower-ranked features from *Training* and *Testing* datasets
    Execute ***Classification*** **Stage** with reduced features
    Record accuracy for current n
end while

for attributes. Let's consider a training dataset, denoted as D consisting of n attributes labeled as $a_1, a_2, a_3, \ldots, a_n$ and m instances labeled as $i_1, i_2, i_3, \ldots, i_m$, it is represented in the matrix form shown in (6.4.1).

		Attributes			
		a_1	a_2	$\cdots$	a_n
Instances	i_1	X_{11}	X_{12}	$\cdots$	X_{1n}
	i_2	X_{21}	X_{22}	$\cdots$	X_{2n}
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	i_m	X_{m1}	X_{m2}	$\cdots$	X_{mn}

(6.4.1)

The values $\mathrm{x_{m1}}, \mathrm{x_{m2}}, \ldots, \mathrm{x_{mn}}$ in the given matrix represent the attributes of the $\mathrm{m^{th}}$ instance, where n is the number of attributes. Each instance is assigned to a certain class $\mathrm{C_k}$, where k ranges from 1 to p.

Prior to categorization, the importance of each attribute is established as a pre-processing step. Attributes are prioritized based on their significance using the *ASR* method. In order to do so, firstly the database's overall score is determined by the count of unique classes, which is calculated according to equation (6.4.2).

$$\mathrm{Score}(D) = \mathrm{Avg}\left(\sum_{i=1}^{k} p_i^{p_i}\right) \tag{6.4.2}$$

where p_i is the probability that an arbitrary instance in D that belongs to class $\mathrm{C_k}$ [172]. The formula to calculate $\mathrm{p_i}$ is shown in (6.4.3).

$$p_i = \frac{\text{Number of instances belonging to } C_i}{\text{Total number of instances in } D} \tag{6.4.3}$$

In order to determine the attribute score for each attribute, when there are a total of n distinct values, the tuples with n distinct values $\{\mathrm{n_1}, \mathrm{n_2}, \mathrm{n_3}, \ldots, \mathrm{n_j}\}$ are grouped together as $\{\mathrm{G_1}, \mathrm{G_2}, \ldots, \mathrm{G_j}\}$ [172]. The formula for calculating the attribute score, denoted as $\mathrm{A_{score}}$ is given in (6.4.4).

$$A_{\mathrm{Score}} = \mathrm{Score}(D) - \mathrm{Avg}\left(\sum_{j=1}^{n} p(n_j) * \mathrm{Score}(G_j)\right) \tag{6.4.4}$$

where the variable $p(n_j)$ represents the probability of an arbitrary instance in $\mathrm{G_j}$ belonging

to class C_i. The score for each attribute is computed and the ranking is determined for all attributes using the *ASR* method, sorting them in order of relevance such that the attribute having the highest relevance is given rank 1.

Within the initial stage, the complete feature set $a_1, a_2, a_3, \ldots, a_n$, without performing any feature selection, is given rankings ranging from 1 to n. Then, each attribute is assigned a weight using the rank sum weight method [172]. This approach computes and standardizes the weights so that the total weight of all the ranked features combined is equal to 1, as depicted in the formula in equation (6.4.5).

$$w(a_j) = \frac{(n - r(a_j) + 1)}{\sum_{i=1}^{n} (n - r(a_i) + 1)} \tag{6.4.5}$$

The main objective of the proposed classifier is to ascertain the class label for test instances that do not have a label. The approach uses the PMC algorithm [176] to detect the instances with the greatest count of matching attributes. This is done by comparing the attribute values of the training instances and test instances, as shown in equations (6.4.6) and (6.4.7). The number of matches for the m^{th} instance is represented as $n_{a'}(m)$, and its computation is given in (6.4.6)

$$n_{a'}(m) = \sum_{i=1}^{n} s(a_i) \tag{6.4.6}$$

In this context, $s(a_i)$ denotes the attribute match score, which has a binary value of either 0 or 1. The score is 1 if the value of attribute a_i in the test sample is identical to the corresponding attribute value in the training sample; otherwise, the score is 0 [172]. This is stated in equation (6.4.7).

$$s(a_i) = \begin{cases} 1, \text{ if the value of } a_i \text{ matches} \\ \text{with the training sample } a_{im} \\ 0, \quad \text{Otherwise} \end{cases} \tag{6.4.7}$$

The approach subsequently selects the training samples that exhibit the highest number of attribute matches with the provided test sample [176]. The chosen training samples are further categorized according on their class labels, as depicted in (6.4.8).

$$i_k \in \mathbb{G}_c \text{ iff } n_{a'}(k) = \text{ maximum and } i_k \in C_j \tag{6.4.8}$$

In $\mathbb{G}c$, c denotes the number of classes in the chosen training set samples. All instances with the maximum na' and belonging to class C_j are grouped together. The rank weights of the selected attributes are then applied to these chosen training instances to determine the pattern score of the test sample t concerning the training samples x_i, denoted as $PS(x_i)$ [172], with the formula shown as:

$$PS(x_i) = \sum_{i=1}^{n} s(a_i) * w(a_i) \tag{6.4.9}$$

The average pattern score $PS(x_{ic})$ for class c is computed by finding the mean score within each group of the chosen training set samples. The pattern score of the test sample, denoted as PS_{test}, is the highest score obtained by averaging the pattern scores of groups $\mathbb{G}_c$. In the end, the predicted class label for the test sample is determined by the class group that has the greatest average pattern score [172], as described in equations (6.4.10) and (6.4.11).

$$PS_{\text{test}} = \max(PS(x_c)) \tag{6.4.10}$$

$$i_{\text{test}} \in C_j \text{ iff } PS(x_c) = PS_{\text{test}} \tag{6.4.11}$$

In case multiple groups attain the maximum average pattern score, the class label is predicted based on the number of training instances with the highest attribute match count. Specifically, the class label for the test sample is determined by the class with the greatest number of training instances exhibiting the highest attribute match count and the expression is shown in (6.4.12).

$$\hat{c}_{\text{test}} = \arg\max_{c} \left\{ \sum_{x_{ic} \in \mathbb{G}_c} n_{a'}(x_{ic}) \right\} \tag{6.4.12}$$

where $\hat{c}_{\text{test}}$ represents the predicted class label for the test sample, and $\mathbb{G}_c$ denotes the set of training samples belonging to class c.

In our case, the features, be it the set of permissions, system calls, or permissions-system calls combined, form the attributes in columns whereas the application samples are the instances in rows. In the first iteration, *AWPS* combined with *ASR* is applied to the complete set of n features, and classification accuracy is noted down.

Feature Selection stage

Attribute selection is a crucial stage that enhances the accuracy of classification. With the belief that feature selection reduces the total number of false predictions, instead of using n features, we reduce the feature set to a lower value of n by omitting the lower-ranked features of the *ASR* list from both the *Testing* and *Training* datasets. Consequently, the lower-ranked feature columns are eliminated from both the *Training* and *Testing* datasets before feeding them into the classifier for pattern matching. After this, the complete *Classification* Stage was again executed using the reduced value of n, and classification accuracy was observed. We denote this as the second iteration. Feature selection is an additional step designed to rectify falsely predicted test samples. Hence, only the test samples whose class labels have not yet been correctly predicted are subjected to the *Classification* stage in subsequent iterations.

In our research, the feature selection process was conducted by eliminating 20 features at a time after rounding off the total number of features to a convenient starting point. For instance, with an initial total of 137 permissions we execute the *Classification* stage and mark it as the first iteration. Then we rounded it down to 130 by removing seven features, giving us the new reduced value of n to be 130. This was followed by executing the *Classification* Stage and classification accuracy was observed. This constituted the second iteration. In the third iteration, we eliminate 20 features at a time, reducing the total lot to 110 permissions. Classification accuracy is duly noted for the reduced feature set. In subsequent iterations the value of n is reduced by intervals of 20, reducing the total to 90 for the fourth iteration and to 70 for the fifth iteration, and so on, till the value of n becomes 10. This process continued until the total number of features in both the *Training* and *Testing* datasets reached 10. At this point, the interval was adjusted to one, and one feature was eliminated at a time in each iteration. This iterative process continued until no features remained in either dataset. The classification accuracy, calculated at each iteration, was analyzed at the end of the process.

Using the *ASR*-based feature selection algorithm as mentioned above, we answer research question three, i.e., how to rank the features to eliminate the irrelevant ones.

6.5 Results and Discussions: Hybrid Model - II

In this section, we present and discuss the experimental results obtained using the proposed *PattMatch* classifier. As discussed in subsection 6.4.2, the proposed classifier employs a method called *Average Weighted Pattern Score (AWPS)* combined with *Attribute Score-based Ranking (ASR)* [172]. If the following subsections, we first discuss the rankings derived from the *ASR* method. We then describe the classification results on the *Testing* dataset when the proposed classifier is executed for each of the three categories, namely static, dynamic, and hybrid. Finally, we compare our proposed model with similar models for Android malware detection.

6.5.1 *Attribute Score-based Ranking (ASR)* results

The dataset's overall score is calculated in each case using the formula given in equation (6.4.2). The attribute score for each feature is calculated using equation (6.4.4). After sorting the features based on these scores, the rank weights are determined according to their ranks using the formula provided in equation (6.4.5).

Tables 6.4 and 6.5 provide a comprehensive summary of the top 10 permissions and system calls ranked using *ASR* for static and dynamic categories respectively, including their corresponding attribute scores, attribute ranks, and rank weights. In a similar manner, Table 6.6 showcases the top 10 ranked features when *ASR* is applied to the hybrid dataset of permissions and system calls, along with their attribute scores, attribute ranks, and rank weights. As shown in Table 6.4, the permission *SET_PREFERRED_APPLICATIONS* holds the top position with the highest attribute score, a rank of one, and a rank weight of 0.014492754. The scores and weights of the subsequent ranked permissions are detailed in the table. Conversely, the permission *READ_PHONE_STATE* is identified as the least relevant, with the lowest attribute score of 0.30804777, a rank of 137, and a rank weight of 0.000105787. Similarly, Table 6.5 shows that the system call named *pipe* achieves the highest attribute score using the *ASR* method, earning it the top rank and a rank weight of 0.016. The details for other system calls are provided in the table. The system call *SYS_333* has the lowest attribute score of 0.327097242134042, a rank of 137, and a rank weight of 0.000129032258064516.

The total number of permissions considered for our study is 137 whereas the total number of system calls is 124. When combining permissions and system calls as a part of the hybrid model, the total number of features becomes 261. As shown in Table 6.6, the per-

Table 6.4: Top 10 permissions ranked with *ASR* along with their corresponding attribute scores and rank weights

Permissions	Attribute Score	Rank Weight
SET_PREFERRED_APPLICATIONS	0.354140277	0.014492754
READ_SYNC_STATS	0.354140266	0.014386967
SET_PROCESS_LIMIT	0.354140239	0.014281181
CONTROL_LOCATION_UPDATES	0.354140047	0.014175394
REBOOT	0.354139987	0.014069608
SET_TIME	0.354139609	0.013963821
ACCESS_CHECKIN_PROPERTIES	0.354139253	0.013858034
SET_ALARM	0.354139215	0.013752248
WRITE_CALENDAR	0.354139086	0.013646461
READ_INPUT_STATE	0.354139032	0.013540675

Table 6.5: Top 10 system calls ranked with *ASR* along with their corresponding attribute scores and rank weights

System calls	Attribute Score	Rank Weight
pipe	0.35414	0.016
SYS_341	0.35414	0.015871
shutdown	0.35414	0.015742
listen	0.35414	0.015613
getgid32	0.354139	0.015484
nanosleep	0.354139	0.015355
getegid32	0.354139	0.015226
ptrace	0.354139	0.015097
SYS_318	0.354138	0.014968
sendfile64	0.354138	0.014839

mission *SET_PREFERRED_APPLICATIONS* is identified as the most relevant feature among all permissions and system calls according to the *ASR* method, placing it at the top of the table. This top rank corresponds to a rank weight of 0.007633588. Conversely, the feature *READ_PHONE_STATE* is ranked last, at position 261, with an attribute score of 0.308047769601107 and a rank weight of 0.0000292474627826036.

Table 6.6: Top 10 permissions and system calls ranked with *ASR* along with their corresponding attribute scores and rank weights

Features	Attribute Score	Rank Weight
SET_PREFERRED_APPLICATIONS	0.354140277	0.007633588
READ_SYNC_STATS	0.354140266	0.00760434
pipe	0.354140264	0.007575093
SYS_341	0.354140263	0.007545845
SET_PROCESS_LIMIT	0.354140239	0.007516598
shutdown	0.354140217	0.00748735
CONTROL_LOCATION_UPDATES	0.354140047	0.007458103
REBOOT	0.354139987	0.007428856
listen	0.35413972	0.007399608
SET_TIME	0.354139609	0.007370361

In the following subsection, we present the classification results obtained using the proposed classifier.

6.5.2 Classification results on *Testing* dataset

This section focuses on presenting the classification results achieved by our proposed classifier when applied to the three dataset categories: static, dynamic, and hybrid. Tables 6.7, 6.8, and 6.9 summarize the number of features used, the number of correct predictions, and the classification accuracy whereas the Figures 6.3, 6.4, and 6.5 illustrate the decrease in the number of false predictions as irrelevant features were omitted and only relevant features were utilized for each category.

Classification results for static category

We applied the proposed *PattMatch* classifier to the static category *Training* and *Testing* datasets first, each containing 137 permissions. Table 6.7 displays the total number of correct predictions made by our classifier out of the 20,000 testing samples, alongside the classification accuracy results for each iteration.

In the initial iteration, where the complete set of permissions was utilized, the proposed classifier correctly predicted the class labels for 18,769 test samples, resulting in a classification accuracy of 93.84%. In the second iteration, with the number of features reduced to 130, there was an increase in correct predictions, raising the classification accuracy to 95.53%. Details of subsequent iterations are provided in the table. Notably, the highest number of correct predictions was observed when the static feature set was reduced to only 10 permissions, achieving an accuracy of 99.57% with 19,915 correct predictions out of 20,000 test samples.

Table 6.7: Classification results obtained by applying the proposed classifier to the static category dataset

Number of permissions used	**Number of correct predictions (out of 20,000)**	**Classification accuracy (in %)**
137 (All permissions)	18769	93.84
130	19107	95.53
110	19377	96.88
90	19889	99.44
70	19893	99.46
50	19908	99.54
30	19913	99.56
10	**19915**	**99.57**

Classification results for dynamic category

Moving on with our research plan, we then applied the proposed classifier to the dynamic category *Training* and *Testing* datasets, each containing 124 system calls. Table 6.8 displays the

total number of correct predictions made by our classifier out of the 20,000 testing samples, alongside the classification accuracy results for each iteration.

In the initial iteration, where the complete set of system calls was utilized, the proposed classifier correctly predicted the class labels for 16,906 test samples, resulting in a classification accuracy of 84.53%. In the second iteration, with the number of features reduced to 110, there was an increase in correct predictions, raising the classification accuracy to 88.22%. Details of subsequent iterations are provided in the table. Notably, the highest number of correct predictions was observed when the dynmaic feature set was reduced to only 10 system calls, achieving an accuracy of 98.90% with 19,781 correct predictions out of 20,000 test samples.

Table 6.8: Classification results obtained by applying the proposed classifier to the dynamic category dataset

Number of system calls used	**Number of correct predictions (out of 20,000)**	**Classification accuracy (in %)**
124 (All system calls)	16906	84.53
110	17644	88.22
90	18704	93.52
70	19092	95.46
50	19689	98.44
30	19780	98.9
10	**19781**	**98.90**

Classification results for hybrid category

Lastly, with an aim to further elevate the detection accuracy by combining the static and dynamic features, we applied the proposed classifier to the hybrid category *Training* and *Testing* datasets, each containing a total of 261 features. Table 6.9 displays the total number of correct predictions made by our classifier out of the 20,000 testing samples, alongside the classification accuracy results for each iteration.

In the initial iteration, where the complete set of features was utilized, the proposed classifier correctly predicted the class labels for 19,185 test samples, resulting in a classification accuracy of 95.92%. In the second iteration, with the number of features reduced to 250, there was an increase in correct predictions, raising the classification accuracy to 96.97%. Details of subsequent iterations are provided in the table. Notably, the highest number of correct predictions was observed when the hybrid feature set was reduced to only 10 features, achieving a remarkable accuracy of 99.93% with 19,987 correct predictions out of 20,000 test samples.

Furthermore, the significance of feature selection is demonstrated in Figures 6.3, 6.4, and 6.5, where a gradual decrease in the frequency of false predictions is observed in all three

Table 6.9: Classification results obtained by applying the proposed classifier to the hybrid category dataset

Number of features used	**Number of correct predictions (out of 20,000)**	**Classification accuracy (in %)**
261 (All permissions and system calls)	19185	95.92
250	19394	96.97
230	19582	97.91
210	19708	98.54
190	19790	98.95
170	19822	99.11
150	19861	99.30
130	19966	99.83
110	19975	99.87
90	19977	99.88
70	19983	99.91
50	19983	99.91
30	19984	99.92
10	**19987**	**99.93**

categories as irrelevant features were iteratively omitted. Figure 6.3 illustrates that utilizing the entire set of 137 permissions leads to 1231 false predictions. By excluding the seven least important permissions, the value of n decreases to 130, resulting in a significant reduction in the number of false predictions to 893. The figure clearly illustrates a substantial decrease. As we decrease the value of n and provide fewer permissions to the classifier, the number of false predictions continues to decrease. The frequency of false predictions is minimized when only 10 permissions are utilized for detection. According to the classification method, we now decrease the value of n by one instead of 20. However, no subsequent alteration in the number of false predictions was noted. Therefore, the optimal classification accuracy with minimal false predictions is attained by utilizing 10 permissions.

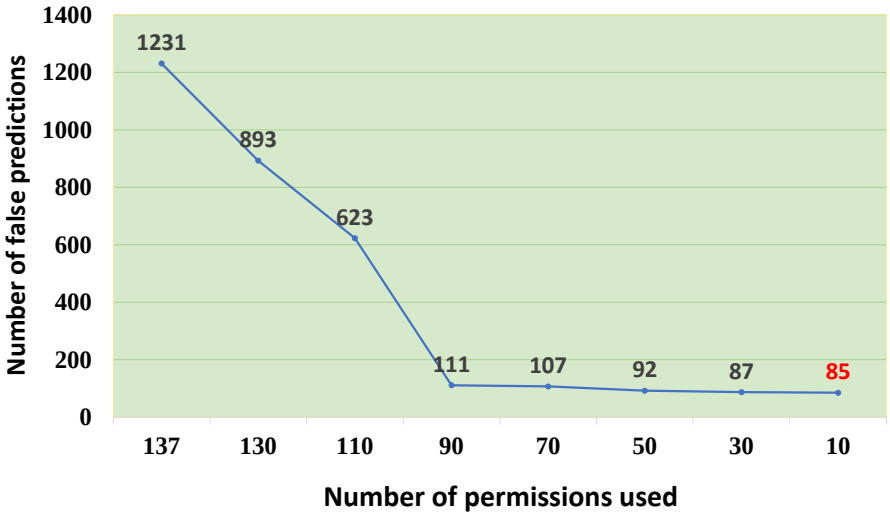


Figure 6.3: Classification results obtained by applying the proposed classifier to the static category dataset

A similar trend can be observed for the cases of dynamic and hybrid categories. For instance, it can be seen in Figure 6.4, when all system calls were used, the number of false predictions was 3,094. By iteratively omitting the least relevant system calls, the number of false predictions was reduced to a minimum of 219.

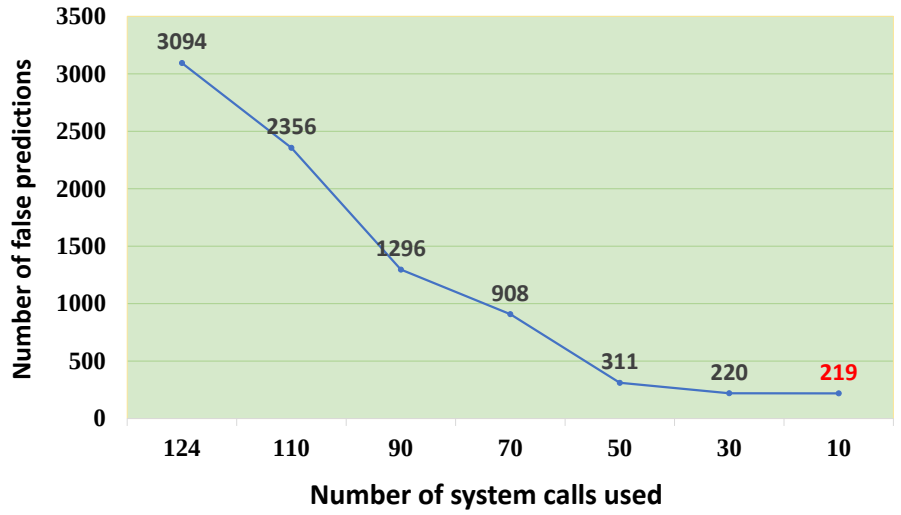


Figure 6.4: Classification results obtained by applying the proposed classifier to the dynamic category dataset

Similarly, in the hybrid category, as shown in Figure 6.5, using all 261 features resulted in 815 false predictions. However, when only the top 10 permissions and system calls were used, the number of false predictions decreased significantly to just 13.

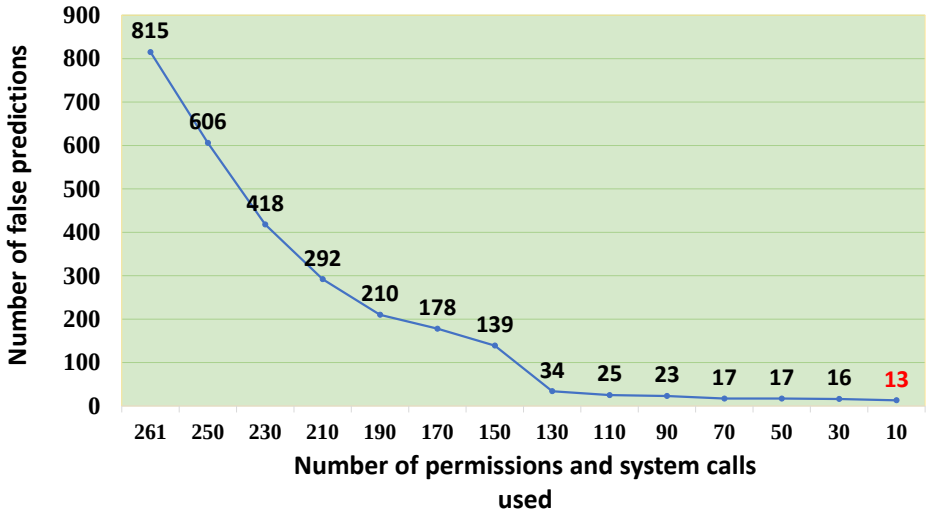


Figure 6.5: Classification results obtained by applying the proposed classifier to the hybrid category dataset

Discussion - We would like to emphasize that the classification accuracy was notably low

when no feature ranking or selection was applied, and all features were simultaneously fed into the classifier. As evidenced in Tables 6.7 and 6.8, utilizing all 137 permissions or 124 system calls resulted in significantly reduced classification accuracies such as 93.84% and 84.53% respectively. **Based on the results and the low detection accuracy depicted by Tables 6.7 and 6.8, we answer our research question two that feature selection helps us eliminate irrelevant features that can hamper detection accuracy.**

The motivation behind the proposed research was to address the limitations inherent in both static analysis and dynamic analysis by combining them into a hybrid detection model, aiming for a more robust approach to detection. It was hypothesized that some test samples might not be accurately classified based solely on permissions but could be correctly predicted using only system calls, and vice versa. This hypothesis was confirmed when preliminary results showed that even with the most relevant 10 permissions, there were 85 incorrect predictions. However, when these test samples were put through our proposed classifier as a part of dynamic category datasets, a majority of these misclassifications were corrected in the second iteration of using system calls.

These findings underscored that static analysis alone lacks certain insights that dynamic analysis can provide, and vice versa. By combining these approaches, a more effective and efficient detection model can be developed. Experimental results demonstrated that the hybrid approach significantly outperformed using static or dynamic analysis alone. Specifically, the hybrid model led to only 13 false predictions, compared to 85 and 219 when using static analysis or dynamic analysis alone, respectively. This highlights the capability of the hybrid detection model to accurately classify benign and malicious software more reliably.

6.5.3 Comparison with other classifiers

In this subsection, we compare the performance of our proposed classifier with some widely used classification algorithms used in the literature. In particular, we computed the detection accuracy using ML classifiers [135] such as Decision Tree (DT), Random Forest (RF), Support Vector Machines (SVM), Bagging (BC), Naive Bayes (NB), Logistic Regression (LR) and simultaneously using some DL classifiers namely, Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Artificial Neural Network (ANN), and fully connected Deep Neural Network (DNN) were evaluated against the same dataset of normal and malicious apps as ours. We would like to highlight that we did not utilize any feature ranking or selection

techniques before applying the ML and DL classifiers, i.e., the complete testing dataset was fed to the classification algorithms for detection for each category. Figures 6.6 summarize the comparison results when we apply the ML and DL algorithms for the classification of malware along with our proposed classifier in case of static, dynamic, and hybrid category *Testing* datasets respectively. When applying ML and DL classifiers, we observed that the Random Forest (RF) classifier achieved the highest detection accuracy of 94.18% while using permissions. In contrast, our proposed classifier significantly outperformed these methods, achieving the highest classification accuracy of 99.57% with only 10 permissions.

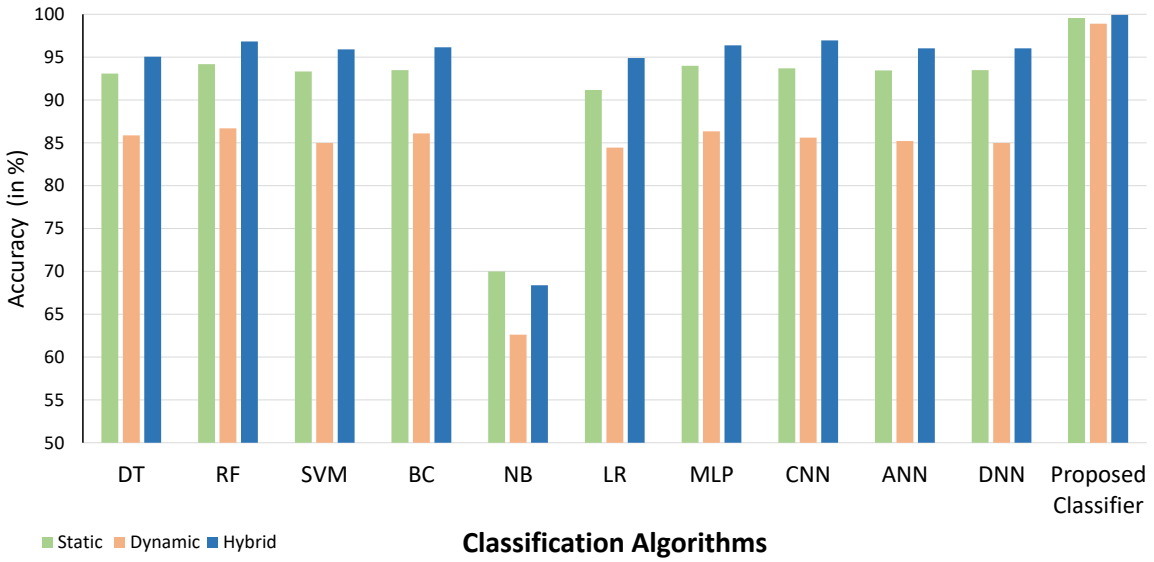


Figure 6.6: Comparison of our proposed classifier with other ML and DL algorithms when applied to the static, dynamic, and hybrid category datasets

Among the ML and DL classifiers, the highest accuracy is again achieved by the Random Forest (RF) classifier for the case of dynamic category datasets. However, our proposed classifier attains a maximum accuracy of 98.9% using 10 system calls, demonstrating superior performance.

In a similar manner, the CNN classifier achieves the highest accuracy among the conventional classifiers with 96.95% when applied to the hybrid category datasets, whereas our proposed classifier achieves an impressive accuracy of 99.93%, easily surpassing the other methods. This indicates that our proposed classifier *PattMatch* consistently outperforms the other conventional classification algorithms across all three categories in terms of detection accuracy.

6.5.4 Comparison with other related works

In this section, we compare the performance of our proposed classifier with other methodologies that incorporate permissions or system calls for Android malware detection. Specifically, we implement the approaches adapted in studies [120], [177] and [178]

Anupama et al. [120] proposed a hybrid methodology that combined permissions and system calls to create a detection model utilizing a range of machine learning and deep learning classifiers. Feature reduction was carried out using a customized Fisher score algorithm. The selected features were then used to train various machine learning classifiers, including Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), and Logistic Regression (LR). We applied their proposed approach to all three category datasets used in our study.

Another study that we implemented was centered on a dynamic analysis detection model utilizing system calls presented by Bhat et al. [177]. This study employed enhanced classification algorithm techniques such as Bagging (BC), AdaBoost (AdaB), and Stacking, along with individual classifiers like Decision Tree (DT), Support Vector Machines (SVM), Logistic Regression (LR), and Naive Bayes (NB). As a preprocessing step to reduce feature set dimensionality, they omitted features with low chi-square scores, deeming them irrelevant. In a similar manner to the previous study, we meticulously adhered to their methodology, utilizing the same classifiers across all three categories of our datasets. Consequently, we ranked the features based on their chi-square scores. The authors specifically selected the top 30 features from these rankings for their classifiers, discarding the rest, and we adopted the same approach.

Lastly, we implemented the approach described in [178]. The authors presented an Android malware detection system that utilized only the relevant permissions, reduced using a technique called Principal Component Analysis (PCA). The number of components to be selected is determined by the total variance value to be preserved which was set at 80-90%. Following these steps, they obtained a reduced feature vector which was then used as input for seven machine learning algorithms such as K-Nearest Neighbors (KNN), Naive Bayes (NB), Sequential Minimal Optimization (SMO), Multi-Layer Perceptron (MLP), Random Forest (RF), Decision Tree (DT), and Logistic Regression (LR). Following their approach step by step, in the first step, we ranked the three category datasets using PCA. After that, we set the threshold between the range of 80-90% and fed the transformed feature set to the above-mentioned classifiers.

Table 6.10: Compiled detection results to compare the performance of the proposed classifier with other related works across the three dataset categories

Dataset category	**Approach used**	**Number of features used**	**Detection accuracy (in %)**
Static	Anupama et al. [120]	50 permissions	93.93
	Bhat et al. [177]	30 permissions	94.44
	Şahin et al. [178]	68 permissions	95.52
	Our proposed classifier	**10 permissions**	**99.57**
Dynamic	Anupama et al. [120]	68 system calls	86.3
	Bhat et al. [177]	30 system calls	88.78
	Şahin et al. [178]	39 system calls	87.04
	Our proposed classifier	**10 system calls**	**98.9**
Hybrid	Anupama et al. [120]	128 permissions and system calls	96.67
	Bhat et al. [177]	30 permissions and system calls	94.19
	Şahin et al. [178]	100 permissions and system calls	97.15
	Our proposed classifier	**10 permissions and system calls**	**99.93**

Table 6.10 presents the comparative results of our proposed classifier against related works across the three dataset categories. As can be seen from the table, for the static category, when we followed the approach in [120], we managed to achieve a detection accuracy of 93.93% using 50 permissions, following [177], we reported 94.44% with 30 permissions and following [178], we observed the maximum accuracy of 95.52 using 68 permissions. Whereas, our proposed method significantly outperforms these, achieving 99.57% with just 10 permissions. In the dynamic category, applying [120] we attained 86.3% accuracy with 68 system calls, with [177], we achieved 88.78% using 30 system calls, and with [178], we observed the highest detection accuracy of 87.04% considering 39 system calls. Our method surpasses these results with a detection accuracy of 98.9% using only 10 system calls. Similarly, for the hybrid category, while using the methodology in [120], we reported 96.67% accuracy using 128 permissions and system calls, with that of [177] we achieved 94.19% using 30 permissions and system calls and while following [178], we observed the highest detection accuracy of 97.15% considering 100 permissions and system calls. Simultaneously, our classifier demonstrates superior performance, achieving 99.93% accuracy with only 10 permissions and system calls, underscoring its efficiency and effectiveness. Hence, we can conclude that our proposed approach outperforms the similar work of Android malware detection performing feature selection on permissions or system calls.

Furthermore, we compare some other similar works of Android malware detection utilizing hybrid analysis in terms of detection accuracy. Table 6.11 summarizes this comparison. Most studies have attempted to detect malware using conventional ML or DL classifiers, whereas some have ranked the features using tests such as chi-square or information gain. As shown in

the table, our model outperforms all these studies in terms of detection accuracy. Hence, we can conclude that our proposed model is better than many state-of-the-art techniques presented in the literature for Android malware detection.

Table 6.11: Comparison of proposed work with the existing literature based on hybrid malware detection models

Related Work	**Methodology**	**Detection accuracy (in %)**
Surendran et al. [121]	TAN (Tree Augmented naive Bayes) based malware detection model by employing the conditional dependencies features	97
Lu et al. [122]	Hybrid detection model employing Deep belief Network (DBN) and Gated Recurrent Unit (GRU) algorithms	96.82
Zhang et al. [179]	Utilized subgraph isomorphism matching for malware detection using hybrid features	95
Taher et al. [180]	Feature reduction using fuzzy and meta-heuristic optimization techniques followed by using the Harris Hawks Optimization (HHO) algorithm	98.1
Sharma and Agrawal [181]	Adapted a meta-heuristic swarm-based algorithm to reduce the feature set, followed by using Intelligent Water Drop Algorithm (IWD) for detection	99.12
Mahdavifar et al. [182]	Utilized Pseudo-Label Stacked Auto-Encoder (PLSAE) for detection	97.7
Ficco [183]	Ensemble approach combing multiple algorithms and features	93.28
***PattMatch* classifier (our proposed approach)**	**Classifier employing the combination of an *Average Weighted Pattern Score (AWPS)* technique with *Attribute Score-based Ranking (ASR)* for feature selection**	**99.93**

6.6 Conclusion and Future Work

In conclusion, this chapter discussed two hybrid detection models leveraging the merits of both static and dynamic features. First, we introduced *AndroV-Rank*, a robust framework for Android malware detection that integrates permissions and system calls to extract a refined set of class-distinguishing features. By employing the *VIKOR* method for feature ranking, our approach not only enhances detection accuracy but also streamlines the feature set to a mere 65 attributes, resulting in a remarkable accuracy of 96.55%. Additionally, we proposed another novel hybrid detection model for Android malware that even bypasses the use of conventional data mining algorithms. Our approach centered on *PattMatch*, an instance-based pattern-matching classifier that combines *Average Weighted Pattern Scoring (AWPS)* with *Attribute Score-based Ranking (ARS)*. This model effectively identifies the class labels of unlabeled test samples by focusing on a reduced feature set, achieving optimal classification accuracy. Experimental results have demonstrated the superiority of our hybrid detection model. The hybrid model, which integrates both static and dynamic features, achieved an impressive accuracy of 99.93% using just 10 attributes. Unlike traditional data mining, machine learning, and deep learning algorithms, our classifier demonstrated superior detection accuracy on the same dataset, high-

lighting the robustness and efficiency of the proposed hybrid detection model. Additionally, our model surpasses various state-of-the-art Android malware detection techniques in terms of detection accuracy. The performance of both models significantly exceed traditional static and dynamic analysis methods in terms of detection accuracy, underscoring the limitations of these standalone techniques. In our future work, we aim to enhance the capabilities of our model by including malware category and family classification along with the binary classification performed in this study.

Chapter 7

Android Malware Multi-Category Classification via Highly Discriminative Feature Ranking

---

In this chapter, we discuss two models for Android malware multi-category classification into the four categories: *Adware, Fraudware Trojans, Ransomware, and Spyware*, by using the least number of features while simultaneously ensuring higher classification accuracy. In Section 7.1, we explain the motivation behind proposing a Android malware multi-category classification model and a brief overview of the two models proposed. In Section 7.2, we explain in detail the methodology behind our first model. Section 7.3 explains the results obtained from the proposed model - I. In Section 7.4, we explain the methodology behind our second proposed multi-category classification model named *AndroMultiCat*. Section 7.5 discusses the results obtained from the second model and finally, we conclude the chapter in Section 7.6.

7.1 Introduction

Android malware has emerged as a significant threat, compromising user privacy, device functionality, and overall security. The pervasive adoption of Android devices, coupled with their open architecture, makes them particularly susceptible to malicious software. These

threats, collectively referred to as Android malware, not only disrupt device operations but also endanger sensitive user data.

Android malware is broadly classified into categories such as *Adware, Fraudware, Ransomware,* and *Spyware*. Each type poses distinct risks, ranging from financial exploitation to data breaches. Among these, malware that exploits root privileges is of particular concern. Such variants facilitate extortion, as seen in ransomware attacks, or enable persistent access and data theft, exemplified by spyware.

1. ***Adware***: This category displays intrusive advertisements on user devices, often without consent. *Adware* generates revenue for developers by coercing users to view or interact with unwanted ads, thereby degrading the user experience.

2. ***Fraudware Trojans***: *Fraudware*, or *Trojans*, masquerades as legitimate applications to deceive users into installation. Once active, these programs steal sensitive information, send premium SMS messages, or download additional malicious software, causing further harm.

3. ***Ransomware***: *Ransomware* encrypts data or locks devices, demanding payment to restore access. These attacks cause significant distress and financial loss for affected users, making ransomware a particularly insidious threat.

4. ***Spyware***: Operating covertly, *Spyware* captures sensitive user data, including keystrokes, browsing activity, and communications. The collected information is often used for identity theft or financial fraud, amplifying its impact.

The increasing sophistication of Android malware requires robust detection mechanisms to effectively mitigate these threats.

As discussed in Chapter 2, binary classification, which differentiates between benign and malicious samples, serves as an essential initial step. However, multi-category classification, which further identifies the specific type of malware is also an additional step taken up by many researchers using a range of static and dynamic features. Hence, in this chapter, we present two Android malware multi-category detection models aimed at performing multi-category classification of malware into the four categories: *Adware, Fraudware trojans, Ransomware, and Spyware*, by using the least number of features that exhibit higher distinguishing power between different malware categories while simultaneously ensuring higher classification accuracy.

7.2 Proposed Multi-Category Detection model-1

Despite the advantages of static analysis, its limitations have led researchers to explore dynamic analysis as a solution to overcome these challenges. Hence, in this work we propose a dynamic Android malware detection model that leverages system call monitoring to accurately detect and classify malware into the four categories: *Adware, Fraudware trojans, Ransomware*, and *Spyware*, by using the least number of features while simultaneously ensuring higher classification accuracy. To achieve this, we employ the *ReliefF* algorithm as a feature ranking and selection technique, which formulates a *ReliefF* Score that assesses the contribution of each feature to the classification task. This method is particularly suited for categorical datasets, such as ours, as it effectively evaluates the relevance of features in distinguishing between multiple malware categories. By identifying and ranking features based on their ability to differentiate between classes, *ReliefF* enables us to select the most impactful features that enhance classification accuracy while discarding irrelevant or lower-ranked features. This approach not only streamlines the feature set but also improves the overall robustness of our malware category classification model.

The proposed model's methodology is primarily divided into two modules, *Ranking* and *Classification*. Initially, we compile an extensive dataset that includes dynamic features (system calls). Subsequently, we split this dataset into two segments: the *Training* dataset and the *Testing* dataset. In the Ranking module, we begin by inputting the *Training* dataset and ranking the features based on their *ReliefF* scores. In the *Classification* module, we propose a novel algorithm that applies Machine Learning (ML) and Deep Learning (DL) techniques to get the best features that can provide higher classification accuracy.

RANKING MODULE

7.2.1 Data collection

To initiate this research, we employed the *Kronodroid* dataset [170] also used in chapter 6. More details about the dataset have been discussed in 6.2.1. In order to build a dynamic analysis-based Android multi-category classification model, we focused only on dynamically extracted system calls, where call frequency data records non-zero values for occurrences and zero for absences. The dataset categorizes samples into *Adware*, *Fraudware*, *Ransomware*, and *Spyware*. For training, we used 50% of each category (10,000 samples), with the remaining

50% for testing (10,000 samples). This approach aims to leverage dynamic features for a robust multi-category classification of Android malware, addressing the limitations inherent to static analysis. We processed the dataset into *Training* and *Testing*, with each encompassing 289 system calls. Details of the dataset composition are presented in Table 7.1.

Table 7.1: Details of the dataset including malware categories and the families comprising each category.

Malware Category	**Families comprising the category**
Adware	Airpush, Agent, FakeApp, Kuguo, Dowgin and Youmi
Fraudware	Boxer, FakeInst and SMSreg
Ransomware	Slocker
Spyware	DroidKungFu, GinMaster, BankBot, Simhosy and Malap

7.2.2 Feature Ranking

In this study, we applied the *ReliefF* algorithm to rank features based on their relevance for multi-category malware classification. *ReliefF* is a feature selection technique designed to estimate the importance of each feature by assessing how well it distinguishes between instances from different classes. For a given multiclass dataset with n applications and p attributes, where samples belong to m classes, *ReliefF* evaluates each feature's ability to separate samples by iteratively selecting instances and calculating feature weights based on intra- and inter-class distances.

The algorithm operates as follows: for each randomly selected instance i, *ReliefF* finds its nearest neighbors within the same class (*nearest hits*) and from different classes (*nearest misses*). The weight $W(A)$ of each attribute A is updated to reflect its ability to distinguish between classes. The update rule for feature weights is given by:

$$W(A) = W(A) - \frac{1}{k}\sum_{j=1}^{k}\left(\text{diff}(A,i,\text{hit}_j)\right) + \frac{1}{k(m-1)}\sum_{l\neq\text{class}(i)}\sum_{j=1}^{k}P(l)\,\text{diff}(A,i,\text{miss}_{j,l}) \tag{7.2.1}$$

where:

- k is the number of nearest neighbors considered,
- $\text{diff}(A,i,j)$ represents the difference between instances i and j on attribute A,
- $P(l)$ is the prior probability of class l,
- hit_j is the j-th nearest neighbor within the same class as i,
- $\text{miss}_{j,l}$ is the j-th nearest neighbor from a different class l.

Using *ReliefF* with our multiclass dataset of Android malware, the algorithm ranks features, i.e. system calls, by prioritizing those that contribute most to distinguishing between categories namely *Adware*, *Fraudware*, *Ransomware*, and *Spyware*. This ranking allows our model to focus on the most discriminatory features for improved classification accuracy.

7.2.3 Machine Learning and Deep Learning Classifiers

We used several ML and DL algorithms in our classification approach. We applied five widely used techniques, namely Decision Trees (DT), Random Forest (RF), Bagging classifier (BC), Gaussian Naive Bayes (NB) as ML classifiers and Multilayer Perceptron (MLP) as DL classifier. All experiments with these classifiers were performed using ten-fold cross-validation.

Classification Module

7.2.4 Proposed Malware Multi-Category Classification Algorithm

Following the ranking of features using the *ReliefF* algorithm, we obtained a sorted list of system calls based on their effectiveness in distinguishing between various malware categories. Our objective now is to identify the optimal subset of features that maintains the highest classification accuracy while minimizing the number of features used.

We begin by incorporating all available features in the first iteration and recording the classification accuracy achieved by the classifiers utilized in this study. Subsequently, we eliminate the lowest-ranked ten features from the *Testing* dataset and assess the resulting accuracy. In the next iteration, we remove an additional ten features, continuing this process iteratively by excluding the next ten lowest-ranked features in each round. This procedure is repeated until we observe a decline in classification accuracy.

Ultimately, the output will yield the minimal feature set that achieves the best classification performance, thereby optimizing the feature selection for our multi-category malware classification model.

7.3 Results and Discussion: Multi-Category Model - I

In this section, we present the experimental results obtained by applying the proposed malware multi-category classification model to the dynamic category dataset. As previously discussed, we utilized the ranking method on the *Training* dataset to compute the *ReliefF* scores of the system calls. Subsequently, within the *Classification* module, we applied various ML and DL classifiers to the *Testing* dataset.

7.3.1 *ReliefF* Ranking Results

In this section, we present the results of the *ReliefF* ranking applied to the system calls dataset. Table 7.2 displays the top ten system calls, sorted by their effectiveness in distinguishing between malware classes. Notably, the system call *eventfd2* ranks highest in the table, achieving the *ReliefF* score of 0.0920. The rankings of the remaining system calls are detailed in the table. With *getpid* coming out to be the least scored, indicating its relatively low capability to differentiate between malware classes.

Table 7.2: Top 10 system calls ranked in order of their *ReliefF* Score

System Call	***ReliefF* Score**
eventfd2	0.0920
epoll_create1	0.0920
fstatfs64	0.0739
flock	0.0690
readlinkat	0.0687
fchmod	0.0643
ugetrlimit	0.0568
socketpair	0.0555
mkdirat	0.0541
sendmsg	0.0494

7.3.2 Classification Results on the Testing Dataset

In this section, we present the classification results obtained on the *Testing* dataset by iteratively reducing the feature set based on system call rankings given by the *ReliefF* algorithm. Initially, with all 289 system calls included, we achieved an accuracy of 94.01%. In subsequent iterations, system calls were removed in increments of ten, beginning with the least important features, as determined by the *ReliefF* ranking.

With each iteration, the classification accuracy was evaluated to identify the optimal subset of features. By the 23rd iteration, reducing the feature set to the top 70 system calls yielded

the highest observed accuracy of 94.50%. Beyond this point, further reduction in features led to a decline in accuracy. Thus, the optimal subset of features was determined to be 70 system calls, achieving the highest accuracy while reducing the feature set by approximately 75.6%.

Table 7.3 below summarizes the classification accuracy across iterations.

Table 7.3: Classification Accuracy Across Iterations of Feature Reduction

Features used	Accuracy (%)	Features used	Accuracy (%)
50	88.39	170	94.05
60	93.94	180	94.07
70	94.50	190	94.20
80	94.11	200	94.12
90	94.05	210	94.07
100	94.05	220	94.01
110	94.10	230	94.06
120	94.10	240	93.99
130	94.14	250	94.08
140	94.10	260	94.02
150	94.10	270	94.08
160	94.11	280	94.03

7.4 Proposed Multi-Category Detection model - II

Static analysis examines application code without execution, efficiently identifying known malware signatures and patterns but struggles with obfuscated or polymorphic malware. Dynamic analysis executes applications to observe behavior, revealing hidden malicious activities like network communication, yet it is resource-intensive for large-scale screening. Hybrid analysis combines static code examination with dynamic behavioral analysis, offering a balanced approach to enhance detection accuracy and overcome evasion tactics. In this research, we propose a hybrid classification model, integrating static permissions and dynamic system calls, aimed at performing multi-category classification of malware into the four categories: *Adware, Fraudware trojans, Ransomware*, and *Spyware*, by using the least number of features that exhibit higher discriminative power between different malware categories while simultaneously ensuring higher classification accuracy.

To accomplish this, we implement a ranking technique that formulates a *Class Discrimination Strength* vector [184], which is used to sort individual features based on their ability to differentiate between classes. This ranking process enables the selection of the top features that contribute to the best classification accuracy, while irrelevant lower-ranked features are

omitted. Our approach combines permissions from the static feature set and system calls from the dynamic feature set to improve the robustness of malware category classification.

Contributions - The major contributions of this study are highlighted below:

- We designed a hybrid detection model that harnesses the benefits of both static and dynamic analysis, mitigating the shortcomings of each method.
- We employed a ranking technique centering around the *Class Discrimination Strength vector* and a new metric, the *Discrimination Score*, to sort features based on their ability to distinguish between malware classes.
- We proposed a novel algorithm that leverages attribute rankings to develop an efficient Android malware multi-category classification system, achieving higher accuracy with minimal features.

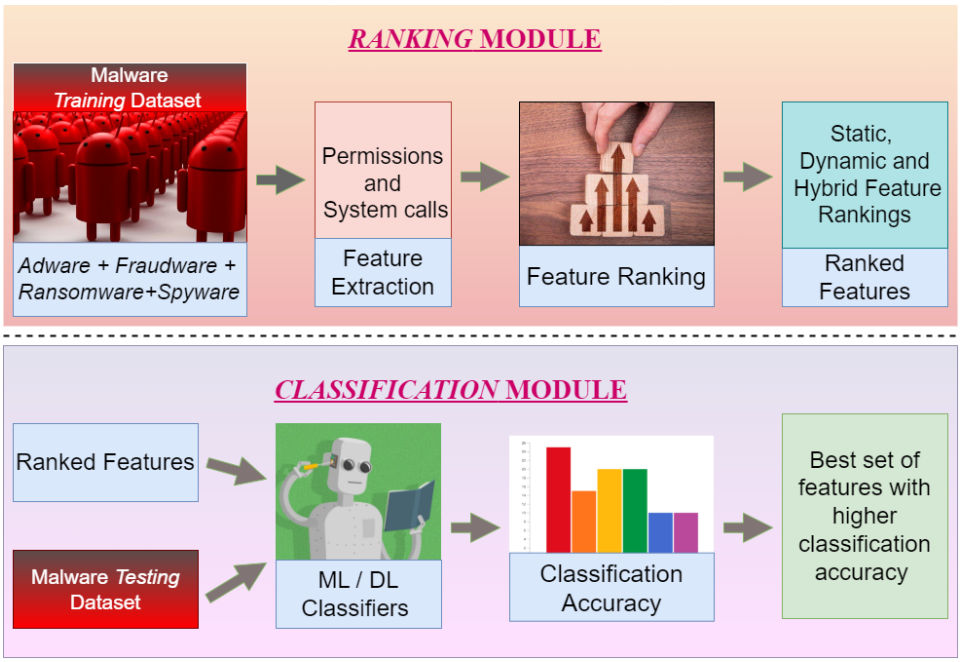


Figure 7.1: *AndroMultiCat* System Design

The proposed model *AndroMultiCat*'s methodology is primarily divided into two modules, *Ranking* and *Classification* module as shown in Figure 7.1. Initially, we compile an extensive dataset that includes static features (permissions) and dynamic features (system calls). Subsequently, we split this dataset into two segments: the *Training* dataset and the *Testing* dataset. In the Ranking module, we begin by inputting the *Training* dataset and ranking the features based on their discrimination strength. This is achieved by calculating the class discrimination

strength vector for each individual feature. In the *Classification* module, we propose a novel algorithm that applies Machine Learning (ML) and Deep Learning (DL) techniques to get the best features that can provide higher classification accuracy.

The subsequent subsections provide a detailed discussion of both modules in the proposed model.

Ranking Module

7.4.1 Dataset Accumulation

To initiate our research, we utilized the *Kronodroid* dataset [170], the largest hybrid-feature Android dataset spanning from 2008 to 2020. This dataset uniquely includes both static features like permissions, intents, and metadata, and dynamic features represented by system calls. Malware samples were sourced from repositories such as *Drebin, AMD, VirusTotal*, and *VirusShare*. The permissions dataset uses a binary format where '1' indicates permission requests, and '0' indicates absence. Similarly, the system call dataset records the frequency of calls with non-zero values, and '0' for absence. The dataset comprises 34,335 rows categorized into *Adware, Fraudware, Ransomware*, and *Spyware*. For training, 70% of each category was selected, leaving 30% for testing, resulting in 24,033 samples for training and 10,302 for testing. After preprocessing, we retained 167 permissions and 289 system calls for analysis.

Our goal is to develop a model leveraging hybrid features, capable of performing Android malware multi-category classification, that addresses the limitations of both static and dynamic analysis models. To enable a comprehensive comparison among the three analysis techniques, we further processed the dataset to create three distinct categories for both the *Testing* and *Training* datasets: static, dynamic, and hybrid. The static category includes datasets with 167 permissions, the dynamic category includes datasets with 289 system calls, and the hybrid category combines both permissions and system calls, totaling 456 features.

Permissions provide an overview, while system calls offer detailed insights. This enhances detection system robustness by scrutinizing suspicious activities comprehensively. Permissions assess app requests pre-installation, identifying potentially malicious apps early. Real-time system call monitoring detects malware activities as they occur, optimizing our hybrid detection model.

7.4.2 Feature Ranking

Assume in the given multi-class *Training* dataset there are n samples or applications, each with p attributes, and these n samples belong to m classes. In the following, we define and employ a vector representation for attributes, which can differentiate their class recognition strength [184].

Let F_{ij} denote whether the j-th attribute is requested by the i-th application. That is, in the application attribute matrix, each column represents an attribute, $F_j = F_{1j}, F_{2j}, \ldots, F_{nj}$, indicating if the attribute is requested (1) or not (0) by each application.

For the j-th feature, its mean request level in the k-th class is denoted as $\bar{F}_{jk}$ for $k = 1, 2, \ldots, m$. The value $|\bar{F}_{jk} - \bar{F}_{jl}|$ captures the difference between the mean request levels of the j-th feature in the k-th class and in the l-th class. Obviously, if this value is small, then the j-th feature would not be effective in discriminating samples from these two classes, but it could be effective otherwise. Therefore, we define the *Class Discrimination Strength vector*, $(\bar{F}_j)$, for the j-th feature as

$$\bar{F}_j = |\bar{F}_{j1} - \bar{F}_{j2}|, |\bar{F}_{j1} - \bar{F}_{j3}|, \ldots, |\bar{F}_{j1} - \bar{F}_{jm}|, |\bar{F}_{j2} - \bar{F}_{j3}|, \ldots, |\bar{F}_{j2} - \bar{F}_{jm}|, \ldots, |\bar{F}_{jm-1} - \bar{F}_{jm}| \quad (7.4.1)$$

After defining the *Class Discrimination Strength vector*, $\bar{F}_j$, we compute the *Discrimination Score* for each attribute by taking the absolute modulus of the vector with each having $\frac{m(m-1)}{2}$ entries. This process ensures that the discrimination power of each attribute is accurately quantified, facilitating the identification of features that are most effective in distinguishing between different classes. In our study, which aims to classify malware into four distinct categories—*Adware, Fraudware, Ransomware*, and *Spyware*—the value of m is defined as 4. With the *Training* dataset comprising a total of 24,033 applications, the value of n is therefore 24,033. For the static category datasets, the value of p is set to 167, corresponding to the total number of permissions in the dataset. For the dynamic category, p is 289, reflecting the number of system calls. In the hybrid category, the value of p is established as 456, representing the sum of both permissions and system calls.

7.4.3 Machine Learning and Deep Learning Classifiers

We used several ML and DL classifiers [135] in our classification approach. We applied nine widely used techniques, namely Decision Trees (DT), Random Forest (RF), Support Vector Machine (SVM), Bagging classifier (BC), Gaussian Naive Bayes (NB), Logistic Regression (LR), as ML classifiers and Multilayer Perceptron (MLP), Artificial Neural Networks (ANN), Dense Neural Network (DNN) as DL classifiers. All experiments with these classifiers were performed using ten-fold cross-validation.

Classification Module

7.4.4 Proposed Malware Multi-Category classification Algorithm

This section outlines our classification algorithm for identifying effective features in malware classification. We start by ranking features from our *Training* dataset based on their *Discrimination Scores*, which indicate how well they differentiate between malware types. Initially, we select the top-ranked feature and evaluate only its classification accuracy using ML and DL methods on the *Testing* dataset. We update the maximum accuracy whenever a higher accuracy is achieved in subsequent iterations. Each iteration involves adding the next highest-ranked feature to our set and assessing classification accuracy with the expanded feature set. We continue this process until no further improvement in accuracy is observed. The algorithm terminates when the classification accuracy no longer exceeds the current maximum. It then outputs the highest achieved accuracy along with the set of features that contributed to it, optimizing our approach to malware classification.

The results derived from the proposed approach are discussed in the subsequent section.

7.5 Results and Discussion: Multi-Category Detection model - II

In this section, we present the experimental results obtained by applying the proposed malware multi-category classification model to the three types of datasets: static, dynamic, and hybrid. As previously discussed, we utilized the ranking method on the *Training* dataset to

compute the discrimination scores of the attributes. Subsequently, within the *Classification* module, we applied various ML and DL classifiers to the *Testing* dataset.

7.5.1 Discrimination Score-based Ranking results

In this subsection, we discuss the rankings obtained using the proposed Class *Discrimination Score*-based ranking method when applied to the three dataset categories. Tables 7.4 and 7.5 illustrate the ranked features and their corresponding *Discrimination Scores*.

As shown in Table 7.4, the top 10 permissions ranked using the proposed method for the static category datasets are presented. The permission *READ_PHONE_STATE* ranks highest with a discrimination score of 6.904831664, indicating its superior capability to distinguish between malware categories. Conversely, the permission *WRITE_VOICEMAIL* is the lowest ranked among the total lot of 167 permissions. The rankings of the other permissions are also depicted in the table. Similarly, the top 10 system calls when the discrimination strength ranking method was applied to the dynamic category *Training* dataset are shown alongside. The system call *sysinfo* resides at the top of the table with the highest discrimination score of 3.431581, while *SYS_369* is the lowest ranked amongst the other 289 attributes.

Table 7.4: Top 10 features ranked according to their Discrimination scores for the static and dynamic category

Static Category		**Dynamic Category**	
Permissions	**Discrimination Score**	**System calls**	**Discrimination Score**
READ_PHONE_STATE	6.904831664	sysinfo	3.431581
INTERNET	6.264180883	getpriority	3.203326
SEND_SMS	6.089840222	SYS_333	3.1282
ACCESS_NETWORK_STATE	5.850882026	setrlimit	3.04046
RECEIVE_SMS	5.431887402	socketpair	3.024955
READ_SMS	4.272869115	getrlimit	2.943412
ACCESS_COARSE_LOCATION	4.141156216	uname	2.881312
ACCESS_WIFI_STATE	3.833775255	setsockopt	2.571311
CALL_PHONE	3.76863352	SYS_312	2.190685
ACCESS_FINE_LOCATION	3.731631844	SYS_339	2.053392

Table 7.5 presents the top 20 permissions and system calls when the proposed ranking method was applied to the hybrid category *Training* dataset. The permission *READ_PHONE_STATE* again ranks highest with a score of 6.904832, while *WRITE_VOICEMAIL* is at the bottom of the list comprising 456 attributes.

Table 7.5: Top 20 features ranked according to their Discrimination scores for the Hybrid category

Permissions and System calls	Discrimination Score
READ_PHONE_STATE	6.904832
INTERNET	6.264181
SEND_SMS	6.08984
ACCESS_NETWORK_STATE	5.850882
RECEIVE_SMS	5.431887
READ_SMS	4.272869
ACCESS_COARSE_LOCATION	4.141156
ACCESS_WIFI_STATE	3.833775
CALL_PHONE	3.768634
ACCESS_FINE_LOCATION	3.731632
sysinfo	3.431581
KILL_BACKGROUND_PROCESSES	3.223883
getpriority	3.203326
SYSTEM_ALERT_WINDOW	3.176393
SYS_333	3.1282
setrlimit	3.04046
socketpair	3.024955
BIND_DEVICE_ADMIN	2.970807
getrlimit	2.943412
nr_permissions	2.925607

7.5.2 Classification results on the Testing dataset

In this subsection, we present the classification results obtained by applying ML and DL algorithms to the *Testing* datasets for three categories analyzed in our study. Figure 7.2 illustrates the highest classification accuracy obtained as the number of features used in the three category datasets is varied. It is important to note that classification accuracies were recorded for the entire set of features. However, the figure only displays the range from five to 130 features to highlight that no significant improvement in classification accuracy was observed beyond a certain number of iterations.

Figure 7.2 illustrates our classification results. Initially, we applied ML and DL classifiers to the static *Testing* dataset, ranking features by the *Discrimination Score*-based ranking method. Starting with the top-ranked feature, we measured classification accuracy for identifying malware categories. As more features were added, accuracy improved. Using the top five permissions yielded 78.58% accuracy with the Bagging Classifier (BC). Accuracy increased with additional permissions, reaching a peak of 94% with 45 permissions using the Random Forest (RF) classifier. Thus, 45 permissions were deemed optimal, achieving 94% accuracy with 27% of the total 167 permissions. Next, for the dynamic *Testing* dataset, ML and DL algorithms were applied to system calls. With 5 system calls, we achieved 56.1% accuracy using the Support Vector Machine (SVM) classifier. Accuracy improved with more system calls, peaking at 90.99% with 115 system calls, using 40% of the total 289 system calls.

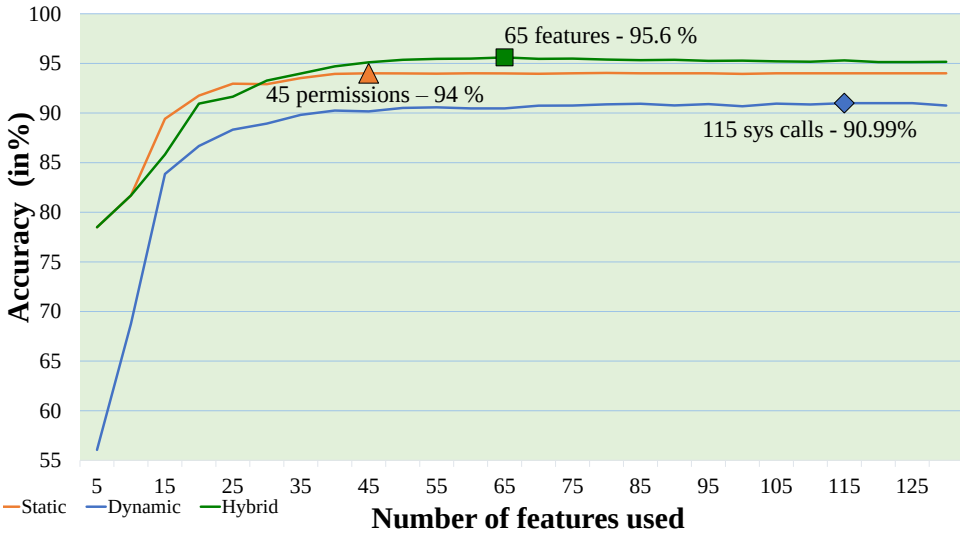


Figure 7.2: Classification results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets

Lastly, we evaluated the hybrid dataset with both permissions and system calls using the *Discrimination Score*-based method. Initially, the top five features yielded 78.58% accuracy with the BC classifier. As more features were added, the highest accuracy reached 95.6% with 65 combined features, utilizing 14% of the total 456 features in our study.

To provide a concise understanding of the classification results, Table 7.6 presents the multi-category classification outcomes using ML and DL classifiers on the three-category *Testing* datasets. The results show that using 45 permissions achieves the highest accuracy of 94% for static analysis, while dynamic analysis with 115 system calls reaches a maximum accuracy of 90.99%.

Table 7.6: Compiled classification results obtained by applying the ML and DL classifiers to the static, dynamic, and hybrid category datasets

Number of features used	**Classfication accuracy using various machine learning and deep learning classifiers (in %)**								
	DT	**RF**	**SVM**	**BC**	**NB**	**LR**	**MLP**	**ANN**	**DNN**
45 permissions	92.86	**94**	85.21	93.47	72.67	86.72	93.15	41.7	41.7
115 system calls	85.67	**90.99**	54.75	89.64	30.49	64.98	71.97	40.66	40.66
65 permissions and system calls	93.52	**95.6**	55.85	94.79	67.35	76.91	84.41	45.64	45.84

These findings highlight that while static and dynamic analyses each have their strengths, they also have limitations when used independently. Static analysis may miss dynamic behaviors, and dynamic analysis can overlook critical static attributes. However, a hybrid approach combining both techniques leverages their strengths. The proposed model achieved a higher accuracy of 95.6% with only 65 features, demonstrating its superior efficacy in accurately classifying the

datasets.

To further demonstrate the applicability and robustness of the proposed hybrid detection framework, an additional experiment was conducted to explore malware family classification. Although the primary focus of this research remains on multi-category classification, this extended analysis aimed to assess whether the proposed model could also capture finer distinctions among malware belonging to different families, such as *Agent, FakeApp, Fakeinst, and Boxer*. Family-level classification represents a more granular task, where malware samples are grouped according to shared traits such as code structure, behavioural signatures, or propagation methods. In practical cybersecurity scenarios, this level of analysis is highly valuable, as it provides deeper insights into the lineage, intent, and operational mechanisms of malicious applications.

While binary and multi-category classification identify whether an application is harmful and the general type of threat it poses, family classification enables a more *context-aware understanding* of malware behaviour. Recognizing that two samples belong to the same family often reveals shared attack strategies, persistence techniques, or common origins, which aids in threat intelligence, forensic analysis, and targeted mitigation. Such information allows security professionals to design more precise countermeasures and track the evolution of malware campaigns over time. However, it must also be acknowledged that the Android malware ecosystem contains a vast and continuously expanding set of families, making exhaustive family-level detection an extensive and challenging research problem.

In this study, a preliminary attempt was made to classify a limited set of eight representative malware families—namely *Airpush, Agent, FakeApp, Kuguo, Dowgin, Youmi, Fakeinst, and Boxer*. The experiment employed the same testing dataset and the top 65 ranked hybrid features used in the primary classification framework. The proposed model achieved a **maximum accuracy of 86%** in distinguishing between these families, indicating that the selected hybrid features retained strong discriminative capability even at a finer level of granularity. Although this experiment does not represent comprehensive family classification, it effectively demonstrates the scalability and adaptability of the proposed hybrid approach. Future research may expand on this direction by incorporating larger and more diverse malware family datasets, thereby further enhancing the practical relevance and generalization ability of the framework in real-world Android malware detection environments.

7.6 Conclusion and Future Work

In conclusion, our research highlights the effectiveness of an Android malware multi-category classification model centering on Discrimination Score-based ranking that integrates static and dynamic features for superior classification accuracy. Static analysis achieved 94% accuracy using 45 permissions (27% of 167 permissions), while dynamic analysis reached 90.99% with 115 system calls (40% of 289 system calls). However, our hybrid model outperformed both, achieving 95.6% accuracy with just 65 features (14% of 456 permissions and system calls). This significant reduction in features while improving accuracy underscores the advantage of combining static and dynamic methods. Introducing the *Class Discrimination Strength vector* and *Discrimination Score*-based ranking methods have been crucial in enhancing feature selection and model efficiency. Our findings advocate strongly for hybrid models in malware detection to combat evolving cyber threats. Future work will focus on expanding our model to include comprehensive malware family classification alongside multi-category classification.

Chapter 8

Conclusion, Future Scope, and Social Impact

Smartphones have surpassed desktop systems in popularity due to their feature-rich applications, offering a wide range of services from online shopping and gaming to location-based functionalities. They have become integral to daily life, often considered more powerful than early personal computers. However, the increasing reliance on smartphones has led to a significant surge in malware attacks, particularly targeting Android devices. Malicious applications can infiltrate smartphones through SMS, MMS, Bluetooth, internet downloads, or app stores, including both official and third-party platforms. These attacks pose serious risks such as system damage, financial loss, and data breaches. Consequently, Android malware detection has garnered substantial attention within the research community in recent years, driven by the escalating frequency of attacks.

This chapter concludes the thesis by summarizing its key contributions, reviewing the proposed models for Android malware detection, and demonstrating their alignment with the established objectives. Additionally, it highlights several open challenges in the literature, underscoring critical areas that require further investigation in future research. Finally, it concludes with a subsection on the social impact of this work, emphasizing its significance in fostering a safer digital environment in today's malware-prone world."

8.1 Conclusion

In this section, we summarize the findings and contributions made in this thesis:

1. Features in the *AndroidManifest* file, such as permissions, intents, and hardware components, often overlap between benign and malicious applications. For instance, the *INTERNET* permission is widely used across both classes, making it difficult to distinguish malicious behavior. To address this challenge, Chapter 3 introduced *PHIGrader*, a system that ranks and evaluates static features—permissions, intents, and hardware components—using frequency-based *Multi-Criteria Decision-Making (MCDM)* techniques, including *TOPSIS*, *EDAS*, and *WASPAS*. First, features are ranked to identify the most discriminative ones. Leveraging these ranked features with machine learning and deep learning classifiers, *PHIGrader* achieved optimal detection performance. Notably, selecting the top 46 features via *TOPSIS* yielded a detection accuracy of 99.10%, outperforming models based on single feature types or other *MCDM* methods. Building on the idea of combining feature strengths to mitigate individual limitations, Chapter 4 introduced *PHIAnalyzer*, which explores seven distinct combinations of the three feature types to identify the most effective subset. The proposed model employs a frequency-based *Chi-square* ranking test followed by a novel detection algorithm. *PHIAnalyzer* achieved 98.49% accuracy using only 12 features—a balanced combination of six permissions and six intents—demonstrating better accuracy and efficiency compared to state-of-the-art methods.

   Despite their success, both approaches were limited by static analysis, which struggles with the inability to capture runtime behavior. To overcome these drawbacks, the subsequent chapter proposed a dynamic, network traffic-based detection mechanism. This approach offers deeper behavioral insights, addressing the limitations of static analysis and improving malware detection.

2. Analyzing network traffic usage patterns has proven to be an effective approach for detecting malware, making network traffic flows a key resource in Android malware detection. However, significant similarities exist in traffic feature patterns between benign and malicious applications. Chapter 5 introduced a robust and efficient Android malware detection system based on dynamic analysis, leveraging critical features derived from the TCP flows of application network data. To address feature-class and feature-feature correlations, we rank features using the statistical measure *crRelevance* and reduce redundancy through *Normalized Mean Residue Similarity (NMRS)*. The experimental results highlighted that our *NMRS*-based detection algorithm, applied to *crRelevance* rankings, effectively reduced the feature set while achieving 99.50% accuracy by considering two

network traffic features: *Packet_size_received* and *Time_interval_between_packets_received*. Moreover, the proposed algorithm outperforms various state-of-the-art Android malware detection techniques. While dynamic analysis addresses many limitations of static analysis, it introduces challenges of itself too. For instance, not all malware samples generate detectable network traffic, as some discreetly transmit data in the background.

Hence, we can say both static and dynamic analyses have demonstrated significant merits, as evidenced by their widespread adoption among researchers and practitioners for Android malware detection. However, each method, when used in isolation, suffers from inherent limitations. In order to integrate the strengths of both methods, the next chapter proposed a hybrid detection approach.

3. Chapter 6 emphasized the importance of hybrid detection models by introducing two hybrid analysis-based methods. Firstly, we introduced *AndroV-Rank*, a novel Android malware detection framework leveraging the *VIKOR MCDM* approach based on frequency, attributes, and criteria. This approach ranked static permissions and dynamically extracted system calls to identify optimal class-distinguishing features for enhanced detection accuracy. The proposed hybrid algorithm, combining machine learning (ML) and deep learning (DL) techniques, further optimized feature selection. Experimental results demonstrated a detection accuracy of 96.55% using only 65 features, reducing the feature set to approximately 24.9% of the original size. In our prior works, we employed various machine and deep learning algorithms, widely adopted in Android malware detection for their advantages. However, traditional methods often face challenges, such as being hindered by computational demands despite their higher accuracy. Moreover, for machine learning models like Decision Trees, SVM, and Naive Bayes, imbalanced data—where one class significantly outweighs others—can lead to the underrepresentation of minority classes.

   These limitations in static and dynamic analysis, coupled with challenges in ML-based approaches, motivated the development of a simple yet robust hybrid detection model. Hence, we further introduced *PattMatch*, an instance-based pattern-matching classifier utilizing *Average Weighted Pattern Scoring* (AWPS) and *Attribute Score-based Ranking* (ASR). This model predicted class labels by matching test sample patterns with training patterns, significantly reducing the feature set to the most relevant attributes. Experimental results validated its performance, achieving 99.93% accuracy with only 10 attributes. Unlike conventional ML and DL classifiers, *PattMatch* demonstrated superior detection

accuracy, underscoring the robustness and efficiency of the proposed hybrid framework. Binary classification is crucial for distinguishing Android apps as benign or malicious, laying the foundation for malware detection. However, modern malware, including *Adware, Fraudware Trojans, Spyware, and Ransomware*, varies widely in signatures and attack methods. Researchers have moved beyond binary classification to explore multi-category classification to address these complexities. The next chapter focuses on this advanced approach to categorize malware types more effectively.

4. Chapter 7 presented two models developed for the multicategory classification of Android malware into four distinct categories: *Adware, Fraudware Trojans, Ransomware, and Spyware*. These models focused on achieving high classification accuracy while reducing the feature set size. The first model, leveraging dynamic analysis, utilized system calls ranked through the *ReliefF* algorithm. Experimental results demonstrated that the optimal feature subset included 70 system calls out of 289, yielding an accuracy of 94.50% and reducing the feature set by approximately 75.6%.

   In contrast, the second model, *AndroMultiCat*, employed a hybrid analysis approach by integrating static permissions with dynamic system calls for improved performance. The model prioritized features based on a Discrimination Score-based ranking. Individually, static analysis achieved 94% accuracy with 45 permissions (27% of 167), and dynamic analysis achieved 90.99% accuracy with 115 system calls (40% of 289). However, the hybrid model surpassed both, attaining 95.6% accuracy with just 65 features—comprising 14% of the combined 456 permissions and system calls. This highlights the efficacy of hybrid analysis in optimizing feature usage while enhancing detection accuracy.

8.2 Future Scope

Recent and stealthier Android Malware still poses many challenges for the research community that need to be further studied. This section identifies some future research scope where research is needed.

1. In this thesis, we primarily focused on classification tasks, including a decent attempt at multi-category classification of Android malware. However, malware also exists in various families, each with distinct characteristics and attack patterns, such as the *DroidKungFu, DroidDream*, and *BankBot* families. These families represent groups of malware

with shared behaviors or origins, making their detection and accurate classification crucial for devising targeted defense mechanisms. In our future work, we aim to expand the capabilities of our model by incorporating malware family classification ([140], [141], [142]) alongside the binary and multi-category classification explored in this thesis . This enhancement will enable more comprehensive malware analysis and facilitate better understanding of familial traits, thereby improving detection and mitigation strategies.

2. At the current stage, the proposed detection frameworks have been developed and evaluated as a standalone model rather than a fully deployed Android application or web-based system. Hence, the present work primarily focuses on classification accuracy and feature optimization, without extending the analysis to real-time operational aspects such as processing latency, memory footprint, or energy consumption. These factors are crucial for practical deployment, as excessive processing time, memory usage, or battery drain can reduce system performance and user acceptance.

   Future research will therefore aim to integrate the proposed model within an actual Android environment or emulator to quantitatively evaluate these metrics under real device conditions. Assessing processing speed will ensure responsiveness during detection, memory evaluation will confirm compatibility with resource-constrained devices, and energy analysis will determine long-term usability. Such evaluation, combined with real-world testing on physical or emulated devices, will strengthen the framework's practical relevance and ensure that it performs efficiently not only in offline analysis but also in real-world Android ecosystems.

3. Another important direction for future research involves evaluating the robustness of the proposed detection framework against adversarial malware and evasion techniques. Adversarial malware refers to malicious applications that are intentionally modified to deceive machine learning classifiers by altering features such as permissions, API calls, or network behaviors without changing their harmful intent. Similarly, evasion techniques enable malware to disguise its behavior or structure to avoid detection, using methods such as code obfuscation, dynamic code loading, or mimicry of benign applications. These techniques pose significant challenges to static, dynamic, and hybrid detection models, as they can effectively bypass conventional analysis mechanisms. Investigating how the proposed model performs when exposed to such adversarial or evasive samples

will be essential for assessing its real-world resilience. Incorporating adversarial training and robust evaluation strategies in future work can therefore enhance the framework's capability to withstand evolving Android malware that deliberately adapts to evade detection.

4. The models proposed in this thesis, in their current form, do not address the detection of colluding apps. Colluding apps refer to a group of malicious applications that communicate covertly to share permissions or data, bypassing security measures to execute malicious activities collectively. As this poses a significant threat to Android security, future work will focus on extending our models to identify and mitigate such collusion by incorporating inter-app communication patterns and advanced contextual analysis techniques.

5. The research community typically identifies Android malware only after it has emerged in the market and infected numerous mobile devices globally. To address this, detection mechanisms must be designed to identify stealthier malware as soon as it enters official or third-party app stores. Existing anti-malware solutions, including Google's Bouncer, have consistently struggled to mitigate new or zero-day malicious applications. Therefore, robust and proactive solutions are essential to detect Android malware before it compromises mobile devices.

8.3 Social Impact of the proposed research

The rapid adoption of Android as the dominant mobile operating system has revolutionized connectivity, communication, and productivity. However, this ubiquity has also made Android devices a prime target for malware developers, leading to an alarming rise in malicious applications. The increasing dependency on Android devices for sensitive activities, such as financial transactions, health monitoring, and personal communication, amplifies the societal risks posed by malware. Our work addresses this pressing challenge by developing advanced detection models that improve malware identification and safeguard users' digital lives.

The hybrid detection models proposed in this research combine static and dynamic analysis techniques, ensuring comprehensive coverage against a diverse range of malware. By leveraging cutting-edge algorithms and feature-ranking methods, these models enhance detection accuracy, enabling early identification and prevention of malware attacks. This contributes

not only to the protection of individual devices but also strengthens the broader cybersecurity infrastructure, which is critical in today's interconnected digital ecosystem.

The societal relevance of this work extends beyond technical advancements, as it promotes user trust in digital technologies. By mitigating threats posed by malicious entities, this research helps ensure the continuity of secure and efficient mobile applications, benefiting businesses, institutions, and individuals alike. In a world increasingly reliant on mobile devices, these contributions are pivotal in fostering a safer digital environment, protecting personal privacy, and enabling equitable access to technology without fear of exploitation.

References

[1] Tom Farley. Mobile telephone history. *Privateline. com, http://www. privateline. com/wp-content/uploads/2016/01/TelenorPage_022-034. pdf*, 2005.

[2] Muhammad Sarwar and Tariq Rahim Soomro. Impact of smartphone's on society. *European journal of scientific research*, 98(2):216–226, 2013.

[3] John Callaham. The history of android: The evolution of the biggest mobile os in the world. *Android Authority*, 2021.

[4] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998–1022, 2014.

[5] Sancheng Peng, Shui Yu, and Aimin Yang. Smartphone malware and its propagation modeling: A survey. *IEEE Communications Surveys & Tutorials*, 16(2):925–941, 2013.

[6] Jon Oberheide and Charlie Miller. Dissecting the android bouncer. *SummerCon2012, New York*, 95:110, 2012.

[7] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

[8] Jonathan J Davis and Andrew J Clark. Data preprocessing for anomaly based network intrusion detection: A review. *computers & security*, 30(6-7):353–375, 2011.

[9] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 70:238–254, 2017.

[10] Nir Nissim, Robert Moskovitch, Oren BarAd, Lior Rokach, and Yuval Elovici. Aldroid: efficient update of android anti-virus software using designated active learning methods. *Knowledge and Information Systems*, 49:795–833, 2016.

[11] Alejandro Martín, Raúl Lara-Cabrera, and David Camacho. Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142, 2019.

[12] Tao Peng, Bochao Hu, Junping Liu, Junjie Huang, Zili Zhang, Ruhan He, and Xinrong Hu. A lightweight multi-source fast android malware detection model. *Applied Sciences*, 12(11):5394, 2022.

[13] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided android malware classification. *Computers & Electrical Engineering*, 61:266–274, 2017.

[14] Abdirashid Ahmed Sahal, Shahid Alam, and Ibrahim Soğukpinar. Mining and detection of android malware based on permissions. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 264–268. IEEE, 2018.

[15] Durmuş Özkan Şahın, Sedat Akleylek, and Erdal Kiliç. Linregdroid: Detection of android malware using multiple linear regression models-based classifiers. *IEEE Access*, 10:14246–14259, 2022.

[16] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[17] Durmuş Özkan Şahın, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kiliç. New results on permission based static analysis for android malware. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–4. IEEE, 2018.

[18] Kabakus Abdullah Talha, Dogru Ibrahim Alper, and Cetin Aydin. Apk auditor: Permission-based android malware detection system. *Digital Investigation*, 13:1–14, 2015.

[19] Arvind Mahindru and Paramvir Singh. Dynamic permissions based android malware detection using machine learning techniques. In *Proceedings of the 10th innovations in software engineering conference*, pages 202–210, 2017.

[20] İbrahim Alper Doğru and Murat Önder. Appperm analyzer: malware detection system based on android permissions and permission groups. *International Journal of Software Engineering and Knowledge Engineering*, 30(03):427–450, 2020.

[21] Fengjun Shang, Yalin Li, Xiaolin Deng, and Dexiang He. Android malware detection method based on naive bayes and permission correlation algorithm. *Cluster Computing*, 21(1):955–966, 2018.

[22] Franklin Tchakounté, A Djakene Wandala, and Yélémou Tiguiane. Detection of android malware based on sequence alignment of permissions. *Int. J. Comput.(IJC)*, 35(1):26–36, 2019.

[23] Seung-hwan Ju, Hee-suk Seo, and Jin Kwak. Research on android malware permission pattern using permission monitoring system. *Multimedia Tools and Applications*, 75:14807–14817, 2016.

[24] Soussi Ilham, Ghadi Abderrahim, and Boudhir Anouar Abdelhakim. Permission based malware detection in android devices. In *Proceedings of the 3rd International Conference on Smart City Applications*, pages 1–6, 2018.

[25] Gianni D'Angelo, Francesco Palmieri, and Antonio Robustelli. A federated approach to android malware classification through perm-maps. *Cluster Computing*, 25(4):2487–2500, 2022.

[26] Ping Xiong, Xiaofeng Wang, Wenjia Niu, Tianqing Zhu, and Gang Li. Android malware detection with contrasting permission patterns. *China Communications*, 11(8):1–14, 2014.

[27] Tianliang Lu and Su Hou. A two-layered malware detection model based on permission for android. In *2018 IEEE International Conference on Computer and Communication Engineering Technology (CCET)*, pages 239–243. IEEE, 2018.

[28] K Kavitha, P Salini, and V Ilamathy. Exploring the malicious android applications and reducing risk using static analysis. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 1316–1319. IEEE, 2016.

[29] Eslam Amer. Permission-based approach for android malware analysis through ensemble-based voting model. In *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, pages 135–139. IEEE, 2021.

[30] Sujata Chakravarty et al. Feature selection and evaluation of permission-based android malware detection. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, pages 795–799. IEEE, 2020.

[31] P Sirisha, T Anuradha, et al. Detection of permission driven malware in android using deep learning techniques. In *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 941–945. IEEE, 2019.

[32] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252, 2012.

[33] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. {WHYPER}: Towards automating risk assessment of mobile applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 527–542, 2013.

[34] Aiman A Abu Samra, Kangbin Yim, and Osama A Ghanem. Analysis of clustering technique in android malware detection. In *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 729–733. IEEE, 2013.

[35] Win Zaw Zarni Aung. Permission-based android malware detection. *International Journal of Scientific & Technology Research*, 2(3):228–234, 2013.

[36] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. Puma: Permission usage to detect malware in android. In *International joint conference CISIS'12-ICEUTE 12-SOCO 12 special sessions*, pages 289–298. Springer, 2013.

[37] Veelasha Moonsamy, Jia Rong, and Shaowu Liu. Mining permission patterns for contrasting clean and malicious android applications. *Future Generation Computer Systems*, 36:122–132, 2014.

[38] Michael Backes, Sebastian Gerling, Christian Hammer, Matteo Maffei, and Philipp von Styp-Rekowsky. Appguard–enforcing user requirements on android apps. In *Tools and Algorithms for the Construction and Analysis of Systems: 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 19*, pages 543–548. Springer, 2013.

[39] Hiroya Kato, Takahiro Sasaki, and Iwao Sasase. Android malware detection based on composition ratio of permission pairs. *IEEE Access*, 9:130006–130019, 2021.

[40] Suleiman Y Yerima, Sakir Sezer, and Gavin McWilliams. Analysis of bayesian classification-based approaches for android malware detection. *IET Information Security*, 8(1):25–36, 2014.

[41] Yousef Seyfari and Akbar Meimandi. A new approach to android malware detection using fuzzy logic-based simulated annealing and feature selection. *Multimedia Tools and Applications*, pages 1–25, 2023.

[42] Arvind Mahindru and AL Sangal. Fsdroid:-a feature selection technique to detect malware from android using machine learning techniques: Fsdroid. *Multimedia Tools and Applications*, 80: 13271–13323, 2021.

[43] Maham Chaudhary and Ammar Masood. Realmalsol: real-time optimized model for android malware detection using efficient neural networks and model quantization. *Neural Computing and Applications*, 35(15):11373–11388, 2023.

[44] Hashida Haidros Rahima Manzil and S Manohar Naik. Android ransomware detection using a novel hamming distance based feature selection. *Journal of Computer Virology and Hacking Techniques*, pages 1–23, 2023.

[45] Hongpeng Bai, Nannan Xie, Xiaoqiang Di, and Qing Ye. Famd: A fast multifeature android malware detection framework, design, and implementation. *IEEE Access*, 8:194729–194740, 2020.

[46] Moutaz Alazab, Mamoun Alazab, Andrii Shalaginov, Abdelwadood Mesleh, and Albara Awajan. Intelligent mobile malware detection using permission requests and api calls. *Future Generation Computer Systems*, 107:509–521, 2020.

[47] Parnika Bhat and Kamlesh Dutta. A multi-tiered feature selection model for android malware detection based on feature discrimination and information gain. *Journal of King Saud University-Computer and Information Sciences*, 34(10):9464–9477, 2022.

[48] Jun Song, Chunling Han, Kaixin Wang, Jian Zhao, Rajiv Ranjan, and Lizhe Wang. An integrated static detection and analysis framework for android. *Pervasive and Mobile Computing*, 32:15–25, 2016.

[49] Diyana Tehrany Dehkordy and Abbas Rasoolzadegan. A new machine learning-based method for android malware detection on imbalanced dataset. *Multimedia Tools and Applications*, 80: 24533–24554, 2021.

[50] Duc V Nguyen, Giang L Nguyen, Thang T Nguyen, Anh H Ngo, and Giang T Pham. Minad: Multi-inputs neural network based on application structure for android malware detection. *Peer-to-Peer Networking and Applications*, pages 1–15, 2022.

[51] Ahmad Firdaus, Nor Badrul Anuar, Mohd Faizal Ab Razak, and Arun Kumar Sangaiah. Bio-inspired computational paradigm for feature investigation and malware detection: interactive analytics. *Multimedia Tools and Applications*, 77:17519–17555, 2018.

[52] MV Varsha, P Vinod, and KA Dhanya. Identification of malicious android app using manifest and opcode features. *Journal of Computer Virology and Hacking Techniques*, 13:125–138, 2017.

[53] Lichao Sun, Xiaokai Wei, Jiawei Zhang, Lifang He, S Yu Philip, and Witawas Srisa-an. Contaminant removal for android malware detection systems. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1053–1062. IEEE, 2017.

[54] Hemant Rathore, Adarsh Nandanwar, Sanjay K Sahay, and Mohit Sewak. Adversarial superiority in android malware detection: Lessons from reinforcement learning based evasion attacks and defenses. *Forensic Science International: Digital Investigation*, 44:301511, 2023.

[55] Mothanna Almahmoud, Dalia Alzu'bi, and Qussai Yaseen. Redroiddet: android malware detection based on recurrent neural network. *Procedia Computer Science*, 184:841–846, 2021.

[56] Arvind Mahindru and AL Sangal. Somdroid: Android malware detection by artificial neural network trained using unsupervised learning. *Evolutionary Intelligence*, 15(1):407–437, 2022.

[57] Rahim Taheri, Meysam Ghahramani, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, and Mauro Conti. Similarity-based android malware detection using hamming distance of static binary features. *Future Generation Computer Systems*, 105:230–247, 2020.

[58] Seif ElDein Mohamed, Mostafa Ashaf, Amr Ehab, Omar Shereef, Haytham Metwaie, and Eslam Amer. Detecting malicious android applications based on api calls and permissions using machine learning algorithms. In *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, pages 1–6. IEEE, 2021.

[59] Shina Sheen, R Anitha, and V Natarajan. Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing*, 151:905–912, 2015.

[60] Arvind Mahindru and AL Sangal. Mldroid—framework for android malware detection using machine learning techniques. *Neural Computing and Applications*, 33(10):5183–5240, 2021.

[61] Rahim Taheri, Reza Javidan, and Zahra Pooranian. Adversarial android malware detection for mobile multimedia applications in iot environments. *Multimedia Tools and Applications*, 80: 16713–16729, 2021.

[62] Arvind Mahindru and AL Sangal. Semidroid: a behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches. *International Journal of Machine Learning and Cybernetics*, 12:1369–1411, 2021.

[63] Nannan Xie, Xing Wang, Wei Wang, and Jiqiang Liu. Fingerprinting android malware families. *Frontiers of Computer Science*, 13:637–646, 2019.

[64] Mohammed N AlJarrah, Qussai M Yaseen, and Ahmad M Mustafa. A context-aware android malware detection approach using machine learning. *Information*, 13(12):563, 2022.

[65] PC Senthil Mahesh and S Hemalatha. An efficient android malware detection using adaptive red fox optimization based cnn. *Wireless Personal Communications*, 126(1):679–700, 2022.

[66] Mohammad Reza Keyvanpour, Mehrnoush Barani Shirzad, and Farideh Heydarian. Android malware detection applying feature selection techniques and machine learning. *Multimedia Tools and Applications*, 82(6):9517–9531, 2023.

[67] Arvind Mahindru and AL Sangal. Hybridroid: an empirical analysis on effective malware detection model developed using ensemble methods. *The Journal of Supercomputing*, 77:8209–8251, 2021.

[68] Junmei Sun, Kai Yan, Xuejiao Liu, Chunlei Yang, and Yaoyin Fu. Malware detection on android smartphones using keywords vector and svm. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 833–838. IEEE, 2017.

[69] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[70] Hui-juan Zhu, Wei Gu, Liang-min Wang, Zhi-cheng Xu, and Victor S Sheng. Android malware detection based on multi-head squeeze-and-excitation residual network. *Expert Systems with Applications*, 212:118705, 2023.

[71] Mülhem İbrahim, Bayan Issa, and Muhammed Basheer Jasser. A method for automatic android malware detection based on static analysis and deep learning. *IEEE Access*, 10:117334–117352, 2022.

[72] Abdullah Talha Kabakus. Droidmalwaredetector: A novel android malware detection framework based on convolutional neural network. *Expert Systems with Applications*, 206:117833, 2022.

[73] Wei Yuan, Yuan Jiang, Heng Li, and Minghui Cai. A lightweight on-device detection method for android malware. *IEEE transactions on systems, man, and cybernetics: systems*, 51(9): 5600–5611, 2019.

[74] Wenbo Fang, Junjiang He, Wenshan Li, Xiaolong Lan, Yang Chen, Tao Li, Jiwu Huang, and Linlin Zhang. Comprehensive android malware detection based on federated learning architecture. *IEEE Transactions on Information Forensics and Security*, 18:3977–3990, 2023.

[75] Cuiying Gao, Minghui Cai, Shuijun Yin, Gaozhun Huang, Heng Li, Wei Yuan, and Xiapu Luo. Obfuscation-resilient android malware analysis based on complementary features. *IEEE Transactions on Information Forensics and Security*, 18:5056–5068, 2023. doi: 10.1109/TIFS.2023.3302509.

[76] Harshit Kumar, Biswadeep Chakraborty, Sudarshan Sharma, and Saibal Mukhopadhyay. Xmd: An expansive hardware-telemetry-based mobile malware detector for endpoint detection. *IEEE Transactions on Information Forensics and Security*, 18:5906–5919, 2023. doi: 10.1109/TIFS.2023.3318969.

[77] Ankit Kumar Jain, Ninmoy Debnath, and Arvind Kumar Jain. Apuml: An efficient approach to detect mobile phishing webpages using machine learning. *Wireless Personal Communications*, 125(4):3227–3248, 2022.

[78] Marko Dimjašević, Simone Atzeni, Ivo Ugrina, and Zvonimir Rakamaric. Evaluation of android malware detection based on system calls. In *Proceedings of the 2016 ACM on international workshop on security and privacy analytics*, pages 1–8, 2016.

[79] D Kumar, G Radhamani, P Vinod, M Shojafar, N Kumar, and M Conti. Identification of android malware using refined system calls. *Concurr. Comput. Pract. Exp*, 31:e5311, 2019.

[80] Christopher Jun Wen Chew, Vimal Kumar, Panos Patros, and Robi Malik. Real-time system call-based ransomware detection. *International Journal of Information Security*, pages 1–20, 2024.

[81] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6:30996–31011, 2018.

[82] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[83] Vaibhav Rastogi, Yan Chen, and William Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220, 2013.

[84] Mingshen Sun, Tao Wei, and John CS Lui. Taintart: A practical multi-level information-flow tracking system for android runtime. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 331–342, 2016.

[85] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054, 2013.

[86] Alessandro Reina, Aristide Fattori, and Lorenzo Cavallaro. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. *EuroSec, April*, 2013.

[87] Vitor Monte Afonso, Matheus Favero de Amorim, André Ricardo Abed Grégio, Glauco Barroso Junquera, and Paulo Lício de Geus. Identifying android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques*, 11:9–17, 2015.

[88] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26, 2011.

[89] Min Zheng, Mingshen Sun, and John CS Lui. Droidtrace: A ptrace based android dynamic analysis system with forward execution capability. In *2014 international wireless communications and mobile computing conference (IWCMC)*, pages 128–133. IEEE, 2014.

[90] Mario Almeida, Muhammad Bilal, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Matteo Varvello, and Jeremy Blackburn. Chimp: Crowdsourcing human inputs for mobile phones. In *Proceedings of the 2018 World Wide Web Conference*, pages 45–54, 2018.

[91] Jae-wook Jang, Hyunjae Kang, Jiyoung Woo, Aziz Mohaisen, and Huy Kang Kim. Andro-dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information. *computers & security*, 58:125–138, 2016.

[92] Anshul Arora and Sateesh K Peddoju. Minimizing network traffic features for android mobile malware detection. In *Proceedings of the 18th international conference on distributed computing and networking*, pages 1–10, 2017.

[93] Anshul Arora, Shree Garg, and Sateesh K Peddoju. Malware detection using network traffic analysis in android based mobile devices. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 66–71. IEEE, 2014.

[94] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual conference on privacy, security and trust (PST)*, pages 233–23309. IEEE, 2017.

[95] Asaf Shabtai, Lena Tenenboim-Chekina, Dudu Mimran, Lior Rokach, Bracha Shapira, and Yuval Elovici. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43:1–18, 2014.

[96] Shanshan Wang, Qiben Yan, Zhenxiang Chen, Bo Yang, Chuan Zhao, and Mauro Conti. Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security*, 13(5):1096–1109, 2017.

[97] Wenhao Li, Xiao-Yu Zhang, Huaifeng Bao, Qiang Wang, Haichao Shi, and Zhaoxuan Li. A glimpse of the whole: Detecting few-shot android malware encrypted network traffic. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud*

& Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), pages 635–644, 2022. doi: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00111.

[98] Md. Sakir Hossain, Naim Hasan, Md. Abdus Samad, Hossain Md. Shakhawat, Joydeep Karmoker, Foysol Ahmed, K. F. M. Nafiz Fuad, and Kwonhue Choi. Android ransomware detection from traffic analysis using metaheuristic feature selection. *IEEE Access*, 10:128754–128763, 2022. doi: 10.1109/ACCESS.2022.3227579.

[99] Tingting Lu and Junfeng Wang. F2dc: Android malware classification based on raw traffic and neural networks. *Computer Networks*, 217:109320, 2022.

[100] Hongbo Han, Zhenxiang Chen, Qiben Yan, Lizhi Peng, and Lei Zhang. A real-time android malware detection system based on network traffic analysis. In *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15*, pages 504–516. Springer, 2015.

[101] Aqil Zulkifli, Isredza Rahmi A Hamid, Wahidah Md Shah, and Zubaile Abdullah. Android malware detection based on network traffic using decision tree algorithm. In *Recent Advances on Soft Computing and Data Mining: Proceedings of the Third International Conference on Soft Computing and Data Mining (SCDM 2018), Johor, Malaysia, February 06-07, 2018*, pages 485–494. Springer, 2018.

[102] Ying Pang, Zhenxiang Chen, Xiaomei Li, Shanshan Wang, Chuan Zhao, Lin Wang, Ke Ji, and Zicong Li. Finding android malware trace from highly imbalanced network traffic. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, volume 1, pages 588–595. IEEE, 2017.

[103] Jyoti Malik and Rishabh Kaushal. Credroid: Android malware detection by network traffic analysis. In *Proceedings of the 1st acm workshop on privacy-aware mobile computing*, pages 28–36, 2016.

[104] Shanshan Wang, Zhenxiang Chen, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. A mobile malware detection method using behavior features in network traffic. *Journal of Network and Computer Applications*, 133:15–25, 2019.

[105] Rong Chen, Yangyang Li, and Weiwei Fang. Android malware identification based on traffic analysis. In *International conference on artificial intelligence and security*, pages 293–303. Springer, 2019.

[106] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A. Ghorbani. Dynamic android malware category classification using semi-supervised deep learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*, pages 515–522, 2020.

[107] Tianyue Liu, Zhenwan Li, Haixia Long, and Anas Bilal. Nt-gnn: Network traffic graph for 5g mobile iot android malware detection. *Electronics*, 12(4):789, 2023.

[108] Djallel Hamouda, Mohamed Amine Ferrag, Nadjette Benhamida, Zine Eddine Kouahla, and Hamid Seridi. Android malware detection based on network analysis and federated learning. In *Cyber Malware: Offensive and Defensive Systems*, pages 23–39. Springer, 2023.

[109] Huanran Wang, Weizhe Zhang, and Hui He. You are what the permissions told me! android malware detection based on hybrid tactics. *Journal of Information Security and Applications*, 66:103159, 2022.

[110] Hyunjae Kang, Jae-wook Jang, Aziz Mohaisen, and Huy Kang Kim. Detecting and classifying android malware using static analysis along with creator information. *International Journal of Distributed Sensor Networks*, 11(6):479174, 2015.

[111] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, Yu Wang, and Yang Xiang. A3cm: automatic capability annotation for android malware. *IEEE Access*, 7:147156–147168, 2019.

[112] Suleiman Y Yerima, Sakir Sezer, Gavin McWilliams, and Igor Muttik. A new android malware detection approach using bayesian classification. In *2013 IEEE 27th international conference on advanced information networking and applications (AINA)*, pages 121–128. IEEE, 2013.

[113] Hui-Juan Zhu, Tong-Hai Jiang, Bo Ma, Zhu-Hong You, Wei-Lei Shi, and Li Cheng. Hemd: a highly efficient random forest-based malware detection framework for android. *Neural Computing and Applications*, 30:3353–3361, 2018.

[114] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622, 2013.

[115] Shuaihao Yang, Zigang Zeng, and Wei Song. Permdroid: automatically testing permission-related behaviour of android applications. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 593–604, 2022.

[116] Hanxun Zhou, Xinlin Yang, Hong Pan, and Wei Guo. An android malware detection approach based on simgru. *IEEE Access*, 8:148404–148410, 2020.

[117] Zahid Hussain Qaisar and Ruixuan Li. Multimodal information fusion for android malware detection using lazy learning. *Multimedia Tools and Applications*, pages 1–15, 2022.

[118] Wei Wang, Zhenzhen Gao, Meichen Zhao, Yidong Li, Jiqiang Liu, and Xiangliang Zhang. Droidensemble: Detecting android malicious applications with ensemble of string and structural static features. *IEEE Access*, 6:31798–31807, 2018.

[119] Saba Arshad, Munam A Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. Samadroid: a novel 3-level hybrid malware detection model for android operating system. *IEEE Access*, 6:4321–4339, 2018.

[120] ML Anupama, P Vinod, Corrado Aaron Visaggio, MA Arya, Josna Philomina, Rincy Raphael, Anson Pinhero, KS Ajith, and P Mathiyalagan. Detection and robustness evaluation of android malware classifiers. *Journal of Computer Virology and Hacking Techniques*, 18(3):147–170, 2022.

[121] Roopak Surendran, Tony Thomas, and Sabu Emmanuel. A tan based hybrid model for android malware detection. *Journal of Information Security and Applications*, 54:102483, 2020.

[122] Tianliang Lu, Yanhui Du, Li Ouyang, Qiuyu Chen, and Xirui Wang. Android malware detection based on a hybrid deep learning model. *Security and Communication Networks*, 2020:1–11, 2020.

[123] Mohammad Abuthawabeh and Khaled W Mahmoud. Enhanced android malware detection and family classification, using conversation-level network traffic features. *Int. Arab J. Inf. Technol.*, 17(4A):607–614, 2020.

[124] Jiayin Feng, Limin Shen, Zhen Chen, Yuying Wang, and Hui Li. A two-layer deep learning method for android malware detection using network traffic. *IEEE Access*, 8:125786–125796, 2020.

[125] Syed Ibrahim Imtiaz, Saif ur Rehman, Abdul Rehman Javed, Zunera Jalil, Xuan Liu, and Waleed S Alnumay. Deepamd: Detection and identification of android malware using high-efficient deep artificial neural network. *Future Generation computer systems*, 115:844–856, 2021.

[126] Stephen Feldman, Dillon Stadther, and Bing Wang. Manilyzer: automated android malware detection through manifest analysis. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 767–772. IEEE, 2014.

[127] Yung-Ching Shyong, Tzung-Han Jeng, and Yi-Ming Chen. Combining static permissions and dynamic packet analysis to improve android malware detection. In *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, pages 75–81, 2020.

[128] Chao Ding, Nurbol Luktarhan, Bei Lu, and Wenhui Zhang. A hybrid analysis-based approach to android malware family classification. *Entropy*, 23(8):1009, 2021.

[129] Md Shohel Rana and Andrew H Sung. Evaluation of advanced ensemble learning techniques for android malware detection. *Vietnam Journal of Computer Science*, 7(02):145–159, 2020.

[130] Kun Wang, Tao Song, and Alei Liang. Mmda: Metadata based malware detection on android. In *2016 12th International Conference on Computational Intelligence and Security (CIS)*, pages 598–602. IEEE, 2016.

[131] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882, 2014.

[132] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[133] Jason Papathanasiou, Nikolaos Ploskas, Jason Papathanasiou, and Nikolaos Ploskas. Topsis. *Multiple Criteria Decision Aid: Methods, Examples and Python Implementations*, pages 1–30, 2018.

[134] Neşe Yalçin and UNCU Nuşin. Applying edas as an applicable mcdm method for industrial robot selection. *Sigma Journal of Engineering and Natural Sciences*, 37(3):779–796, 2019.

[135] Ian H Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

[136] Tadayoshi Fushiki. Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21:137–146, 2011.

[137] Janani Thiyagarajan, A Akash, and Brindha Murugan. Improved real-time permission based malware detection and clustering approach using model independent pruning. *IET Information Security*, 14(5):531–541, 2020.

[138] Anshul Arora, Sateesh K Peddoju, and Mauro Conti. Permpair: Android malware detection using permission pairs. *IEEE Transactions on Information Forensics and Security*, 15:1968–1982, 2019.

[139] Kartik Khariwal, Jatin Singh, and Anshul Arora. Ipdroid: Android malware detection using intents and permissions. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 197–202. IEEE, 2020.

[140] Tanmoy Chakraborty, Fabio Pierazzi, and VS Subrahmanian. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing*, 17(2):262–277, 2017.

[141] Karim O Elish, Mahmoud O Elish, and Hussain MJ Almohri. Lightweight, effective detection and characterization of mobile malware families. *IEEE Transactions on Computers*, 71(11): 2982–2995, 2022.

[142] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security*, 13(8):1890–1905, 2018.

[143] Bo Sun, Takeshi Takahashi, Tao Ban, and Daisuke Inoue. Detecting android malware and classifying its families in large-scale datasets. *ACM Transactions on Management Information Systems (TMIS)*, 13(2):1–21, 2021.

[144] Yao Li, Dawei Yuan, Tao Zhang, Haipeng Cai, David Lo, Cuiyun Gao, Xiapu Luo, and He Jiang. Meta-learning for multi-family android malware classification. *ACM Transactions on Software Engineering and Methodology*, 2024.

[145] Francesco Mercaldo and Antonella Santone. Formal equivalence checking for mobile malware detection and family classification. *IEEE Transactions on Software Engineering*, 48(7):2643–2657, 2021.

[146] Qijing Qiao, Ruitao Feng, Sen Chen, Fei Zhang, and Xiaohong Li. Multi-label classification for android malware based on active learning. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[147] Yueming Wu, Shihan Dou, Deqing Zou, Wei Yang, Weizhong Qiang, and Hai Jin. Contrastive learning for robust android malware familial classification. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[148] Samah Alsoghyer and Iman Almomani. On the effectiveness of application permissions for android ransomware detection. In *2020 6th conference on data science and machine learning applications (CDMA)*, pages 94–99. IEEE, 2020.

[149] Gulshan Shrivastava and Prabhat Kumar. Sensdroid: analysis for malicious activity risk of android application. *Multimedia Tools and Applications*, 78(24):35713–35731, 2019.

[150] Fauzia Idrees, Muttukrishnan Rajarajan, Thomas M Chen, Yogachandran Rahulamathavan, and Ayesha Naureen. Andropin: Correlating android permissions and intents for malware detection. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 394–399. IEEE, 2017.

[151] Fauzia Idrees and Muttukrishnan Rajarajan. Investigating the android intents and permissions for malware detection. In *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 354–358. IEEE, 2014.

[152] Todd Michael Franke, Timothy Ho, and Christina A Christie. The chi-square test: Often used and more often misinterpreted. *American journal of evaluation*, 33(3):448–458, 2012.

[153] Durmuş Özkan Şahin, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kılıç. A novel permission-based android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, pages 1–16, 2021.

[154] Yash Sharma and Anshul Arora. Phigrader: Evaluating the effectiveness of manifest file components in android malware detection using multi-criteria decision making techniques. *Journal of Network and Computer Applications*, 232:104021, 2024.

[155] Yash Sharma and Anshul Arora. Ipanalyzer: A novel android malware detection system using ranked intents and permissions. *Multimedia Tools and Applications*, 83:78957—-79008, 2024.

[156] Arash Habibi Lashkari, Andi Fitriah A. Kadir, Laya Taheri, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. *In the proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST), Montreal, Quebec, Canada*, 2018.

[157] Laya Taheri, Andi F Abdulkadir, and Arash Habibi Lashkari. Extensible android malware detection and family classification using network-flows and api-calls. *The IEEE (53rd) International Carnahan Conference on Security Technology, India*, 2019.

[158] Arash Habibi Lashkari, Andi Fitriah A. Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A. Ghorbani. Towards a network-based framework for android malware detection and characterization. In *In the proceeding of the 15th International Conference on Privacy, Security and Trust, PST, Calgary, Canada,*, pages –, 2017.

[159] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International conference on information networking (ICOIN)*, pages 712–717. IEEE, 2017.

[160] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[161] Pallabi Borah, Hasin A Ahmed, and Dhruba K Bhattacharyya. A statistical feature selection technique. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 3:1–13, 2014.

[162] Priyakshi Mahanta, Hasin A Ahmed, Dhruba K Bhattacharyya, and Jugal K Kalita. An effective method for network module extraction from microarray data. *BMC bioinformatics*, 13(13):1–11, 2012.

[163] Lars St, Svante Wold, et al. Analysis of variance (anova). *Chemometrics and intelligent laboratory systems*, 6(4):259–272, 1989.

[164] Patrick E McKnight and Julius Najab. Mann-whitney u test. *The Corsini encyclopedia of psychology*, pages 1–1, 2010.

[165] Patrick E McKight and Julius Najab. Kruskal-wallis test. *The corsini encyclopedia of psychology*, pages 1–1, 2010.

[166] Anshul Arora and Sateesh K Peddoju. Ntpdroid: a hybrid android malware detector using network traffic and system permissions. In *2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*, pages 808–813. IEEE, 2018.

[167] Madan Upadhayay, Ashutosh Sharma, Gourav Garg, and Anshul Arora. Rpndroid: android malware detection using ranked permissions and network traffic. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pages 19–24. IEEE, 2021.

[168] Vikas Sihag, Surya Prakash, Gaurav Choudhary, Nicola Dragoni, and Ilsun You. Dimda: Deep learning and image-based malware detection for android. In *Futuristic Trends in Networks and Computing Technologies: Select Proceedings of Fourth International Conference on FTNCT 2021*, pages 895–906. Springer, 2022.

[169] Mohammed Alshehri. App-nts: a network traffic similarity-based framework for repacked android apps detection. *Journal of Ambient Intelligence and Humanized Computing*, 13(3):1537–1546, 2022.

[170] Alejandro Guerra-Manzanares, Hayretdin Bahsi, and Sven Nõmm. Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Computers & Security*, 110:102399, 2021.

[171] Jitesh J Thakkar. *Multi-criteria decision making*, volume 336. Springer, 2021.

[172] S Sathya Bama and A Saravanan. Efficient classification using average weighted pattern score with attribute rank based feature selection. *International Journal of Intelligent Systems and Applications*, 10(7):29, 2019.

[173] Jack V Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225–1231, 1996.

[174] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5-6):352–359, 2002.

[175] Sagar S Nikam. A comparative study of classification techniques in data mining algorithms. *Oriental Journal of Computer Science and Technology*, 8(1):13–19, 2015.

[176] NK Sreeja and A Sankar. Pattern matching based classification using ant colony optimization based feature selection. *Applied Soft Computing*, 31:91–102, 2015.

[177] Parnika Bhat, Sunny Behal, and Kamlesh Dutta. A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. *Computers & Security*, 130:103277, 2023.

[178] Durmuş Özkan Şahin, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kılıç. Permission-based android malware analysis by using dimension reduction with pca and lda. *Journal of Information Security and Applications*, 63:102995, 2021.

[179] Weizhe Zhang, Huanran Wang, Hui He, and Peng Liu. Damba: detecting android malware by orgb analysis. *IEEE Transactions on Reliability*, 69(1):55–69, 2020.

[180] Fatma Taher, Omar AlFandi, Mousa Al-kfairy, Hussam Al Hamadi, and Saed Alrabaee. Droid-detectmw: A hybrid intelligent model for android malware detection. *Applied Sciences*, 13(13): 7720, 2023.

[181] Ravi Mohan Sharma and Chaitanya P Agrawal. Mh-dldroid: A meta-heuristic and deep learning-based hybrid approach for android malware detection. *Int. J. Intell. Eng. Syst*, 15:425–435, 2022.

[182] Samaneh Mahdavifar, Dima Alhadidi, and Ali A Ghorbani. Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of network and systems management*, 30:1–34, 2022.

[183] Massimo Ficco. Malware analysis by combining multiple detectors and observation windows. *IEEE Transactions on Computers*, 71(6):1276–1290, 2021.

[184] Zhipeng Cai, Randy Goebel, Mohammad R Salavatipour, and Guohui Lin. Selecting dissimilar genes for multi-class classification, an application in cancer subtyping. *BMC bioinformatics*, 8: 1–15, 2007.

List of Publications

Journals Published

1. Yash Sharma and Anshul Arora. "PHIGrader: Evaluating the effectiveness of Manifest file components in Android malware detection using Multi Criteria Decision Making techniques." *Journal of Network and Computer Applications*, 232:104021, 2024.
   (SCIE, Impact Factor – 7.7)

2. Yash Sharma and Anshul Arora. "A comprehensive review on permissions-based Android malware detection." *International Journal of Information Security*, 23:1877–1912, 2024.
   (SCIE, Impact Factor – 2.5)

3. Yash Sharma and Anshul Arora. "IPAnalyzer: A novel Android malware detection system using ranked Intents and Permissions." *Multimedia Tools and Applications*, 83:8957–79008, 2024. **(SCOPUS, Impact Factor - 3)***

$*$ - SCIE at the time of publishing

Manuscripts Communicated

1. Yash Sharma and Anshul Arora. "CorrNetDroid: Android Malware Detector leveraging a Correlation-based Feature Selection for Network Traffic features." (Communicated)

2. Yash Sharma and Anshul Arora. "PattMatch: An Instance-based Pattern- Matching classifier for detecting Android malware using hybrid features." (Communicated)

Papers presented in Conferences

1. Yash Sharma and Anshul Arora. “AndroV-Rank: Optimized Android Malware Detection through VIKOR-Ranked Hybrid Features”, *2nd International Intelligent Computing and Technology Conference (ICTCon 2024)* at Central Institute of Technology Kokrajhar, Assam, India.

2. Yash Sharma and Anshul Arora. “Android Malware Categories through Dynamic System Calls Ranked via ReliefF”, *International Conference on Deep Learning, Artificial Intelligence and Robotics, (ICDLAIR) 2024* at National Institute of Technology Kurukshetra, India.

Journal of Network and Computer Applications 232 (2024) 104021

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Research paper

PHIGrader: Evaluating the effectiveness of Manifest file components in Android malware detection using Multi Criteria Decision Making techniques

Yash Sharma[1,*], Anshul Arora[1]

Department of Applied Mathematics, Delhi Technological University, Delhi, 110042, India

ARTICLE INFO

Keywords:
Android security
Malware detection
Permissions
Intents
Hardware components
Feature ranking

ABSTRACT

The popularity of the Android operating system has itself become a reason for privacy concerns. To deal with such malware threats, researchers have proposed various detection approaches using static and dynamic features. Static analysis approaches are the most convenient for practical detection. However, several patterns of feature usage were found to be similar in the normal and malware datasets. Such high similarity in both datasets' feature patterns motivates us to rank and select only the distinguishing set of features. Hence, in this study, we present a novel Android malware detection system, termed as *PHIGrader* for ranking and evaluating the efficiency of the three most commonly used static features, namely permissions, intents, and hardware components, when used for Android malware detection. To meet our goals, we individually rank the three feature types using frequency-based Multi-Criteria Decision Making (MCDM) techniques, namely TOPSIS and EDAS. Then, the system applies a novel detection algorithm to the rankings involving machine learning and deep learning classifiers to present the best set of features and feature type with higher detection accuracy as an output. The experimental results highlight that our proposed approach can effectively detect Android malware with 99.10% detection accuracy, achieved with the top 46 intents when ranked using TOPSIS, which is better than permissions, hardware components, or even the case where other popular MCDM techniques are used. Furthermore, our experiments demonstrate that the proposed system with frequency-based MCDM rankings is better than other statistical tests such as mutual information, Pearson correlation coefficient, and t-test. In addition, our proposed model outperforms various popularly used feature ranking methods such as Chi-square, Principal Component Analysis (PCA), Entropy-based Category Coverage Difference (ECCD), and other state-of-the-art Android malware detection techniques in terms of detection accuracy.

1. Introduction

In the present age and time, there exists an app for almost every diversified service required by man, such as online shopping, social networking, positioning, and navigation. As the statistics report, the Google Play Store, which now serves as an official app store for the Android operating system, has witnessed a huge rise in the number of applications over the span of 14 years. If we take a closer look at the numbers, it has grown from 16 thousand applications in 2009 to 3.553 million applications until 2023, i.e., a huge increase of 3.537 million.[2] Android dominates the market share with a whooping 68.79% of the total smartphones being used worldwide, followed by Apple iOS with around 30% share.[3] The openness and popularity of Android makes it the primary target of malicious attackers who attempt to steal private information, transfer credit into their account by subscribing to premium services, start unwarranted premium-rate subscriptions of SMS services, or even commit advanced frauds.

Requirement of Android Malware Detection Systems

In simple words, mobile malware can be defined as any type of malicious code designed specifically to disrupt the functionality and integrity of a mobile system without the user's consent. The "Quick Heal Annual Threat Report 2022" shows that there were 1,11,894 malware detections in 2022, which accounts for 1 malware per minute[4] and these numbers are expected to steadily grow in the coming years, especially due to the trend of mobile banking and electronic payment, to perform various illegal acts such as malicious charges, system damages, and privacy breaches. The most common malware types include

* Corresponding author.
E-mail addresses: ysharma2098@gmail.com (Y. Sharma), anshul15arora@gmail.com (A. Arora).
[1] These authors contributed equally to this work.
[2] https://www.bankmycell.com/blog/number-of-google-play-store-apps/
[3] https://gs.statcounter.com/os-market-share/mobile/worldwide
[4] https://www.quickheal.co.in/documents/threat-report/

https://doi.org/10.1016/j.jnca.2024.104021
Received 30 October 2023; Received in revised form 5 August 2024; Accepted 4 September 2024
Available online 6 September 2024

Refine Your Search Results

Journal of network and computer applications

Search

Search Results

Found 2,651 results (Page 1)

Share These Results

Exact Match Found

JOURNAL OF NETWORK AND COMPUTER APPLICATIONS

Publisher: **ACADEMIC PRESS LTD- ELSEVIER SCIENCE LTD , 24-28 OVAL RD, LONDON, ENGLAND, NW1 7DX**

ISSN / eISSN: **1084-8045 / 1095-8592**

Web of Science Core Collection: **Science Citation Index Expanded**

Additional *Web of Science* Indexes: **Essential Science Indicators**

Share This Jou

International Journal of Information Security
https://doi.org/10.1007/s10207-024-00822-2

REGULAR CONTRIBUTION

A comprehensive review on permissions-based Android malware detection

Yash Sharma[1] · Anshul Arora[1]

Abstract
The first Android-ready "G1" phone debuted in late October 2008. Since then, the growth of Android malware has been explosive, analogous to the rise in the popularity of Android. The major positive aspect of Android is its open-source nature, which empowers app developers to expand their work. However, authors with malicious intentions pose grave threats to users. In the presence of such threats, Android malware detection is the need of an hour. Consequently, researchers have proposed various techniques involving static, dynamic, and hybrid analysis to address such threats to numerous features in the last decade. However, the feature that most researchers have extensively used to perform malware analysis and detection in Android security is Android permission. Hence, to provide a clarified overview of the latest and past work done in Android malware analysis and detection, we perform a comprehensive literature review using permissions as a central feature or in combination with other components by collecting and analyzing 205 studies from 2009 to 2023. We extracted information such as the choice opted by researchers between analysis or detection, techniques used to select or rank the permissions feature set, features used along with permissions, detection models employed, malware datasets used by researchers, and limitations and challenges in the field of Android malware detection to propose some future research directions. In addition, on the basis of the information extracted, we answer the six research questions designed considering the above factors.

Keywords Android security · Android malware · Permissions based detection · Static detection · Mobile security · Literature review

1 Introduction

In the last decade, we have witnessed exponential growth of the Android operating system in the mobile market. According to a recent report, the Android system constitutes more than 80% of the entire market of smart phones.[1] The main reason behind Android's success is its free, open-source code, which empowers smartphone manufacturers to transform their devices with pre-installed applications and customized user interfaces for a beautiful customer experience. However, Android's open-source nature is both a boon and a bane. Although it brings the benefits of technological broadband and updates, it also allows criminals to use it for ill practices. Nowadays, mobile phones are not only used for communication purposes, but they have gradually become a crucial part of our lives, containing the smallest to the most critical and private user data.

The Android OS was released in 2008, and the first Android malware was spotted in 2010, which targeted users by subscribing to premium SMS services. Since then, malware attacks have been on the rise, and security attempts have been made to keep up with the ever-increasing and constantly changing malicious attacks. The total amount of Android malware worldwide has already increased from 22,088 in 2012 to 33,237,653 in January 2023.[2] Looking closer at the real-time threat analysis and statistics of Android malware worldwide, we will understand how desperately the Android Market needs Android security and malware detection systems. For instance, the **Judy** auto-clicking adware stands out as a significant incident that affected the Google

[1] https://www.tenda.com.cn.

[2] https://portal.av-atlas.org/malware.

Yash Sharma
ysharma2098@gmail.com

Anshul Arora
anshul15arora@gmail.com

[1] Department of Applied Mathematics, Delhi Technological University, Delhi 110042, India

Published online: 04 March 2024

Refine Your Search Results

International Journal of information security

Search

So

Search Results

Found 3,741 results (Page 1)

Share These Results

Exact Match Found

INTERNATIONAL JOURNAL OF INFORMATION SECURITY

Publisher: **SPRINGER , ONE NEW YORK PLAZA, SUITE 4600 , NEW YORK, United States, NY, 10004**

ISSN / eISSN: **1615-5262 / 1615-5270**

Web of Science Core Collection: **Science Citation Index Expanded**

Additional *Web of Science* Indexes: **Current Contents Engineering, Computing & Technology | Essential Science Indicators**

Share This Jou

Multimedia Tools and Applications
https://doi.org/10.1007/s11042-024-18511-6

IPAnalyzer: A novel Android malware detection system using ranked Intents and Permissions

Yash Sharma[1] · **Anshul Arora**[1]

Received: 31 July 2023 / Revised: 2 October 2023 / Accepted: 29 January 2024

Abstract

Android malware has been growing in scale and complexity, spurred by the unabated uptake of smartphones worldwide. Millions of malicious Android applications have been detected in the past few years, posing severe threats like system damage, information leakage, etc. This calls for novel approaches to mitigate the growing threat of Android malware. Among various detection schemes, permission and intent-based ones have been widely proposed in the literature. However, many permissions and intents patterns are similar in normal and malware datasets. Such high similarity in both datasets' permissions and intents patterns motivates us to rank them to find the distinguishing features. Hence, we have proposed a novel Android malware detection system named IPAnalyzer that first ranks the permissions and intents with a frequency-based Chi-square test. Then, the system applies a novel detection algorithm that combines ranked permissions and intents and involves various machine learning and deep learning classifiers. As a result, the proposed system gives the best set of permissions and intents with higher detection accuracy as an output. The experimental results highlight that our proposed approach can effectively detect Android malware with 98.49% detection accuracy, achieved with the combination of the top six permissions and top six intents. Furthermore, our experiments demonstrate that the proposed system with the Chi-square ranking is better than other statistical tests like Mutual Information and Pearson Correlation Coefficient. Moreover, the proposed model can detect Android malware with better accuracy and less number of features than various state-of-the-art techniques for Android malware detection.

Keywords Android security · Mobile malware · Malware detection · Permissions · Intents · Feature selection

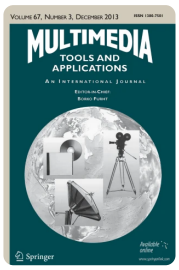
Anshul Arora contributed equally to this work.

Yash Sharma
ysharma2098@gmail.com

Anshul Arora
anshul15arora@gmail.com

[1] Department of Applied Mathematics, Delhi Technological University, Delhi -110042, India

Published online: 01 March 2024



Multimedia Tools and Applications

An International Journal

Publishing model

Hybrid

Overview

Multimedia Tools and Applications publishes original research on multimedia development and system support tools as well as case studies of multimedia applications.

- Recognized as the first journal in the field of multimedia.
- Boasts the highest Google h5-index score in the field of multimedia.
- Features case studies alongside experimental and survey articles.

Journal metrics

Journal Impact Factor
3.0 (2023)

5-year Journal Impact Factor
2.9 (2023)

Submission to first decision (median)
66 days

Downloads
4.1M (2024)

Guest Editors: Lucia Cimmino, Matteo Polsinelli, David Freire-Obregon, Giuseppe Placidi

View all updates →

Journal information

Electronic ISSN
1573-7721

Abstracted and indexed in
ACM Digital Library
ANVUR
BFI List
Baidu
CLOCKSS
CNKI
CNPIEC
DBLP
Dimensions
EBSCO
EI Compendex
Google Scholar
INSPEC
Japanese Science and Technology Agency (JST)
Naver
OCLC WorldCat Discovery Service
Portico
ProQuest
SCImago
SCOPUS
TD Net Discovery Service
Wanfang

Yash Sharma

ysharma2098@gmail.com
+91-9013562595
https://scholar.google.com/citations?user=b__PpOIAAAAJ&hl=en

Education

2021 – 2025 **Ph.D. in Android Malware Detection** from Delhi Technological University, Bawana, Delhi, India
Thesis title: *Design and Development of malware detection models for Android smartphones*

2019 – 2021 **M.Sc. Mathematics** from Delhi Technological University DTU, Delhi, India
7.51 CGPA - 75.10 %

2016 – 2019 **B.Sc.(Hons.) Mathematics** from Keshav Mahavidyalaya, University of Delhi, India
7.203 CGPA - 68.42 %

2002 – 2016 **Schooling** from CBSE board (Batch – 2016) Greenfields Public School Delhi, India
Sr Secondary : 87.8 %
Secondary : 10 CGPA - 95 %

Achievements

2021 Qualified **CSIR- UGC NET JRF 2021 in Mathematical Sciences with AIR 110**
Qualified **GATE Exam 2021 in MA with AIR 67 (Gate score -677)**

Research Publications

Journal Articles

1. Y. Sharma and A. Arora, "Phigrader: Evaluating the effectiveness of manifest file components in android malware detection using multi criteria decision making techniques," *Journal of Network and Computer Applications, Elsevier (SCIE)*, vol. 232, p. 104 021, 2024.
2. Y. Sharma and A. Arora, "A comprehensive review on permissions-based android malware detection," *International Journal of Information Security, Springer, (SCIE)*, vol. 23, pp. 1877–1912, 2024.
3. Y. Sharma and A. Arora, "Ipanalyzer: A novel android malware detection system using ranked intents and permissions," *Multimedia Tools and Applications, Springer, (SCIE)*, vol. 83, pp. 78 957–79 008, 2024.

Conference Proceedings

1. Y. Sharma and A. Arora, "Classifying android malware categories through dynamic system calls ranked via relieff," in *Proceedings of the 6th International Conference on Deep Learning, Artificial Intelligence and Robotics (ICDLAIR 2024)*, Springer Nature, vol. 193, 2025, p. 53.

2 Y. Sharma and A. Arora, "Androv-rank: Optimized android malware detection through vikor-ranked hybrid features," in *accepted for publication in proceedings of the 2nd International Intelligent Computing and Technology Conference (ICTCON 2024)*, Springer, 2024.

3 Y. Sharma, S. Sharma, and A. Arora, "Feature ranking using statistical techniques for computer networks intrusion detection," in *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, IEEE, 2022, pp. 761–765.

Skills

Data Curation	Extensive experience in curating large-scale Android application datasets (over 150,000 apps), including collection, preprocessing, and extraction of both static features (permissions, intents, hardware components) and dynamic features (network traffic, TCP packet flows).
Languages	Strong reading, writing, and speaking competencies for English and Hindi
Coding	Python, C, C++, , JAVA, LaTeX, Excel
Misc.	Academic research, teaching, LaTeX typesetting and publishing.

References

Dr. Anshul Arora
Assistant Professor
Department of Applied Mathematics
Delhi Technological University, Delhi, India
Email: anshul15arora@gmail.com
Phone: +91-94165-48025

Prof. Aditya Kaushik
Professor
Department of Applied Mathematics
Delhi Technological University, Delhi, India
Email: akaushik@dtu.ac.in
Phone: +91-84275-80104